PARSQL: Enhancing Text-to-SQL through SQL Parsing and Reasoning

Yaxun Dai[1*], **Haiqin Yang**[2†], **Mou Hao**[3], **Pingfu Chao**[1†]

[1]Soochow University, [2]International Digital Economy Academy (IDEA), [3]DataStory Ltd.
yaxundai@gmail.com, haiqin.yang@gmail.com
mouhao@datastory.com.cn, pfchao@suda.edu.cn

Abstract

Large language models (LLMs) have made significant strides in text-to-SQL tasks; however, small language models (SLMs) are crucial due to their low resource consumption and efficient inference for real-world deployment. Due to resource limitations, SLMs struggle to accurately interpret natural language questions and may overlook critical constraints, leading to challenges such as generating SQL with incorrect logic or incomplete conditions. To address these issues, we propose PARSQL, a novel framework that leverages SQL parsing and reasoning. Specifically, we design PARSer, an SQL parser that extracts constraints from SQL to generate sub-SQLs for data augmentation and producing step-by-step SQL explanations (reason) via both rule-based and LLM-based methods. We define a novel text-to-reason task and incorporate it into multi-task learning, thereby enhancing text-to-SQL performance. Additionally, we employ an efficient SQL selection strategy that conducts direct similarity computation between the generated SQLs and their corresponding reasons to derive the final SQL for post-correction. Extensive experiments show that our PARSQL outperforms models with the same model size on the BIRD and Spider benchmarks. Notably, PARSQL-3B achieves 56.98% execution accuracy on BIRD, rivaling 7B models with significantly fewer parameters, setting a new state-of-the-art performance. Code can be found here.

1 Introduction

Text-to-SQL, or NL2SQL, is a longstanding and pivotal task focusing on transforming natural language (NL) questions into executable SQLs [1], streamlining database interactions for non-experts and significantly enhancing information retrieval efficiency (Deng et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023; Liu et al., 2024a). The primary challenge lies in converting the flexible and diverse expressions of NL into standardized formal SQLs (Zhang et al., 2024b).

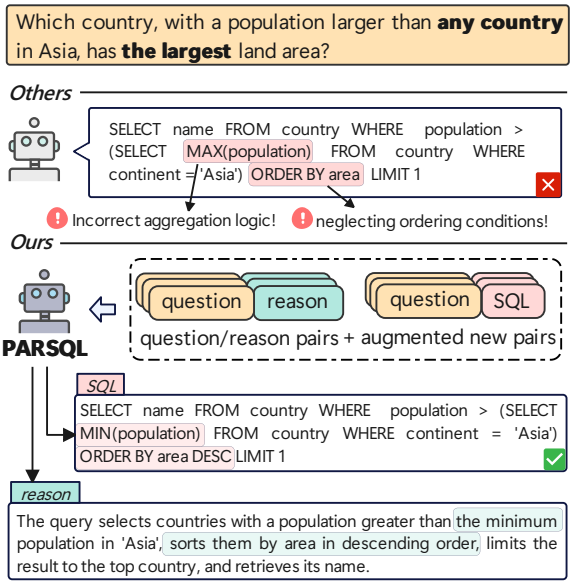


Figure 1: Common errors in SLMs and their mitigation using PARSQL. SLMs often overlook implicit constraints, such as the descending order (DESC) implied by "the largest", and misinterpret semantic nuances, like using MAX(population) instead of MIN(population) due to an incorrect understanding of "larger than any country". PARSQL employs new question/reason pairs and augmented pair data to train the model, enhancing query constraint sensitivity and reducing the semantic gap between questions and the generated SQLs.

Recently, methods leveraging closed-source (Talaei et al., 2024; Li et al., 2024a; Gao et al., 2024; Luo et al., 2024) and open-source (Li et al., 2024b; Dai et al., 2025) large language models (LLMs) have proliferated, consistently advancing performance, making this difficulty less of a limitation [2]. While research predominantly focuses on

[*]Work done during an internship at IDEA.
[†]Corresponding authors.
[1]For brevity, we use "SQLs" to refer to SQL statements and "sub-SQLs" to refer to sub-SQL statements.
[2]https://bird-bench.github.io/

Findings of the Association for Computational Linguistics: ACL 2025, pages 661–681

LLMs (Liu et al., 2024a,b), small language models (SLMs) offer comparable applicability in practical scenarios, with advantages such as low resource consumption and ease of local deployment. These traits make SLMs well-suited for resource-limited and privacy-sensitive contexts (Fan et al., 2024). However, due to their smaller model sizes, SLMs still encounter challenges in converting free-form NL into constrained and formal SQL statements.

Figure 1 highlights two types of errors: SLMs often overlook implicit constraints in NL questions, such as missing conditions (Li et al., 2023a; Scholak et al., 2021), ordering rules (e.g., descending order implied by the largest"). Additionally, the generated SQLs may be syntactically similar but logically incorrect, such as misinterpreting larger than any country" or incorrectly using `MAX(population)` instead of `MIN(population)`. These issues arise due to the semantic gap between flexible and diverse NL and highly structured SQL.

Some methods have employed data augmentation to enhance the model's semantic understanding of NL (Zhang et al., 2024a; Li et al., 2024b; Yang et al., 2024). The expansion of the corpus aims to improve the model's performance and reduce instances of overlooked constraints. However, data augmentation mainly focuses on diversifying training data and improving domain adaptation (Yang et al., 2024), which remains insufficient for capturing constraint differences in NL. Moreover, to bridge the semantic gap between free-form NL and structured and formal SQL, many studies (Pourreza and Rafiei, 2023; Talaei et al., 2024; Zhang et al., 2023a) have integrated Chain-of-Thought (CoT) (Chu et al., 2024) in LLMs, generating intermediate reasoning steps to mitigate errors in direct mapping. However, CoT often proves ineffective in SLMs and may exacerbate errors (Wei et al., 2022), leading to inaccurate SQL generation.

To address the challenges of constraint differentiation and semantic alignment in SLMs, we propose PARSQL, a novel framework that combines SQL parsing and reasoning. Specifically, PARSQL introduces an effective SQL parser, PARSer, to decompose SQLs into abstract syntax trees (ASTs) (Wang et al., 1997), yielding constraints, sub-SQLs, and reasoning paths. The parsed results enable creating augmented "new pairs" (question/SQL pairs) that resemble the original data without certain constraints and "reasons" to provide step-by-step SQL generation explanations as CoT reasoning. Subsequently, we introduce an auxiliary task, "text-to-reason" (NL2Reason), and apply multi-task learning to bridge the semantic gap between NL and SQLs. Finally, PARSQL employs an efficient SQL selection strategy that directly computes the similarity between the generated SQLs and their corresponding reasons to yield the final SQL for post-correction without requiring additional training.

We summarize our contributions as follows:

- We propose PARSQL to improve text-to-SQL in SLMs by integrating SQL parsing and reasoning. It introduces innovative data augmentation, multi-task learning for enhanced reasoning and constraint recognition, and an efficient SQL selection strategy for post-correction.
- We introduce PARSer, which decomposes SQL into sub-SQLs and reasoning paths. It constructs question/sub-SQL pairs and provides step-by-step explanations based on rules.
- We show that PARSQL boosts SLM performance. PARSQL-3B achieves 56.98% execution accuracy, matching 7B models with fewer parameters. It improves constraint sensitivity by up to 5.83% and semantic similarity by 12.20%.

2 Related Work

2.1 Text-to-SQL with SLMs

Small Language Models (SLMs), typically with fewer than 10B parameters (Nguyen et al., 2024), have received less attention in text-to-SQL tasks compared to LLMs (Liu et al., 2024a). However, they offer advantages like lower resource consumption and broader applicability (Nguyen et al., 2024; Fan et al., 2024). SLMs also outperform pre-trained models by leveraging larger corpora (Li et al., 2024b). The CodeS introduced open-source models ranging from 1B to 15B parameters, advancing SLMs in text-to-SQL (Li et al., 2024b). Additionally, distillation or task-specific fine-tuning can further enhance SLM performance (Hsieh et al., 2023; Kang et al., 2023; Fan et al., 2024). We propose a novel multi-task learning approach, using step-by-step SQL explanation as an auxiliary task to improve SLM performance in text-to-SQL.

2.2 Data Augmentation

Many methods leverage LLMs to generate synthetic data (Yang et al., 2024; Li et al., 2024b; Zhang et al., 2024a). SENSE (Yang et al., 2024) employs both strong and weak LLMs to generate preference data, improving domain generalization, while CodeS (Li et al., 2024b) introduces a

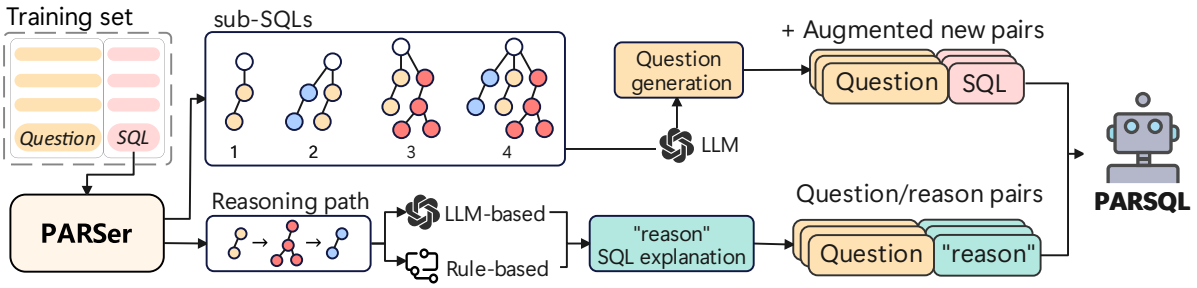


Figure 2: Data preparation and multi-task learning for PARSQL.

bi-directional augmentation technique for domain adaptation. Fin-SQL (Zhang et al., 2024a) uses LLMs to expand data, addressing the challenge of limited training data. Differently, we focus on SQL parsing to generate authentic sub-SQLs for questions generation, with emphasis on enhancing the model's ability to identify constraint differences.

2.3 Text-to-SQL with Abstract Syntax Trees

The Abstract Syntax Tree (AST) (Wang et al., 1997) provides a foundational structure for parsing and generating SQL queries. Early grammar-based decoders, such as RAT-SQL (Wang et al., 2020) and IRNet (Guo et al., 2019a), introduced tree-structured methods for generating ASTs and parsing them into SQL. While these approaches (Guo et al., 2019b; Zhang et al., 2023b) emphasize syntactic information, they fail to preserve reasoning information. Our work revisits ASTs, demonstrating their dual role as both a syntactic scaffold and a structured representation of reasoning steps. By decomposing ASTs into sub-SQL components and tracking their transformations, we develop rule-based methods that generate NL explanations aligned with these transformations. Unlike template-based approaches limited to predefined syntax patterns (Elgohary et al., 2020, 2021), our method systematically covers all syntactic structures in datasets such as BIRD and Spider. These explanations serve as "reasons" for data augmentation, improving semantic alignment.

