Feather-SQL: A Lightweight NL2SQL Framework with Dual-Model Collaboration Paradigm for Small Language Models

Wenqi Pei[1], **Hailing Xu**[1], **Henry Hengyuan Zhao**[1], **Shizheng Hou**[1], **Han Chen**[1], **Zining Zhang**[1], **Pingyi Luo**[2], **Bingsheng He**[1,*]

[1]National University of Singapore [2]4Paradigm

Abstract

Natural Language to SQL (NL2SQL) has seen significant advancements with large language models (LLMs). However, these models often depend on closed-source methods and high computational resources, posing challenges in data privacy and deployment. In contrast, small language models (SLMs) struggle with NL2SQL tasks, exhibiting poor performance and incompatibility with existing frameworks. To address these issues, we introduce **Feather-SQL**, a new lightweight framework tailored for SLMs. Feather-SQL improves SQL executability and accuracy through: (i) schema pruning and linking, (ii) multi-path and multi-candidate generation. Additionally, we introduce **1+1 Model Collaboration Paradigm**, which pairs a strong general-purpose chat model with a fine-tuned SQL model, combining strong analytical reasoning with high-precision SQL generation. Experimental results on BIRD demonstrate that Feather-SQL improves NL2SQL performance on SLMs, with around 10% boost for models without fine-tuning. The proposed paradigm raises the accuracy ceiling of SLMs to 54.76%, highlighting its effectiveness. Our implementation is available at `https://github.com/CedricPei/Feather-SQL`.

1 Introduction

Natural Language to SQL (NL2SQL) is the task of converting natural language questions into corresponding SQL queries, allowing users to retrieve structured data from databases without requiring proficiency in SQL. In recent years, the field has seen significant advancements with the emergence of large language models (LLMs) such as GPT-4 (OpenAI et al., 2024), enabling frameworks like CHASE-SQL (Pourreza et al., 2024) and XiYan-SQL (Gao et al., 2025b) to achieve state-of-the-art (SOTA) performance. However, two limitations hinder their practical adoption. First, main-

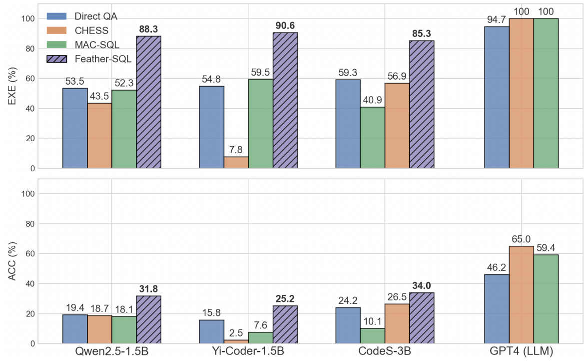

Figure 1: NL2SQL performance on the BIRD DEV dataset. <u>EXE</u> (Executability) measures successful query executions, while <u>ACC</u> (Accuracy) measures correct result matches.

stream methods depend on closed-source models, and their reliance on external APIs introduces data privacy risks in sensitive domains like healthcare and finance (Liu et al., 2024). Second, most open-source research focuses on models with 7B–30B parameters, leaving small language models (SLMs) with 4B or fewer parameters relatively underexplored. Meanwhile, many relational databases are deployed on high-performance systems with limited GPU resources.

In this paper, we focus on enhancing NL2SQL performance using SLMs. As shown in Figure 1, SLMs face two key challenges: (i) one critical issue is their sharp decline in executability. Unlike LLMs, which can effectively handle long-context dependencies, SLMs struggle with complex database schema and verbose prompts, often leading to hallucinated outputs (Nguyen et al., 2024; Qu et al., 2024) (Figure 2); (ii) existing frameworks for NL2SQL with LLMs are incompatible with SLMs, as they rely on strong instruction-following capabilities to produce intermediate results, which SLMs lack. As illustrated in Figure 3, SLMs' outputs frequently violate imposed requirements: they often fail to conform to JSON or array specifications and do not meet predefined constraints. Directly

*Corresponding author.

Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 2069–2086

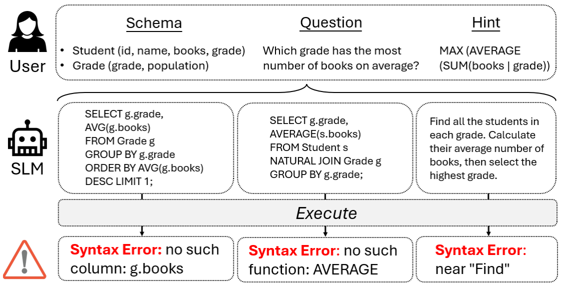

Figure 2: Examples of typical syntax errors produced by small language models (SLMs) in an NL2SQL scenario.

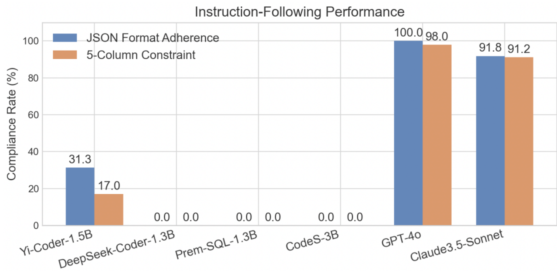

Figure 3: Experiments on CHESS-provided BIRD subset for schema linking. Models are required to output a JSON-formatted response containing no more than five relevant columns related to the question.

applying these frameworks to SLMs may further degrade executability.

To address these challenges, we propose **Feather-SQL**, a lightweight framework tailored for SLMs to enhance both executability and accuracy. It consists of six key components: schema pruning, schema linking, multi-path generation, multi-candidate generation, correction, and selection. Designed for SLMs, schema pruning discards irrelevant tables, allowing models to concentrate on essential database elements. Schema linking improves alignment between questions and database schema, ensuring accurate column selection. To mitigate errors from linking and pruning, multi-path generation explores diverse query formulation strategies, enhancing robustness. Multi-candidate generation further improves the model's executability and accuracy by enhancing the variety of generated SQL queries, thereby increasing the likelihood of producing correct candidates. The best candidate is then selected through execution validation and ranking. Complementing these components, we introduce extraction and simplification prompting strategies. Extraction selectively retrieves key information, while simplification removes extraneous prompt details to lower computational overhead. By integrating these techniques, Feather-SQL enables SLMs to generate SQL queries more reliably despite their inherent limitations.

A common approach to enhancing SLMs is fine-tuning. However, while fine-tuned SLMs for SQL generation tasks (e.g., Prem-SQL (Anindyadeep, 2024), CodeS (Li et al., 2024)) outperform general-purpose chat models on SQL generation, they suffer from catastrophic forgetting (Luo et al., 2025; Kotha et al., 2024) on auxiliary tasks—where task-specific fine-tuning erodes their foundational reasoning abilities. To counter this, we propose **1+1 Model Collaboration Paradigm**, in which a general-purpose chat model handles reasoning-intensive auxiliary tasks (e.g., schema linking and candidate selection), while a fine-tuned SQL model focuses on SQL generation. This collaboration leverages both models' strengths: the chat model provides broad reasoning ability, while the SQL model delivers domain-specific precision. Experiments confirm that the paradigm improves overall performance. Our main contributions are as follows:

- We introduce Feather-SQL, an NL2SQL framework for SLMs to address their unique challenges of low executability and incompatibility with existing LLM-based frameworks.
- We propose a novel 1+1 Model Collaboration paradigm that mitigates catastrophic forgetting in fine-tuned SLMs by delegating reasoning-intensive tasks to a general-purpose chat model.
- Experiments on Spider and BIRD demonstrate that Feather-SQL consistently achieves strong performance with various SLMs and yields SOTA results on BIRD within the scope of SLMs when paired with the paradigm.