3 Methodology

Problem Definition Formally, given an NL question Q and a database $\mathcal{D}$ with schema $\mathcal{S}$, the text-to-SQL task aims to translate Q into an SQL query y that can be executed on $\mathcal{D}$ to answer the question Q. The database $\mathcal{D}$ contains the schema $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{R})$ of a triplet, which includes table names $\mathcal{T}$, column names $\mathcal{C}$ and foreign key relations $\mathcal{R}$.

Overall PARSQL consists of three main steps: First, the data preparation step utilizes PARSer to generate "new pairs" and "reason pairs". Rule-based and LLM-based methods are employed to generate reasons describing the step-by-step SQL generation process. Second, multi-task learning is applied to enhance PARSQL's sensitivity to constraints in the questions using "new pairs", while leveraging "reason pairs" to aid in understanding the underlying logic of SQL generation. Finally, during inference, PARSQL generates two outputs: SQL statements and their corresponding reasons. An efficient selection strategy is employed to choose the final SQL with the highest score, calculated based on the similarity of SQL and its reason. Figure 2 illustrates the first two steps, while Figure 4 demonstrates the inference process.

3.1 PARSer for SQL Parsing and Reasoning

We designed PARSer to construct "new pairs" and "reason pairs" data, which decomposes an SQL into sub-SQLs and translates an SQL using a rule-based method, as shown in Figure 3.

Decomposing SQL into sub-SQLs Specifically, PARSer begins by parsing the SQL query into an Abstract Syntax Tree (AST), e.g., using SQLGlot (Mao, 2024) or Apache Calcite (Begoli et al., 2018). It then traverses the AST to extract constraints based on the type of each node. A constraint is defined as a subtree within the AST, where the root is an operator node, and all children are non-operator nodes; see Appendix A.1.2 for the definition of node types. As illustrated in Fig. 3, in the `WHERE` clause, there are two constraints: `age=18` and `sex=F`, each having an `Equal` operation node as the root, with children nodes of non-operation type. By isolating individual constraints from the AST, different sub-SQL queries can be generated. For instance, removing the constraint `age=18` and reconstructing the remaining AST results in a sub-

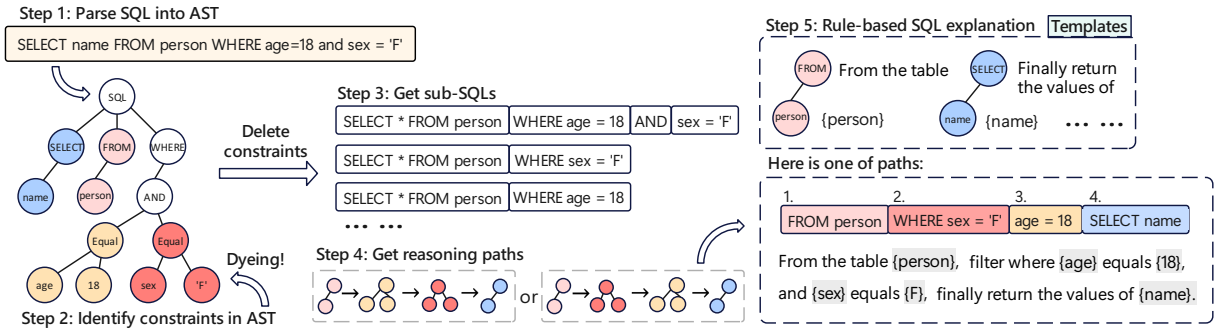


Figure 3: Illustration of PARSer 's sub-SQLs splitting and rule-based SQL explanation process. Each constraint in the AST is highlighted in a different color.

SQL: `SELECT name FROM person WHERE sex=F`.

Rule-based SQL Translation Since SQL engines follow a specific order when executing the clauses of a query[3], we traverse the constraints in the order of `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, and `SELECT` to align with this execution sequence. This approach allows us to explore different reasoning paths, enabling the step-by-step parsing of SQL queries diversely. For example, in the `WHERE` clause, there are two parallel constraints, yielding two different sequences of processing these constraints. We refer to this order of traversal as the reasoning path. For each node in the AST, we define an explanation template; detailed in Appendix A.1.3. Guided by the reasoning path, we traverse each AST node and translate it into a NL description based on the corresponding template. For example, the template for a `FROM` node is "`From the table {}`", where `{}` represents the node's content. Similarly, the template for a `WHERE` node is "`filter where {}`", and for an `Equal` node, the template is "`{} equals {}`".

3.2 Augmented Data Generation

After obtaining constraints, sub-SQLs, and rule-based generated reasons, via parsing an SQL via PARSer, we generate two types of augmented data:

- **Augmented (question, SQL) pairs:** A pair consists of a question generated from a given parsed sub-SQL using a prompt template, as shown in Fig. 9. The question resembles the original but omits certain constraints from the original SQL.
- **(question, reason) pairs:** In a pair, the question is given from the original question/SQL pair, and the generated "reason" is a step-by-step explanation of the SQL. In addition to the aforementioned rule-based method for generating "reason,", we also employ an LLM-based approach, utilizing a prompt template in Fig. 11.

3.3 Multi-task Learning

We apply multi-task learning to train PARSQL on two tasks: NL2SQL and NL2Reason, aiming to enhance the model's differentiation ability. For a question Q, we first follow Li et al. (2024b) to construct the database prompts (DP) and prepare the corresponding task's input by a prefix token:

$$x_{\mathrm{P}} = [\mathrm{P}] + \mathrm{DP} + Q \quad (1)$$

where $\mathrm{P} = \mathrm{SQL}$ for NL2SQL and $\mathrm{P} = \mathrm{REASON}$ for NL2Reason. We then apply multi-task learning for supervised fine-tuning (Hsieh et al., 2023) using LoRA (Hu et al., 2022a), minimizing the overall loss (here denote the loss on a single instance):

$$L = -\sum_{i=1}^{|y^{\mathrm{SQL}}|} \log(P_G(y_i^{\mathrm{SQL}}|y_{<i}^{\mathrm{SQL}}, x_{\mathrm{SQL}}) - \lambda \sum_{i=1}^{|y^{\mathrm{REASON}}|} \log(P_G(y_i^{\mathrm{REASON}}|y_{<i}^{\mathrm{REASON}}, x_{\mathrm{REASON}}) \quad (2)$$

where P_G represents the conditional probability of PARSQL generating the outputs, λ is a hyperparameter balanceing the loss on the NL2SQL and NL2Reason tasks. We train separately on rule-based and LLM-based reason pairs, resulting in two variants: PARSQL$_{rule}$ and PARSQL$_{llm}$.

3.4 SQL Selection

PARSQL incorporates an efficient SQL selection strategy for post-correction. As illustrated in Fig. 4, during inference, PARSQL first generates several SQLs and the corresponding reasons. The SQLs are then converted into NL explanations using a rule-based method, and compared with the reasons to compute the N-gram similarity, without requir-

[3] https://sqlglot.com/sqlglot.html

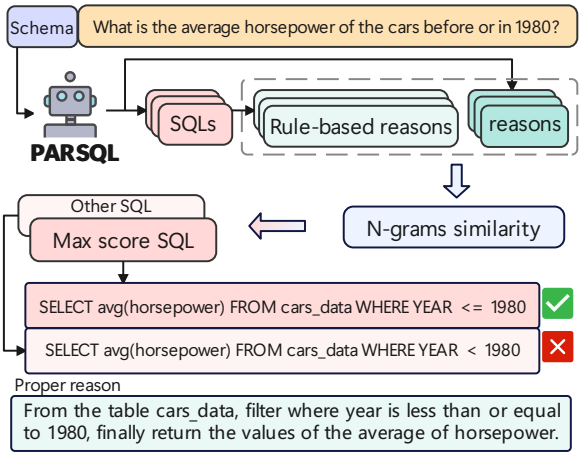


Figure 4: SQL selection strategy based on N-gram similarity between SQL and reason in PARSQL.

ing additional training. The SQL with the highest similarity is selected as a result. This strategy enhances the efficiency of post-correction using reasons, especially for outputs that are semantically similar but logically flawed.

4 Experiments

4.1 Experimental Settings

Baselines For supervised fine-tuning, nearly all baselines are derived from the state-of-the-art (SOTA) text-to-SQL approaches listed on the official leaderboards of the BIRD and Spider benchmarks (Li et al., 2023a, 2024b; Yang et al., 2024; Li et al., 2023b; Scholak et al., 2021). We select SFT CodeS as our primary competitive baseline and present the results obtained by re-running the provided checkpoints and codes (Li et al., 2024b). Additionally, we compare PARSQL with strong LLM-based methods (Pourreza and Rafiei, 2023; Gao et al., 2024; Li et al., 2024a; Luo et al., 2024).

In the SQL selection strategy, we apply two baselines: (1) The first-valid-execution method selects the first executable SQL generated by the model (Li et al., 2024b, 2023a). (2) The execution-guided self-consistency method selects the SQL with the highest consistency in execution results through a majority vote (Liu et al., 2024a,b).

Datasets We conduct experiments on two popular text-to-SQL benchmarks: BIRD (Li et al., 2023c) and Spider (Yu et al., 2018). We also assess robustness using three variants: Spider-DK (Gan et al., 2021b), Spider-Syn (Gan et al., 2021a), and Spider-Realistic (Deng et al., 2021). BIRD includes 9,428 training and 1,534 development samples, while Spider has 8,659 training and 1,034 development samples. Details of the datasets are provided in Appendix A.4.1.

Methods	BIRD Dev	
	EX (%)	VES (%)
Prompting Methods w/ Closed-Source LLMs		
CHASE-SQL + Gemini (Pourreza et al., 2024)	**73.01**	73.0/-
CHESS + GPT-4 (Talaei et al., 2024)	65.00	-
PTD-SQL + GPT-4 (Luo et al., 2024)	57.0	57.7/-
SuperSQL + GPT4 (Li et al., 2024a)	58.5	61.99/-
DAIL-SQL + GPT-4 (Gao et al., 2024)	54.76	56.08 / -
Fine-tuning Models w/ Open-Source LLMs		
RESDSQL-3B + NatSQL (Li et al., 2023a)	43.9	45.64 / -
SFT Llama2-7B (Li et al., 2024b)	45.37	46.98 / -
SENSE-7B (Yang et al., 2024)	51.8	-
SFT CodeS-7B (Li et al., 2024b)	57.00	58.80 / **72.54**
SFT CodeS-1B (Li et al., 2024b)	49.54	51.07 / 62.49
PARSQL-1B$_{rule}$	51.69 (+2.15)	67.71 (+5.22)
PARSQL-1B$_{llm}$	51.76 (+2.22)	67.96 (+5.47)
SFT CodeS-3B (Li et al., 2024b)	55.02	56.54 / 70.96
PARSQL-3B$_{rule}$	56.00 (+0.98)	71.67 (+0.71)
PARSQL-3B$_{llm}$	56.98 (+1.96)	72.43 (+1.47)

Table 1: Comparison on BIRD Dev: "-/-" in the VES column indicates that the results are copied from the original paper and reproduced by us. Values in parentheses record PARSQL improvement over SFT CodeS with the same model size[4].