2 Related Work

2.1 Conventional Methods

Early NL2SQL methods were rule-based or template-based (Zelle and Mooney, 1996; Li and Jagadish, 2014; Saha et al., 2016). Although effective on small datasets, these approaches demanded extensive manual engineering and did not generalise well. The arrival of sequence-to-sequence (Seq2Seq) models marked a shift to data-driven methods. Models such as Seq2SQL (Zhong et al., 2017), SQLNet (Xu et al., 2017), IRNet (Guo et al., 2019), RyanSQL (Choi et al., 2021), and RESDSQL (Li et al., 2023a) jointly encode the

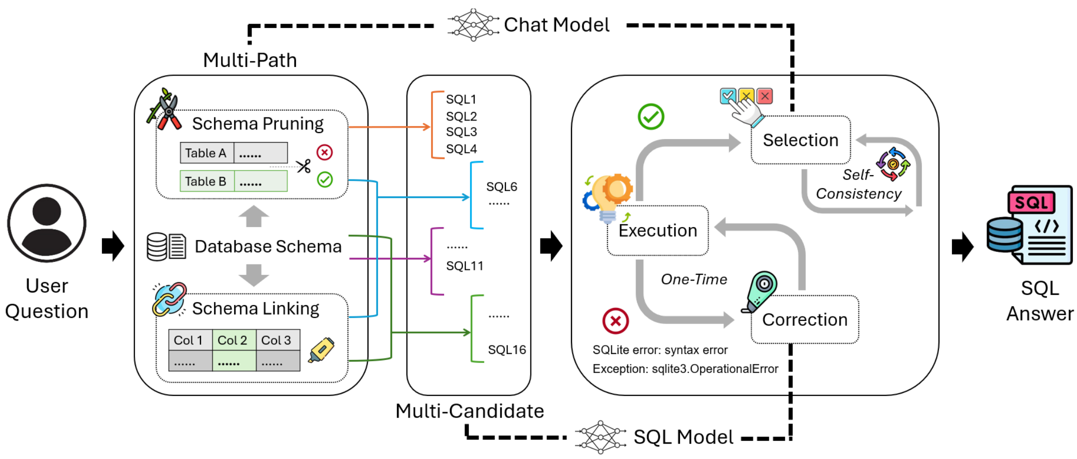

Figure 4: An overview of **Feather-SQL** for small language models (SLMs) in NL2SQL. The **1+1 Model Collaboration Paradigm** pairs a general-purpose chat model with a SQL fine-tuned model: the chat model conducts the multi-path and selection stages (upper dashed links), while the SQL model performs the multi-candidate and correction stages (lower dashed links).

natural-language question and database schema before decoding the corresponding SQL query. Fine-tuning pretrained language models (PLMs) including BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020) further improves robustness, yet still requires substantial annotated data and struggles with highly complex schemas.

2.2 LLM and SLM Approaches

Instruction-tuned large language models (LLMs) now achieve state-of-the-art performance by decomposing NL2SQL into subtasks. Methods such as DIN-SQL (Pourreza and Rafiei, 2023), TASQL (Qu et al., 2024), MAC-SQL (Wang et al., 2024), and CHESS (Talaei et al., 2024) exceed earlier accuracy, but their multi-stage prompting incurs significant computation, and potential privacy risks when queries leave the user's environment.

To alleviate these drawbacks, researchers have turned to small language models (SLMs). Approaches such as CodeS (Li et al., 2024), DTS-SQL (Pourreza and Rafiei, 2024), Prem-SQL (Anindyadeep, 2024), and SQLCoder (Defog) fine-tune SLMs on NL-to-SQL datasets. However, training makes them susceptible to catastrophic forgetting, diminishing their compositional-reasoning ability. MSc-SQL (Gorti et al., 2025) trains separate around 10B models for different subtasks to preserve capabilities, but at the expense of extra memory and storage, limiting practical deployment. Therefore, a lightweight framework that empowers SLMs to perform NL2SQL effectively remains an open and important research goal.

3 Methodology

3.1 Feather-SQL

As shown in Figure 4, we propose Feather-SQL to enhance the performance of SLMs in NL2SQL. We will elaborate on its six components in the following sections.

Schema Pruning. This step reduces schema complexity by filtering out tables irrelevant to the question. Only the Data Definition Language (DDL) statements of tables judged pertinent advance to later stages, preventing SLMs from being overwhelmed by lengthy inputs while preserving essential information. Although it was previously explored by Jose and Cozman (2023) who applied it as a training-time preprocessing driven by statistical analysis, our approach performs it at inference-time using one SLM.