Metrics For the BIRD benchmark, we utilize the official evaluation script[5], which includes the Execution Accuracy (EX) and Valid Efficiency Score (VES). EX measures whether the generated SQL produces the same execution result as the ground truth SQL. VES is calculated as the ratio of the execution time of the ground truth SQL to that of the predicted SQL. For the Spider benchmark, we adhere to the official evaluation protocol and utilize EX and test-suite accuracy (TS) (Zhong et al., 2020) metrics[6], where TS is a more reliable metric to verify whether a SQL query consistently passes all EX checks across various tests generated through database augmentation.

Implementation Details All the experiments are run on a server with 8 NVIDIA RTX 3090 GPUs and an AMD EPYC 7742 CPU of 128 GB memory. Details of each step are provided below: **(1) Data construction:** The BIRD and Spider training sets are parsed by PARSer to extract sub-SQLs and rule-based (question, reason) pairs. We utilize GLM-4-0520 (Zeng et al., 2024) to generate aug-

[4]This also applies to the results in Table 2 and Table 3.

[5]https://bird-bench.github.io/

[6]https://yale-lily.github.io/spider

Methods	Spider Dev	
	EX (%)	TS (%)
Prompting Methods w/ Closed-Source LLMs		
PURPLE + GPT-4 (Ren et al., 2024)	**87.8**	**83.3**
PTD-SQL + GPT-4 (Luo et al., 2024)	85.7	-
SuperSQL + GPT4 (Li et al., 2024a)	87.0	-
DIN-SQL+GPT-4 (Pourreza and Rafiei, 2023)	82.8	74.2
DAIL-SQL + GPT-4 (Gao et al., 2024)	83.1	76.6
Fine-tuning Models w/ Open-Source LLMs		
RESDSQL-3B + NatSQL (Li et al., 2023a)	84.1	73.5
Graphix-T5-3B + PICARD (Li et al., 2023b)	81.0	75.0
T5-3B + PICARD (Scholak et al., 2021)	79.3	69.4
SFT Llama2-7B (Li et al., 2024b)	77.8	73.0
SENSE-7B (Yang et al., 2024)	83.2	81.7
SFT CodeS-7B (Li et al., 2024b)	84.7	79.4
SFT CodeS-1B (Li et al., 2024b)	77.8	71.2
PARSQL-1B$_{rule}$	80.6 (+2.8)	73.8 (+2.6)
PARSQL-1B$_{llm}$	80.2 (+2.4)	73.7 (+2.5)
SFT CodeS-3B (Li et al., 2024b)	82.2	76.3
PARSQL-3B$_{rule}$	83.5 (+1.3)	77.3 (+1.1)
PARSQL-3B$_{llm}$	83.8 (+1.6)	77.7 (+1.4)

Table 2: Comparison on Spider Dev.

mented "new pairs" and LLM-based "reason pairs". **(2) Schema linking:** The database prompt is constructed following SFT CodeS (Li et al., 2024b) because the schema filter in SFT CodeS achieves exceptional performance on BIRD, allowing to obtain the relevant database schema and values, along with additional metadata. **(3) Training:** We fine-tune models using LoRA (Hu et al., 2022b) on CodeS-1B and CodeS-3B (Li et al., 2024b). The maximum input length is set to 4,096. The learning rate is set to 1e-4 with a cosine warmup scheduler. Training is conducted for 10 epochs with a batch size of 8 and λ of 0.8. **(4) Inference:** We set the beam size to 4 and limited the maximum output token length to 256 for both SQL and reason generation. For PARSQL$_{rule}$, we apply SQL selection based on N-grams. For PARSQL$_{llm}$, we adopt the first-valid-execution strategy.

4.2 Main Results

Evaluation on In-domain Benchmarks Table 1 reports the results of compared methods on BIRD Dev, highlighting the following: (1) Both PARSQL$_{rule}$ and PARSQL$_{llm}$ outperform SFT CodeS with the same model sizes, specifically 1B and 3B. Notably, PARSQL-3B$_{llm}$ improves the EX score by 1.96% over SFT CodeS-3B, matching SFT CodeS-7B's performance with only a 0.02% difference. (2) PARSQL-3B surpasses closed-source baselines like DIN-SQL and DAIL-SQL, which rely on GPT-4, establishing a new SOTA for small-parameter models on complex BIRD tasks. Table 2 demonstrates similar improvements on

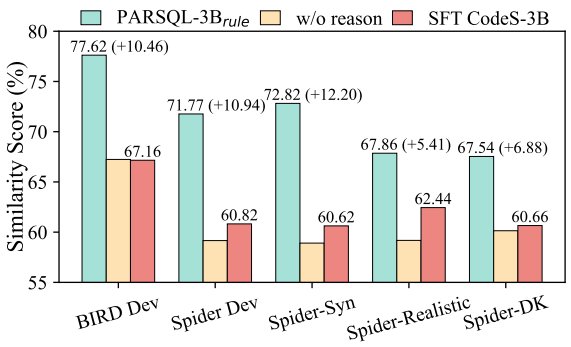


Figure 5: Comparison of similarity between questions and SQLs on five datasets.

Spider Dev, a simpler dataset. PARSQL-1B$_{rule}$ achieves a 2.8% increase in EX and a 2.6% improvement in TS over SFT CodeS-1B, indicating fewer minor errors. Overall, $PARSQL$ performs exceptionally well on both complex and simple tasks. Additionally, PARSQL$_{rule}$ offers comparable performance to PARSQL$_{llm}$ at lower API costs, making it suitable for low-cost settings.

Robustness Evaluation Table 3 evaluates PARSQL's robustness on three Spider variants: Spider-Syn, Spider-Realistic, and Spider-DK, highlighting the following: (1) PARSQL outperforms SFT CodeS of the same model size in average performance across its both variants. (2) PARSQL-3B$_{llm}$ surpasses RESDSQL-3B, T5-3B, and SFT CodeS-3B in average performance, establishing itself as the top performer at the 3B scale. This highlights the effectiveness of PARSQL$_{llm}$ in enhancing out-of-domain performance.

4.3 Ablation Studies

Effect of Key Components Table 4 reports the ablation studies of PARSQL-3B on BIRD Dev, highlighting the following: (1) Removing "reason pairs" significantly impacts performance, with drops of 0.92% for PARSQL-3B$_{rule}$ and 1.9% for PARSQL-3B$_{llm}$, highlighting the critical role of "reason pairs". (2) Even without using LLM-generated data ("new pairs"), PARSQL-3B$_{rule}$ achieves a performance of 55.41%, outperforming CodeS-3B (55.02%). This demonstrates that the performance gains do not stem from LLM-based augmentation, but rather from improvements brought by NL2Reason tasks. PARSQL, despite removing components such as LLM-augmented data, still surpasses the small-model SOTA, highlighting its effectiveness in reducing the semantic gap, im-

Methods	Spider-Syn		Spider-Realistic		Spider-DK	Average
	EX (%)	TS (%)	EX (%)	TS (%)	EX (%)	EX (%)
SQL-PaLM+PaLM 2 (Sun et al., 2024)	74.6	-	77.6	-	66.5	72.90
FastRAT$_{ext}$+GPT-4 (Shen et al., 2024)	74.4	-	80.9	-	72.3	75.86
TA-SQL+GPT-4 (Qu et al., 2024)	-	-	79.5	-	72.9	-
DART-SQL+GPT-3.5 (Mao et al., 2024)	-	-	79.3	-	71.4	-
ChatGPT (Li et al., 2023c)	58.6	48.5	63.4	49.2	62.6	61.53
RESDSQL-3B + NatSQL (Li et al., 2023a)	**76.9**	66.8	81.9	70.1	66.0	74.93
T5-3B + PICARD (Scholak et al., 2021)	69.8	61.8	71.4	61.7	62.5	67.90
SENSE-7B (Yang et al., 2024)	72.6	64.9	**82.7**	75.6	**77.9**	**77.73**
SFT CodeS-7B (Li et al., 2024b)	74.8	**67.4**	82.3	**76.8**	72.9	76.67
SFT CodeS-1B (Li et al., 2024b)	64.7	56.9	70.1	62.0	63.2	66
PARSQL-1B$_{rule}$	67.2 (+2.5)	59 (+2.1)	72.8 (+2.7)	65 (+3.0)	64.3 (+1.1)	68.1 (+2.1)
PARSQL-1B$_{llm}$	65.1 (+0.4)	57.3 (+0.4)	72.2 (+2.1)	62.8 (+0.8)	65.8 (+2.6)	67.7 (+1.7)
SFT CodeS-3B (Li et al., 2024b)	73.1	65.4	78.9	72.8	70.3	74.1
PARSQL-3B$_{rule}$	72.9 (-0.2)	64.8 (-0.6)	79.3 (+0.4)	74.2 (+1.4)	70.7 (+0.4)	74.3 (+0.2)
PARSQL-3B$_{llm}$	73.9 (+0.8)	66.6 (+1.2)	81.1 (+2.1)	72.8 (+0.0)	69.9 (-0.4)	74.97 (+0.87)

Table 3: Comparison on Spider variants.

	EX(%)	VES(%)
PARSQL-3B$_{rule}$	56.00	71.67
w/o new pairs	55.41(-0.59)	71.25(-0.42)
w/o rule-based reason pairs	55.08(-0.92)	70.20(-1.47)
w/o SQL selection	55.61(-0.39)	71.43(-0.24)
PARSQL-3B$_{llm}$	56.98	72.46
w/o new pairs	56.06(-0.92)	71.59(-0.87)
w/o LLM-based reason pairs	55.08(-1.9)	70.20(-2.26)

Table 4: Ablation study on the components of PARSQL.

Methods	BIRD (EX%)		Spider (EX%)	
	RVS	RVS + CSD	RVS	RVS + CSD
SFT CodeS-1B	53	38.57	76	66.8
PARSQL-1B$_{rule}$	54(+1.0)	43.95(+5.38)	78(+2.0)	71.7(+4.9)
SFT CodeS-3B	59	46.19	80	71.2
PARSQL-3B$_{rule}$	**59**	**52.02(+5.83)**	**81(+1.0)**	**73.9(+2.7)**

Table 5: Model performance in RVS and adding CSD.

proving interpretability, and enhancing reasoning transparency. These results underscore the importance of PARSQL in bridging the semantic gap between natural language questions and SQL, and in improving the reasoning capabilities necessary for accurate SQL generation.

Why "new pairs" is Helpful? To further explore the importance of "new pairs", we randomly sampled 100 samples from the Dev sets to create a Random Validation Subset (RVS). For each RVS sample, we generate sub-SQLs and corresponding questions, forming the Constraint-Sensitivity Datasets (CSD), with 123 samples for BIRD and 84 for Spider as shown in Appendix A.4.4.

Table 5 presents the model's performance on both test cases. While both models perform closely on RVS, PARSQL shows significant gains after adding CSD: a 5.38% increase on BIRD and 4.9% on Spider for PARSQL-1B$_{rule}$, and 5.83% (BIRD) and 2.7% (Spider) at the 3B scale, respectively. The improvement highlights PARSQL's sensitivity to question constraints, its ability to translate these constraints into SQL clauses, and the effectiveness of "new pairs" in handling ambiguous queries.

Why "reason pairs" is Helpful? To explore the role of "reason pairs", we calculate the vectors' cosine similarity between the questions and the generated SQLs across different models, utilizing the last layer of the hidden states of the corresponding counterparts to form the vectors. Figure 5 reports the similarity scores on five datasets. The results indicate that incorporating "reason pairs" significantly boosts similarity, with scores on BIRD Dev and Spider Dev increasing by over 10%. Combining these findings with results from Table 4, it is evident that training with "reason pairs" improves PARSQL's capabilities in SQL generation and semantic interpretation, aligning question intents more closely with SQL semantics.

Effect of Two Types of Generated "reasons" To investigate the effect of two "reason" generation methods, namely, rule-based and LLM-based, we compute the statistics of the "reason pairs": similarities between the reasons and the questions via embedding and N-gram methods, and the average length of the reasons. Figure 6 illustrates the correlation between these three indicators and

Methods	BIRD Dev		Spider Dev		Spider-Syn		Spider-Realistic		Spider-DK
	EX	VSE	EX	TS	EX	TX	EX	TS	EX
Baselines (Solutions in PARSQL-3B$_{rule}$)									
First-valid-execution	55.61	71.43	83.4	77.0	72.8	64.6	79.1	74.0	70.5
Execution-guided self-consistency	54.89	69.75	82.5	76.1	70.2	62.1	78.3	73.2	70.1
Oracle	63.56	76.14	87.5	80.9	80.2	72.3	83.9	78.3	78.3
Based on similarity between SQLs and reasons									
Hidden state embedding	50.2	63.14	78.5	71.5	68.2	58.6	75.2	66.5	65.2
all-mpnet-base-v2 (Reimers, 2021)	55.28	70.2	80.1	72.7	67.8	59.1	74.8	68.5	67.5
GTEv1.5-en-large (Zhang et al., 2024c)	55.54	70.43	81.8	74.5	69.0	59.7	75.8	71.1	68.8
Based on similarity between rule-based NL explanations of SQL and reasons									
all-mpnet-base-v2 (Reimers, 2021)	55.61	70.86	83.1	77.0	**72.9**	64.7	79.1	**74.4**	**70.7**
GTEv1.5-en-large (Zhang et al., 2024c)	55.8	**71.84**	**83.5**	**77.3**	72.4	64.4	78.9	74.2	70.1
N-grams	**56.0**	71.67	**83.5**	**77.3**	**72.9**	**64.8**	**79.3**	74.2	**70.7**

Table 6: Comparison of different SQL selection strategies on PARSQL-3B$_{rule}$.

	EX(%)	VES(%)
Granite-3B-Code	51.30	66.41
w/ PARSQL$_{llm}$	52.93(+1.63)	68.33(+1.92)
DeepSeek-Coder-1.3B	46.94	61.72
w/ PARSQL$_{llm}$	48.57(+1.63)	63.39(+1.67)
CodeS-1B	49.54	62.49
w/ PARSQL$_{llm}$	51.76(+2.22)	67.96(+5.47)
CodeS-3B	55.02	70.96
w/ PARSQL$_{llm}$	56.98(+1.96)	72.43(+1.47)

Table 7: Transferability of PARSQL on BIRD Dev.

EX(%)	1B	3B	7B
SFT CodeS	76.4	80.1	82.6
PARSQL$_{rule}$	**78.1(+1.7)**	**82.2(+1.1)**	**83.3(+0.7)**

Table 8: Impact of Model Scale on Spider Dev.

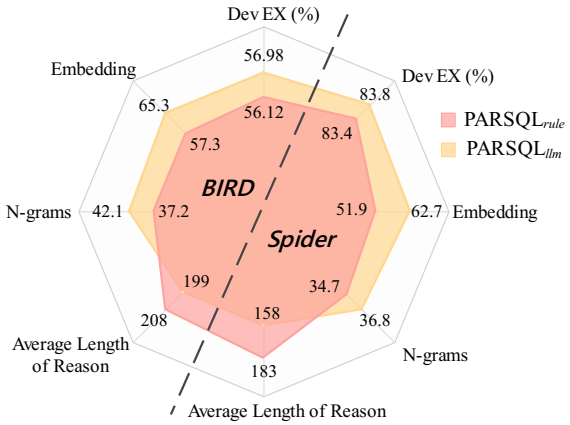


Figure 6: Performance of PARSQL$_{rule}$ and PARSQL$_{llm}$ w.r.t. "reasons" on the training sets of BIRD and Spider.

model performance, revealing that PARSQL performs better with semantically closer and shorter "reasons." Unlike rule-based methods that generate detailed explanations, LLM-based methods typically abstract SQL concepts into concise reasoning, aligning more effectively with the questions and reducing the semantic gaps.

Effect of Selection Strategy Table 6 compares different SQL selection strategies for PARSQL-3B$_{rule}$ across five datasets and highlights: (1) Rule-based SQL explanations, using either sentence embeddings or N-grams, generally outperform the baselines; (2) N-grams outperform embeddings on four datasets (except Spider-DK) due to better alignment with model-generated reasons, while embeddings may lose semantic details, especially when reasons are longer; (3) Direct SQL-reason similarity performs worse, emphasizing the semantic gap between SQLs and NL, which is mitigated through rule-based SQL explanations. (4) Oracle sets the upper bound for SQL selection. The rule-based method may misselect a query if the model generates the correct SQL but fails to produce the corresponding reason, lowering the similarity score. For more experimental details and results, please refer to Appendix A.4.5.

Transferability of PARSQL Table 7 reports the performance of widely-used baselines on BIRD Dev to explore the transferability of PARSQL. For each base model, the first row reports its performance when fine-tuned solely on the BIRD training set, while the second row (denoted as "w/ PARSQL$_{llm}$") reflects results after further fine-tuning on the LLM-generated augmented data from

Model	Easy	Medium	Hard	Extra	All
SFT CodeS-1B	91.10	83.60	68.40	51.80	77.80
PARSQL-1B$_{llm}$	91.90(+0.80)	83.60(+0.00)	**73.60(+5.20)**	53.00(+1.20)	80.20(+2.40)
PARSQL-1B$_{rule}$	**93.50(+2.40)**	**87.0(+3.40)**	70.10(+1.70)	**54.80(+3.00)**	**80.60(+2.80)**
SFT CodeS-3B	93.50	87.00	73.60	61.40	82.20
PARSQL-3B$_{llm}$	**94.80(+1.30)**	**88.60(+1.60)**	**79.90(+6.30)**	58.40(-3.00)	**83.80(+1.60)**
PARSQL-3B$_{rule}$	93.50(+0.00)	87.40(+0.40)	74.10(+0.50)	**67.50(+6.10)**	83.50(+1.30)

Table 9: EX across queries of varying levels of difficulty on Spider Dev (%).

PARSQL. Across all evaluated models, PARSQL consistently improves both execution accuracy (EX) and verification-based semantic accuracy (VES), demonstrating its strong generalizability. For instance, on Granite-3B-Code, PARSQL$_{llm}$ boosts EX from 51.30% to 52.93% (+1.63%) and VEX from 66.41% to 68.33% (+1.92%), indicating a substantial gain in semantic alignment. These results highlight that PARSQL not only offers a simple yet effective augmentation pipeline but also transfers well across different architectures and parameter scales, making it a broadly applicable solution for improving text-to-SQL performance.

Impact of Model Scale on Performance Table 8 shows the performance of PARSQL and SFT CodeS across different model sizes (1B, 3B, 7B) on the Spider dev set. All models are evaluated with a beam size of 1 to avoid out-of-memory issues, which may restrict PARSQL's SQL selection capabilities. Despite this limitation, PARSQL consistently outperforms SFT CodeS at all scales: +1.7% at 1B, +1.1% at 3B, and +0.7% at 7B. These improvements demonstrate that PARSQL is effective even with smaller models, and its advantage is most pronounced at the 1B scale. As model size increases, the performance gap narrows—likely because larger models can implicitly capture semantic alignment, reducing the relative impact of external reasoning supervision ("reason pairs"). Nonetheless, PARSQL maintains a consistent edge, suggesting that reasoning guidance remains beneficial even for stronger backbones.

Fine-grained Analysis on Hardness Spider's difficulty labels reveal PARSQL's superiority across all levels, as shown in Table 9. PARSQL-1B shows a significant performance improvement across all difficulty levels, with the highest gains as follows: Easy (2.4%), Medium (3.4%), Hard (5.20%), and Extra Hard (3.0%). PARSQL-3B also demonstrates notable improvements: Easy (1.3%), Medium (1.6%), Hard (6.3%), and Extra Hard (6.1%). This indicates that PARSQL has a stronger advantage in handling Hard and Extra Hard levels, benefiting from the enhanced constraint recognition and reasoning capabilities provided by the "new pairs" and "reason pairs".

5 Conclusion

We propose PARSQL to enhance the performance of SLMs for text-to-SQL tasks by leveraging SQL parsing and reasoning through novel data augmentation and efficient selection strategies. PARSQL utilizes PARSer to generate sub-SQLs and step-by-step explanations, offering more detailed insights into SQL syntax for data augmentation. Additionally, PARSQL incorporates multi-task learning to improve constraint differentiation and reasoning, alongside an efficient SQL selection strategy for post-correction. Extensive experimental results show that PARSQL outperforms SLMs in constraint sensitivity and semantic similarity between NL and SQLs, with PARSQL-3B rivaling 7B models despite having fewer parameters.

6 Limitations

We present several limitations as follows:

- In the "reason" generation process, our rule-based method currently only supports the SQLite dialect for SQL. While it can parse all SQL queries in the BIRD and Spider datasets, it may fail to parse SQL from other dialects.
- When adopting closed-source models for data generation, we rely solely on rule-based methods for data cleaning, which may affect the quality of the generated data. More data generation methods can be further explored.
- PARSer currently focuses exclusively on SQL processing. Expanding it to other tasks, such as code generation or mathematical computation, through the use of abstract syntax trees for data augmentation and rule-based reasoning steps, remains a promising area for future research.

References

Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. 2018. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *SIGMOD Conference*, pages 221–230. ACM.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2024. Navigate through enigmatic labyrinth A survey of chain of thought reasoning: Advances, frontiers and future. In *ACL (1)*, pages 1173–1203. Association for Computational Linguistics.

Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Yiying Yang, Haiqin Yang, Yuhang Zhao, Pingfu Chao, and Wenhao Jiang. 2025. Reex-sql: Reasoning with execution-aware reinforcement learning for text-to-sql. *arXiv preprint arXiv:2505.12768*.

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, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, pages 2166–2187. International Committee on Computational Linguistics.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 1337–1350. Association for Computational Linguistics.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your parser: Interactive text-to-sql with natural language feedback. In *ACL*, pages 2065–2077. Association for Computational Linguistics.

Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo A. Ramos, and Ahmed Hassan Awadallah. 2021. NL-EDIT: correcting semantic parse errors through natural language interaction. In *NAACL-HLT*, pages 5599–5610. Association for Computational Linguistics.

Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language models and large language models for zero-shot NL2SQL. *Proc. VLDB Endow.*, 17(11):2750–2763.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2505–2515. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8926–8931. 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.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019a. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4524–4535. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019b. Towards complex text-to-sql in cross-domain database with intermediate representation. In *ACL (1)*, pages 4524–4535. Association for Computational Linguistics.

Sabaat Haroon, Chris Brown, and Muhammad Ali Gulzar. 2024. Desql: Interactive debugging of SQL in data-intensive scalable computing. *Proc. ACM Softw. Eng.*, 1(FSE):767–788.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 8003–8017. Association for Computational Linguistics.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022a. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022b. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Minki Kang, Seanie Lee, Jinheon Baek, Kenji Kawaguchi, and Sung Ju Hwang. 2023. Knowledge-augmented reasoning distillation for small language models in knowledge-intensive tasks. In *NeurIPS*.

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

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to SQL: are we fully ready? [experiment, analysis \u0026 benchmark]. *Proc. VLDB Endow.*, 17(11):3318–3331.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 13067–13075. AAAI Press.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024b. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3):127.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 13076–13084. AAAI Press.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023c. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024a. A survey of NL2SQL with large language models: Where are we, and where are we going? *CoRR*, abs/2408.05109.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Yuyu Luo, and Nan Tang. 2024b. A survey of nl2sql with large language models: Where are we, and where are we going? *CoRR*, abs/2408.05109.

Ruilin Luo, Liyuan Wang, Binghuai Lin, Zicheng Lin, and Yujiu Yang. 2024. PTD-SQL: partitioning and targeted drilling with llms in text-to-sql. In *EMNLP*, pages 3767–3799. Association for Computational Linguistics.

Toby Mao. 2024. Python sql parser and transpiler.

Wenxin Mao, Ruiqi Wang, Jiyu Guo, Jichuan Zeng, Cuiyun Gao, Peiyi Han, and Chuanyi Liu. 2024. Enhancing text-to-sql parsing through question rewriting and execution-guided refinement. In *ACL (Findings)*, pages 2009–2024. Association for Computational Linguistics.

Chien Van Nguyen, Xuan Shen, Ryan Aponte, Yu Xia, Samyadeep Basu, Zhengmian Hu, Jian Chen, Mihir Parmar, Sasidhar Kunapuli, Joe Barrow, Junda Wu, Ashish Singh, Yu Wang, Jiuxiang Gu, Franck Dernoncourt, Nesreen K. Ahmed, Nedim Lipka, Ruiyi Zhang, Xiang Chen, Tong Yu, Sungchul Kim, Hanieh Deilamsalehy, Namyong Park, Mike Rimer, Zhehao Zhang, Huanrui Yang, Ryan A. Rossi, and Thien Huu Nguyen. 2024. A survey of small language models. *CoRR*, abs/2410.20011.

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. *arXiv preprint arXiv:2410.01943*.

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 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *ACL (Findings)*, pages 5456–5471. Association for Computational Linguistics.

Iryna Reimers. 2021. sentence-transformers/all-mpnet-base-v2. *Hugging Face*.

Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: making a large language model a better SQL writer. In *ICDE*, pages 15–28. IEEE.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 9895–9901. Association for Computational Linguistics.

Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, Kaustubh Vyas, Yuanyi Ji, and Jeff Z. Pan. 2024. Improving retrieval-augmented text-to-sql with ast-based ranking and schema pruning. In *EMNLP*, pages 7865–7879. Association for Computational Linguistics.

Ruoxi Sun, Sercan Ö. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2024. Sql-palm: Improved large language model adaptation for text-to-sql. *CoRR*, abs/2306.00739.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: contextual harnessing for efficient SQL synthesis. *CoRR*, abs/2405.16755.

Yuan Tian, Zheng Zhang, Zheng Ning, Toby Jia-Jun Li, Jonathan K. Kummerfeld, and Tianyi Zhang. 2023. Interactive text-to-sql generation via editable step-by-step explanations. In *EMNLP*, pages 16149–16166. Association for Computational Linguistics.

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, ACL 2020, Online, July 5-10, 2020*, pages 7567–7578. Association for Computational Linguistics.

Daniel C. Wang, Andrew W. Appel, Jeffrey L. Korn, and Christopher S. Serra. 1997. The zephyr abstract syntax description language. In *Proceedings of the Conference on Domain-Specific Languages, DSL'97, Santa Barbara, California, USA, October 15-17, 1997*, pages 213–228. USENIX.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. 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 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-sql data from weak and strong llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 7864–7875. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. 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, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.

Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. Chatglm: A family of large language models from GLM-130B to GLM-4 all tools. *CoRR*, abs/2406.12793.

Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024a. Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. In *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago AA, Chile, June 9-15, 2024*, pages 93–105. ACM.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023a. ACT-SQL: in-context learning for text-to-sql with automatically-generated chain-of-thought. In *EMNLP (Findings)*, pages 3501–3532. Association for Computational Linguistics.

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

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024c. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. In *EMNLP (Industry Track)*, pages 1393–1412. Association for Computational Linguistics.

Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023b. Sciencebenchmark: A complex real-world benchmark for evaluating natural language to SQL systems. *Proc. VLDB Endow.*, 17(4):685–698.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 396–411. Association for Computational Linguistics.

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

A Appendix

A.1 Details of PARSer

Algorithm 1: PARSer

Input: SQL
Output: $subSQLs, ReasoningPaths$
1 $subSQLs \leftarrow set(\,)$;
2 $ReasoningPaths \leftarrow [\,]$;
3 $Reasons \leftarrow [\,]$;
4 $ast \leftarrow$ ParserToAst (SQL) ;
5 $ConstraintIds \leftarrow$GetConstraints (ast);
6 $BinaryAst \leftarrow$ AstNode (ast) ;
7 **for** *id* **in** *ConstraintIds* **do**
8 TravrseAST($BinaryAst, id, subSQLs$)
9 **Function** TravrseAST(*node, id, subSQLs*)**:**
10 $subSQLs.add(node.ast.tosql(\,))$;
11 **if** not *node.left* **and** not *node.right* **then**
12 ast = DeleteConstraint($node.ast, id$) ;
13 $node.left \leftarrow$ AstNode (ast) ;
14 $node.right \leftarrow$ AstNode $(node.ast)$;
15 **return** 0
16 TravrseAST($node.left, id, subSQLs$) ;
17 TravrseAST($node.right, id, subSQLs$);
18 **return** 0
19 $Leaves \leftarrow$ GetAllLeaves($BinaryAst$) ;
20 $ReasoningPaths \leftarrow$ CombinePath($Leaves$) ;
21 **for** *path* **in** *ReasoningPaths* **do**
22 $Reasons.add($sqlexplanation$(path, ast))$;
23 **return** $subSQLs, ReasoningPaths, Reasons$

PARSer is a SQL disassembly tool we developed. It features the capabilities of generating sub-SQLs and translating SQL statements based on predefined rules. The detailed functionality of PARSer will be explained below.

A.1.1 Pipeline of PARSer

Algorithm 1 outlines the procedure of parser:

- Line 4 parses the SQL into an AST.
- Line 5 gets the set of constraints in the AST based on each node type. Detailed information about these rules is provided in the Appendix A.1.2.
- Line 6 initializes a binary tree to store results, with the AST placed in the root node.
- Lines 7-8 deletes each constraint and save the deleted AST in binary tree.
- Lines 9-18 defines a method to remove constraint in the AST from the leaf nodes of the binary tree, assigning the resulting new AST and a backup of the original AST as the child nodes.
- Lines 19 extract all possible sub-SQLs from the leaf nodes of the binary tree.
- Lines 20 identify the inclusion relationships of constraints in sub-SQLs and retrieve all existing reasoning paths.

Non-operational type nodes	Column, Table, Identifier, Literal, Null, Datatype, TableAlias
Operational type nodes	**Main body types:** SELECT, FROM, WHERE, EXISTS, IIF, CASE, CASE WHEN, JOIN, INNER JOIN, BETWEEN, LIKE, LIMIT, ORDER BY, GROUP BY, DESC,ASC, HAVING, SUBQUERY, WINDOW, OVER **Arithmetic operation types**: AND, OR, ADD (+), SUB (-), MUL (*), DIV (/), GT (>), GTE (>=), LT (<), LTE (<=), EQ (=), NEQ (!=), UNION, INTERSECT **Built-in function types**: AVG, COUNT, MAX, MIN, ROUND, SUM, ABS, NOW, CAST

Table 10: Operational and non-operational type nodes.

- Lines 21-22 traverse each inference path, traverse the AST according to the order of the inference path, and translate each node based on the rules.

A.1.2 Constraint Identification

Constraints are defined as subtrees within the AST, where the root node represents an operator, and all child nodes are non-operator types. The classification of each node type in AST is shown in the Table 10.

- **Non-operational type nodes** include columns, tables, identifiers, Literals, etc.
- **Operational types nodes** are defined as nodes whose operation objects are non-operational nodes. In simple terms, an operation node can be defined as a constraint.

Generally speaking, the subquery node in SQL represents a distinct SQL query. We treat it as a separate constraint and perform constraint decomposition on the subquery independently. A subquery (also known as an inner query or nested query) is a query that is embedded within another SQL query, Sub-query 1 is a sub-query node in sub-query 2.

For each sub-SQL, we can determine the constraints based on the following criteria:

- Dependencies exist between constraints, necessitating careful judgment before deletion. For example, the constraint "WHERE movies.movie_popularity > 1000" relies on the JOIN constraint "INNER JOIN movies ON ratings.movie_id = movies.movie_id." This indicates that the column referenced in the WHERE clause belongs to the movies table. Consequently, when deleting the JOIN constraint, the corresponding WHERE constraint must also

node type	rules
union	{} union {}
intersect	{} intersect{}
except	{} except {}
subquery	the result of the {}
select	subquery_id
table	table name
From	From the table {}
Join	join the table {join_name} on the condition that {column1} equals {column2}
Add	{} add {}
sub	{} subtract {}
Mul	{} multiply {}
Div	{} divide {}
Cast	convert {object1} to {type_object} type
Round	round {object1}
Avg	the average of {this_node}
Sum	the sum of {this_node}
Count	the count of {this_node}
Max	the minimum of {this_node}
Min	the maximum of {this_node}
Length	the length of {this_node}
Between	{table_name} is between {lower_bound} and {upper_bound}
Case	if {condition}, the result will be {true_value}
Or	{condition1} or {condition2}
And	{condition1} and {condition2}
GT	{table1_name} is greater than {table2_name}
LT	{table1_name} is less than {table2_name}
GTE	{table1_name} is greater than or equal to {table2_name}
LTE	{table1_name} is less than or equal to {table2_name}
EQ	{table1_name} equals {table2_name}
NEQ	{table1_name} does not equal {table2_name}
Column	{node.name} of {node.table}
Literal	{value}
Limit	keep only the first {num} rows
Offset	skip the first {num} rows
Distinct	remove duplicate rows
Star	all records

Table 11: Explanation rules for basic SQL elements.

node type	rules
In	{column_node} is in {}
not in	{column_node} is not in {}
Like	{column_node} is in the form of {format_node}
not like	{column_node} is not in the form of {format_node}
Is	{column_node} is {format_node}
not is	{column_node} is not {format_node}
Trim	trim 'string' from {column}
GroupConcat	concatenate the values of {column} in each group
CurrentTimestamp	current time
TimestampDiff	the difference between {time1} and {time2} in {unit}
DateDiff	the difference between {time1} and {time2}
SUBSTR	the substring of {column} starting from {start} with length {length}
STRFTIME	the time {time} formatted as {format}
INSTR	the position of {string} in {column} starting from {start} in the {occurrence} time
REPLACE	{column} with 'old' replaced by 'new'
TOTAL	sum up {columns}
Window	window function on {table_name}, then group rows by {columns}, and sort rows by {columns}
RowNumber	add line number
Where	filter where {conditions}
Group	then group rows by {columns}
Having	and keep the groups where
Order	and sort rows by
Select	finally return the values of {columns}

Table 12: Explanation rules for advanced SQL elements.

be removed; otherwise, the JOIN constraint cannot be deleted. Nodes with dependencies include: GROUP BY and HAVING; and JOIN nodes along with all constraints involving the tables in the JOIN.

- Constraints can be merged to reduce the number of generated sub-SQLs. For instance, the constraints "SELECT name" and "SELECT year" can be combined because both columns belong to the person table. However, if the columns originate from different tables, they cannot be merged. The only node types that can be merged are non-operation column node types.

A.1.3 Rule-based SQL Explanations

Table 11 and Table 12 correspond to the explanation rules for each node type in the AST, where `{}` represents the explanation of the current node's child nodes. We follow the order of constraint nodes in the reasoning path to translate each constraint. Additionally, we use an inorder traversal when processing the AST. For example, for the constraint `MAX(year) + 1`, we first encounter the year node, which is of type `Column` and corresponds to the explanation template `{node.name} of {name.table}`. Next, we traverse the `MAX` node, where the explanation template is `the maximum of {this_node}`, and the explanation content for the year node is inserted as `this_node`, resulting in "`the maximum of year`". Finally, we traverse the `ADD` node, which has the explanation rule `{} add {}`. The explanation content for the `MAX` and `LITERAL` nodes is inserted into the corresponding `{}` placeholders, with the `LITERAL` node having a value of 1. The final explanation for the constraint `MAX(year) + 1` is "`the maximum of year add 1`".

In addition, when encountering multiple nested queries, we fuse the corresponding contents of all nested queries together as shown in the Table 13.

Chaining rules for multiple nested queries
Start with subquery_1: {Explanation content of subquery_1} then, generate subquery_2: {Explanation content of subquery_2} then, generate subquery_3: {Explanation content of subquery_3} ... ... then, generate subquery_N: {Explanation content of subquery_N}

Table 13: Explanation rules when there are N nested queries in a SQL. The explanation content of each subquery in {}.

A.1.4 Results of Parsing on BIRD and Spider

We apply PARSer to BIRD and Spider datasets. For the training set, we limit the number of sub-SQLs to 256, and do not perform data enhancement on samples that are too complex. We count the corresponding results in each dataset, as shown in the Table 14. It can be found that each sample in the BIRD dataset can be decomposed into 14.4 sub-SQLs, the average number of reasoning paths is 199.7, and the number of sub-SQLs involved in each reasoning path is 4.8. For the Spider dataset, the structure of SQL is simpler, so the sub-SQLs and reasoning paths generated by the algorithm are less than those of the BIRD dataset.

A.1.5 Comparison of PARSer and Existing SQL Parsers

PARSer is a self-developed method based on SQLglot. Compared with the existing SQL decomposition in Table 15, it can divide SQL clauses more finely, split all possible executable sub-SQLs, and obtain multiple reasoning paths. We compared it with two recently published works, such as STEPS (Tian et al., 2023) and DeSQL (Haroon et al., 2024).

- Neither STEPS nor DeSQL can support outputting multiple reasoning paths.
- STEPS cannot fine-grain SQL, resulting in clauses not being independent units.
- DeSQL has limited SQL structures involved, cannot handle nested queries, etc., and can only decompose simple SQL.

We show an example of applying PARSer in Table 7. The input is SQL query and the output is sub-SQLs, reasoning paths and rule-based SQL explanation. At the same time, we also display the results of STEPS and DeSQL in Table 16.

Input SQL
SELECT admfname1 FROM schools GROUP BY admfname1 ORDER BY COUNT (admfname1) DESC LIMIT 2

Output
sub-SQLs
1: SELECT * FROM schools,
2: SELECT * FROM schools GROUP BY admfname1,
3: SELECT * FROM schools LIMIT 2,
4: SELECT admfname1 FROM schools,
5: SELECT * FROM schools GROUP BY admfname1 ORDER BY COUNT (admfname1) DESC,
6: SELECT * FROM schools GROUP BY admfname1 LIMIT 2,
7: SELECT admfname1 FROM schools GROUP BY admfname1,
8: SELECT admfname1 FROM schools LIMIT 2,
9: SELECT * FROM schools GROUP BY admfname1 ORDER BY COUNT (admfname1) DESC LIMIT 2,
10: SELECT admfname1 FROM schools GROUP BY admfname1 ORDER BY COUNT (admfname1) DESC,
11: SELECT admfname1 FROM schools GROUP BY admfname1 LIMIT 2,
12: SELECT admfname1 FROM schools GROUP BY admfname1 ORDER BY COUNT (admfname1) DESC LIMIT 2,

---

Reasoning Paths
p1: [1, 2, 5, 9, 12], p2: [1, 2, 5, 10, 12], p3: [1, 2, 6, 9, 12],
p4: [1, 2, 6, 11, 12], p5: [1, 2, 7, 10, 12],

SQL explanation based on p1
From the table schools, then group rows by admfname1, and sort rows by the count of admfname1 in descending order, finally return the values of admfname1, and keep only the first 2 rows.

Figure 7: An example of input and output in PARSer.

A.2 Prompt Templates

In our work, there are totally four kinds of prompts:

- **Sub-question generation prompt**: This prompt is designed to generate augmented (sub-question, sub-SQL) pairs based on the provided sub-SQLs; refer to the prompt in Figure 9 and an example of a (sub-question, sub-SQL) pair in Figure 10.
- **Reason generation prompt**: This prompt generates the reason or description of a chain of thought (CoT) based on the given reasoning path; see the prompt in Figure 11 and an example of a (question, reason) pair in Figure 12.

Dataset	Number of Sub-SQLs			Number of Reasoning Paths			Length of Reasoning Path		
	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
BIRD Train Set	14.4	256	1	199.7	130,704	0	4.8	11	2
BIRD Dev Set	13.7	192	2	97.6	20,160	1	4.8	11	2
Spider Train Set	8.2	240	2	42.4	99,360	1	3.9	11	2
Spider dev Set	8.0	50	2	12.3	560	1	3.9	8	2

Table 14: Results Statistics after PARSer Processing of the BIRD and Spider Datasets.

	Split Clause	Synthetic sub-SQL	multiple paths	Complex syntax	SQL explanation
PARSer	✓(Refinement)	✓	✓	✓	✓(Refinement)
STEPS	✓	×	×	x	✓
DeSQL	✓(Refinement)	✓	×	×	×

Table 15: Comparison of PARSer and existing SQL decomposers.

SQL query	SELECT MAX(CAST(T1.“Free Meal Count (Ages 5-17)” AS REAL) / T1.“Enrollment (Ages 5-17)”) FROM frpm AS T1 INNER JOIN satscores AS T2 ON T1.CDSCode = T2.cds WHERE CAST(T2.NumGE1500 AS REAL) / T2.NumTstTakr > 0.3
PARSer	From the table frpm, join the table satscores, filter where convert satscores.numge1500 to FLOAT type, then divide satscores.numtsttakr, is greater than 0.3, finally return the values of the maximum of convert frpm.free meal count (ages 5-17) to FLOAT type, then divide frpm.enrollment (ages 5-17).
STEPS	In table frpm, Keep the records where the **NumGE1500 AS REAL** of satscores, **the CAST, the /,** and the NumTstTakr of satscores is greater than the 0.3, Return the maximum **value of CAST (“Free Meal Count” of T1 (Ages 5-17, the “Free Meal Count” of T1 (Ages 5-17, the Ages 5-17, the Ages 5-17, and the NumGE1500 of T2 AS REAL**
DeSQL	No SQL explanation function.

Table 16: Comparison of SQL explanation. **Bold fonts** represent incorrect content.

A.3 Details of Data Augmentation

As shown in Table 14, PARSer parses an average of over 10 sub-SQLs and 90 reasoning paths in the BIRD dataset, and over 8 sub-SQLs and 10 reasoning paths in the Spider dataset. Given the large volume, we adopt a strategy to optimize selection and reduce costs. All data enhancements rely on the training set. For sub-SQLs, we select those with constraints differing by two or fewer from the original SQL in the BIRD dataset. In contrast, Spider sub-SQLs may correspond to multiple original SQLs due to its approach of limiting condition deletion. To increase data, we collect sub-SQLs for Spider with constraints differing by three or fewer.

We exclude sub-SQLs with extraneous constraints, such as unnecessary JOIN operations, as removing or retaining these does not affect execution. The generated data is manually reviewed, resulting in 2,110 sub-question/sub-SQL pairs for BIRD and 1,108 for Spider. For reasoning paths, we randomly select one path from the original question and manually filter out low-quality data, yielding 9,108 sub-question/sub-SQL pairs for BIRD and 8,505 for Spider.

A.4 More Details about Experiments

A.4.1 Details of Datasets

In our work, we conduct experiments on the following datasets:

- **BIRD** (Li et al., 2023c) is the first cross-domain, large-scale benchmark specifically designed to bridge the gap between academic research and real-world applications in text-to-SQL parsing. It features a substantial dataset comprising 12,751 text-to-SQL pairs, 95 databases across 37 professional domains, and a total size of 33.4 GB. In comparison to Spider (Yu et al., 2018) and WikiSQL (Zhong et al., 2017), BIRD-SQL emphasizes database content and aligns more closely with real-world scenarios. However, due to hardware limitations of the BIRD submission platform, we are unable to evaluate our model on its test set.
- **Spider** (Yu et al., 2018) is a large-scale semantic parsing and text-to-SQL dataset annotated by 11 students from Yale University. It includes a training set with 8,659 samples, a development set with 1,034 samples, and a test set with 2,147 samples, covering 200 distinct databases across 138 domains. Of these, 160 databases are allo-

λ	0.6	0.8	1.0
EX(%)	50.72	51.76	51.37

Table 17: Effect of train parameter λ.

cated for training and development, while 40 are designated for testing.

- **Spider-DK** (Gan et al., 2021b), **Spider-Syn** (Gan et al., 2021a), and **Spider-Realistic** (Deng et al., 2021) are variants of the original Spider dataset, designed to resemble queries that users might ask in real-world scenarios. These variants allow for a more comprehensive evaluation of the model's robustness in the text-to-SQL task.

A.4.2 Effect of λ

Table 17 test the effect of the hyperparameter λ. We vary λ in {0.6, 0.8, 1.0} and test the performance of PARSQL-1B$_{llm}$ on BIRD Dev. The results indicate that when λ is set to 0.8, the weighting between the question/SQL pairs and the question/Reason pairs is optimized. If the weight of the Reason pairs is too low, the model fails to capture reasoning information effectively. Conversely, if λ is set too high, the model tends to overemphasize the reasoning pairs, neglecting the information from the question/SQL pairs. Both scenarios can diminish the model's effectiveness in generating SQLs.

A.4.3 Fine-grained Analysis of Training Data Query-Type Effectiveness

Following the query categorization in SuperSQL (Li et al., 2024a), we classify queries into five types: nested-query, logical-connector, join, order-by, and group-by. Table 19 presents their distribution in the BIRD training set. To evaluate the impact of augmentation, we compare performance on the BIRD development set, as shown in Table 18. The average improvement reflects the mean accuracy gain of PARSQL-3B$_{rule}$ and PARSQL-3B$_{llm}$ over SFT CodeS-3B across query types. Key observations include:

- PARSQL improves performance across all query types.
- The improvement for **nested queries** is the smallest (0.44%), despite a 12.57% increase in corresponding training data.
- **Order-by** and **group-by** queries exhibit the most significant gains.

These results suggest that handling nested queries remains a key challenge, as both its performance improvement and EX metric are the lowest among all query types, indicating potential areas for future enhancement.

A.4.4 More Experiments on Testing Constraint Sensitivity

Random Validation Subset (RVS) To create the Random RVS, we randomly selected 100 samples from the development set, ensuring a diverse mix of problem types and complexities. The selection process was unbiased, reflecting the general distribution of the development set.

Constraint-Sensitivity Dataset (CSD) The goal of the CSD is to assess the model's sensitivity to variations in constraints. For each RVS sample, we generated subSQL queries by removing a single constraint at a time. These constraints, such as conditions in the WHERE clause or JOIN conditions, were modified individually to isolate the effect of each change. The CSD generation follows a process similar to generating "new pairs":

- **SubSQL generation.** For each RVS sample, PARSer creates multiple subSQLs by removing one constraint at a time, ensuring minimal changes to the query structure.
- **Question generation.** The corresponding question description is adjusted to match the modified query, with GLM-4-0520 (Zeng et al., 2024) used to generate the question based on the modified subSQL.

The resulting CSD contains 123 samples for BIRD and 84 for Spider. Since Spider queries are simpler, fewer subSQLs could be generated, resulting in a smaller number. These datasets allow for a focused evaluation of the model's ability to handle variations in constraints, offering insights into how well it adapts to such changes.

A.4.5 More Experiments on Testing SQL Selection Strategies

We present the performance of PARSQL$_{rule}$ only, as the results of PARSQL$_{llm}$ are not included. This is because the reasons generated by PARSQL$_{llm}$ differ significantly in form and structure from the rule-based SQL explanations (as shown in Figure 7 and Table 12). Such differences can negatively impact SQL selection based on similarity, thereby affecting the model's performance.

We employed two primary methods to calculate the similarity between SQL and "reason". The first directly computes similarity using SQL, while the second translates SQL into its rule-based representation before calculating similarity with "rea-

	Nested-query	Logical-connector	join	Order-by	Group-by
SFT CodeS-3B	31.86	47.41	52.98	44.66	31.86
PARSQL-3B$_{rule}$	31.86	47.72	55.09	**48.87**	35.4
PARSQL-3B$_{llm}$	**32.74**	**49.45**	**55.26**	46.93	**38.05**
Average improvement	+0.44	+1.17	+2.19	+3.24	+4.86

Table 18: EX score (%) on BIRD dev set for different SQL query types.

	Nested-query	Logical-connector	join	Order-by	Group-by
Origin training dataset	723	5063	7212	1722	1009
After adding new pairs	723(+12.57%)	5063(+17.14%)	7212(+17.18%)	1722(+16.48%)	1009(+19.41%)

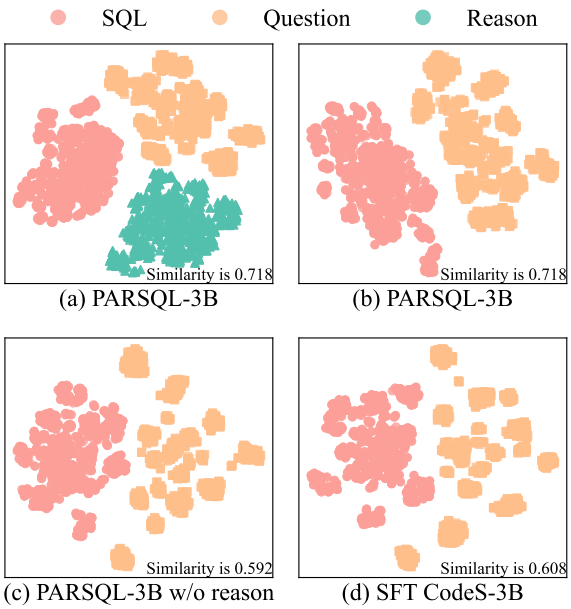
Table 19: Distribution of SQL Query Types in Training Data: Original vs. Augmented Dataset Comparison

son". Each similarity computation utilizes four approaches: an N-gram-based approach and three cosine similarity-based approaches. The latter include PARSQL's hidden state, sentence embeddings from all-mpnet-base-v2 (Reimers, 2021), and embeddings from GTEv1.5-en-large (Zhang et al., 2024c).

Table 20 presents the comprehensive experimental results of PARSQL$_{rule}$ for both the 1B and 3B models. highlighting the following: (1) The result of PARSQL-1B$_{rule}$ reaffirms that rule-based SQL explanations, using either sentence embeddings or N-grams, generally outperform the baseline; (2) Direct SQL-reason similarity performs worse, emphasizing the semantic gap between SQL and natural language, which is mitigated through rule-based SQL explanations. (3) In addition, there is not much difference between using the model to calculate similarity and N-grams. In low resource scenarios, N-grams has more advantages. (4) Oracle represents the upper bound for SQL selection with a beam size of 4. As shown, the rule-based method does not always select the correct SQL query. There is still significant room for improvement in SQL selection strategies based on reasoning. Future work will focus on supporting a wider range of SQL syntaxes and developing advanced SQL selection strategies based on reasoning, aiming to improve the model's adaptability and accuracy.

A.4.6 Visualization of Questions and SQLs

Figure 8 presents a 2-D t-SNE visualization of the vector representations between the question and the generated SQL for both PARSQL and SFT CodeS, where the vectors represent the final hidden layer outputs of the model. Several key observations can be made: (1) After incorporating reason pairs,



(a) PARSQL-3B (b) PARSQL-3B (c) PARSQL-3B w/o reason (d) SFT CodeS-3B

Figure 8: 2-D t-SNE visualization Comparing SQL and Question Embeddings in Spider Dev: PARSQL and SFT CodeS. Embeddings of between question and SQL using last-layer hidden representations.

the similarity between the vectors of the question and SQL in PARSQL shows a significant improvement. (2) The distribution of question vectors in PARSQL is more compact, whereas in SFT CodeS, the distribution is more dispersed. Additionally, in PARSQL, there is a clear boundary between the SQL and question vectors, while in SFT CodeS, the distribution is more chaotic. This suggests that PARSQL is better at distinguishing and aligning the vector representations of questions and SQL, allowing the model to more accurately capture the semantic structure of the question when generating SQL. In contrast, the lack of a clear distinction between the question and SQL vectors in SFT CodeS indicates a degree of ambiguity in understanding

Methods	BIRD Dev		Spider Dev		Spider-Syn		Spider-Realistic		Spider-DK
	EX	VSE	EX	TS	EX	TX	EX	TS	EX
Baselines (Solutions in PARSQL-3B$_{rule}$)									
First-valid-execution	55.61	71.43	83.4	77.0	72.8	64.6	79.1	74.0	70.5
Execution-guided self-consistency	54.89	69.75	82.5	76.1	70.2	62.1	78.3	73.2	70.1
Oracle	63.56	76.14	87.5	80.9	80.2	72.3	83.9	78.3	78.3
Based on similarity between SQLs and reasons (Solutions in PARSQL-3B$_{rule}$)									
Hidden state embedding	50.2	63.14	78.5	71.5	68.2	58.6	75.2	66.5	65.2
all-mpnet-base-v2 (Reimers, 2021)	55.28	70.2	80.1	72.7	67.8	59.1	74.8	68.5	67.5
GTEv1.5-en-large (Zhang et al., 2024c)	55.54	70.43	81.8	74.5	69.0	59.7	75.8	71.1	68.8
N-grams	54.11	68.45	80.7	73.3	69.9	61.1	77.4	71.3	68.6
Based on similarity between rule-based NL explanations of SQL and reasons (Solutions in PARSQL-3B$_{rule}$)									
Hidden state embedding	55.41	69.45	83.3	76.9	72.3	64.1	79.3	74.2	70.3
all-mpnet-base-v2 (Reimers, 2021)	55.61	70.86	83.1	77.0	**72.9**	64.7	79.1	**74.4**	**70.7**
GTEv1.5-en-large (Zhang et al., 2024c)	55.8	**71.84**	**83.5**	**77.3**	72.4	64.4	78.9	74.2	70.1
N-grams	**56.0**	71.67	**83.5**	**77.3**	**72.9**	**64.8**	**79.3**	74.2	**70.7**
Baselines (Solutions in PARSQL-1B$_{rule}$)									
First Valid Execution	51.43	67.54	80.1	73.5	66.9	58.3	72.4	65.2	63.7
Execution-guided self-consistency	50.39	64.51	78.3	71.7	64.2	55.6	70.7	63.4	63.2
Oracle	58.74	72.62	84.2	77.0	73.4	63.8	77.4	69.7	74.2
Based on similarity between SQLs and reasons (Solutions in PARSQL-1B$_{rule}$)									
Hidden state embedding	47.07	61.64	75.5	68.1	61.6	51.8	67.3	58.1	61.7
all-mpnet-base-v2 (Reimers, 2021)	50.2	64.39	76.5	69.2	64.1	55.5	69.3	61.2	62.6
GTEv1.5-en-large (Zhang et al., 2024c)	49.22	63.12	76.7	69.6	63.6	54.7	69.9	62.2	62.6
N-grams	50.26	64.85	77.4	70.7	64.8	55.5	69.5	60.2	60.0
Based on similarity between rule-based NL explanations of SQL and reasons (Solutions in PARSQL-1B$_{rule}$)									
Hidden state embedding	51.43	67.23	79.9	73.0	66.9	58.2	72.4	65.2	64.1
all-mpnet-base-v2 (Reimers, 2021)	51.56	67.01	**80.8**	**74.0**	67.0	58.6	**72.8**	65.6	**64.9**
GTEv1.5-en-large (Zhang et al., 2024c)	**51.69**	**67.79**	80.1	73.5	**67.2**	**59.0**	72.6	**65.7**	64.7
N-grams	**51.69**	67.71	80.6	73.8	**67.2**	**59.0**	72.8	65.0	64.3

Table 20: Comparison of different SQL selection strategies on PARSQL-3B$_{rule}$ and PARSQL-1B$_{rule}$.

the relationship between the question and the generated SQL, which may lead to a larger semantic gap between the question and the resulting SQL.

A.4.7 Case Study of PARSQL vs. SFT CodeS

At the end of our work, we present examples where PARSQL-3B$_{llm}$ produces correct outputs while SFT CodeS-3B generates incorrect results under two types of errors (generating SQL with incorrect logic and incomplete conditions), as shown in Tables 21 to Table 24. These examples highlight the superior capability of PARSQL-3B$_{llm}$ in accurately capturing logical structures and handling complex conditions in SQL generation. In contrast, SFT CodeS-3B demonstrates limitations in ensuring logical correctness and completeness, leading to suboptimal outputs. This comparison underscores the robustness and reliability of PARSQL-3B$_{llm}$, making it more suitable for tasks requiring precise and comprehensive SQL generation.

Question	Please list the CodeS of the schools with a total enrollment of over 500.
Evidence	Total enrollment can be represented by ‘enrollment (k-12)‘ + ‘enrollment (ages 5-17)‘;
READ-SQL result	SELECT DISTINCT cdscode FROM frpm WHERE cast(‘enrollment (k-12)‘ + ‘enrollment (ages 5-17)‘ AS REAL) > 500
SFT CodeS result	SELECT ‘school code‘ FROM frpm WHERE ‘enrollment (k-12)‘ + ‘enrollment (ages 5-17)‘ > 500
Cause analysis	Confusing school code and cdscode

Table 21: Generating SQL with incorrect logic: case 1.

Instructions
You are an expert at translating SQL queries into natural language questions. Your task is to generate a clear, concise, and detailed question that accurately captures the intent of the SQL query.

- -

Example
SQL Query: SELECT avg(ratings.rating_score) FROM movies INNER JOIN ratings ON movies.movie_id = ratings.movie_id WHERE movies.movie_title = 'When Will I Be Loved'
Generated Question: What is the average rating for movie titled 'When Will I Be Loved'?
SQL Query: SELECT products.name FROM products INNER JOIN sales ON products.productid = sales.productid WHERE sales.salespersonid = 20 ORDER BY sales.quantity DESC LIMIT 1
Generated Question: What is the name of the product that is most sold by sale person id 20?
SQL Query: {Origin SQL}
Generated Question: {Origin question}

- -

Now, create a detailed yet concise question that is semantically consistent with the following SQL query. Ensure that the generated question closely follows the structure of the example, while accounting for any differences in the SQL.
SQL Query: {Sub-SQL}
Generated Question:

Figure 9: Sub-question generation prompt for the augmented (sub-question, sub-SQL) pairs.

Original (Question,SQL) pair
Question: What are the URL to the list page on Mubi of the lists with followers between 1-2 and whose last update timestamp was on 2012?
SQL: SELECT list_url FROM LISTS WHERE list_update_timestamp_utc LIKE '2012%' AND list_followers BETWEEN 1 AND 2 ORDER BY list_update_timestamp_utc DESC LIMIT 1

- -

(sub-question, sub-SQL) pairs
Pair #1:
sub-question: What are the URLs of the lists on Mubi with a last update timestamp in 2012 and a follower count between 1 and 2, sorted by the update timestamp in descending order?
sub-SQL: SELECT list_url FROM LISTS WHERE list_update_timestamp_utc LIKE '2012%' AND list_followers BETWEEN 1 AND 2 ORDER BY list_update_timestamp_utc DES
Pair #2:
sub-question: What is the URL of the list page on Mubi with the fewest followers, where the last update timestamp is the most recent?
sub-SQL: SELECT list_url FROM LISTS WHERE list_followers BETWEEN 1 AND 2 ORDER BY list_update_timestamp_utc DESC LIMIT 1

Figure 10: An example of a (sub-question, sub-SQL) pair.

Instructions
Generate a chain-of-thought (CoT) reasoning process that explains how each sub-SQL query incrementally builds towards answering the original question. Ensure the reasoning is clear, concise, and logically follows the progression of the sub-SQLs.

- -

Examples
Question: What is the average writing score of each of the schools managed by Ricci Ulrich? List the schools and the corresponding average writing scores.
reasoning path: /**
1.SELECT * FROM satscores
2.INNER JOIN schools ON satscores.cds = schools.cdscode
3.WHWERE schools.admlname1 = 'Ulrich'
4.WHERE schools.admfname1 = 'Ricci'
5.SELECT satscores.avgscrwrite **/
Now, the requirements are as follows: the output must be short and consist of a sentence, SQL clause generates half of a sentence, split by ',', only the reasoning process needs to be output and output in English:
Generate reasoning path:
First, select information from satscores, then join the schools table, add school information, and add a filter based on Ricci Ulrich; finally, choose to display the average writing score.

- -

I will provide a new question and sub-SQLs list. Following the above example, generate the corresponding reasoning path step-by-step.
Question: {Question}
reasoning path: /**
{reasoning path} **/
Now, The requirements are as follows: the output must be short and consist of a sentence, each SQL clause generates half of a sentence, split by ',', only the reasoning process needs to be output and output in English:
Generate reasoning path:

Figure 11: Reason generation prompt.

Reasoning path
Sub-SQL 1 : SELECT * FROM LISTS,
Sub-SQL 2 : SELECT * FROM LISTS ORDER BY list_update_timestamp_utc DESC,
Sub-SQL 3 : SELECT * FROM LISTS WHERE list_followers BETWEEN 1 AND 2 ORDER BY list_update_timestamp_utc DESC,
Sub-SQL 4 : SELECT * FROM LISTS WHERE list_update_timestamp_utc LIKE '2012%' AND list_followers BETWEEN 1 AND 2 ORDER BY list_update_timestamp_utc DESC,
Sub-SQL 5 : SELECT * FROM LISTS WHERE list_update_timestamp_utc LIKE '2012%' AND list_followers BETWEEN 1 AND 2 ORDER BY list_update_timestamp_utc DESC LIMIT 1,
Sub-SQL 6 : SELECT list_url FROM LISTS WHERE list_update_timestamp_utc LIKE '2012%' AND list_followers BETWEEN 1 AND 2 ORDER BY list_update_timestamp_utc DESC LIMIT 1

(question, reason) pair
question: What are the URL to the list page on Mubi of the lists with followers between 1-2 and whose last update timestamp was on 2012?
reason: Select all lists, then sort by update timestamp in descending order; filter for lists with followers between 1-2 and an update timestamp in 2012; Keep the order and limit it to the first item; retrieve the URL for the specified list.

Figure 12: An example of a (question, reason) pair, where the reason is generated from the given reasoning path for the question.

Question	Sort in descending order all patients by birthday for male patient with albumin not within range.
Evidence	albumin not within range refers to alb <= 3.5 or alb >= 5.5; male = sex = 'M';
READ-SQL result	SELECT patient.id FROM patient INNER JOIN laboratory ON patient.id = laboratory.id WHERE patient.sex = 'M' AND (laboratory.alb <= 3.5 OR laboratory.alb >= 5.5) ORDER BY patient.birthday DESC
SFT CodeS result	SELECT patient.id FROM patient INNER JOIN laboratory ON patient.id = laboratory.id WHERE patient.sex = 'M' AND (laboratory.alb < 3.5 OR laboratory.alb > 5.5) ORDER BY patient.birthday DESC
Cause analysis	Symbols mixed up

Table 22: Generating SQL with incorrect logic: case 2.

Question	List the patient ID, sex and birthday who has abnormal white blood cell count. Group them by sex and list the patient by age in ascending order.
Evidence	abnormal white blood cell count refers to wbc <= 3.5 or wbc >= 9.0;
READ-SQL result	SELECT patient.id, patient.sex, patient.birthday FROM patient INNER JOIN laboratory ON patient.id = laboratory.id WHERE laboratory.wbc <= 3.5 OR laboratory.wbc >= 9.0 GROUP BY patient.id, patient.sex, patient.birthday ORDER BY patient.birthday ASC
SFT CodeS result	SELECT patient.id, patient.sex, patient.birthday FROM patient INNER JOIN laboratory ON patient.id = laboratory.id WHERE laboratory.wbc BETWEEN 3.5 AND 9.0
Cause analysis	Lost the ORDER by condition.

Table 23: Generating SQL with incomplete conditions: case 1.

Question	Between San Diego and Santa Barbara, which county offers the most number of schools that does not offer physical building? Indicate the amount.
Evidence	Does not offer physical building' means virtual = F in the database.
READ-SQL result	SELECT county, count(school) FROM schools WHERE county IN ('San Diego', 'Santa Barbara') AND virtual = 'F' GROUP BY county ORDER BY count(school) DESC LIMIT 1
SFT CodeS result	SELECT county, count(cdscode) FROM schools WHERE county IN ('San Diego', 'Santa Barbara') AND virtual = 'F' GROUP BY county
Cause analysis	There is no guarantee of the required number of rows.

Table 24: Generating SQL with incomplete conditions: case 2.