Schema Linking. This step aligns the question with the database schema by identifying relevant columns (Guo et al., 2019). As a commonly adopted practice, schema linking extracts pertinent columns from complete schema (Wang et al., 2020; Talaei et al., 2024). By establishing precise mappings between questions and database elements, this process significantly enhances SQL generation accuracy.

Multi-Path Generation. This step employs four distinct prompt types: (1) with both schema linking and pruning, (2) linking only, (3) pruning only, and (4) without either operation. The multi-path design mitigates the risk of information loss caused by pruning errors and reduces potential misunder-

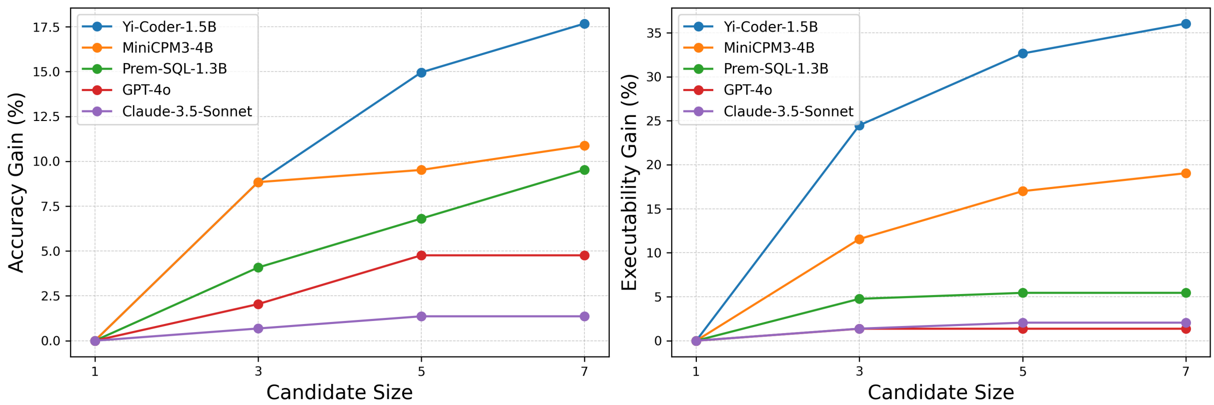

Figure 5: Accuracy gain and executability gain by candidate size. Gains are measured as the percentage-point difference from each model's performance with a single candidate. For both metrics, a set of candidates is counted as correct or executable if at least one candidate in the set meets the criterion.

standings arising from linking inaccuracies.

Multi-Candidate Generation. This step generates multiple SQL queries in parallel to increase the likelihood of producing a correct result (Pourreza et al., 2024; Gorti et al., 2025). To ensure diversity, beam search is employed alongside carefully tuned temperature and top-p parameters. Each path consistently generates a fixed number of candidate queries, maintaining a balanced exploration of possible solutions.

As shown in Figure 5, increasing candidate size yields consistent improvements in both accuracy and executability for SLMs, with notably larger gains compared to LLMs. Larger models are already robust with a single candidate and show only marginal improvements when more candidates are provided. (Details in Appendix B.)

Correction. This step executes each generated query and handles it based on the outcome (Wang et al., 2024; Pourreza and Rafiei, 2023). If a query executes successfully, it is directly added to the array of executable SQL queries. For failed queries, error feedback is used to revise the query through a self-correction approach, generating two new candidate queries. If any of these revised queries are executable, they are also stored in the set of executable SQL queries.

Selection. This step applies a selection-ranking method to assess all executable queries according to their alignment with the expected answers (Pourreza et al., 2024; Gao et al., 2025b; Talaei et al., 2024). If a query yields a limited number of results, the evaluation considers both the query and its execution outcome. In contrast, the evaluation focuses solely on the query itself. The selection process is repeated three times, and the mode of the rankings is returned as the final result.

Method	Stage	Words
CHESS	Information Retrieval	423
	Schema Selection	2522
	Candidate Generation	4888
	Revision	1835
MAC-SQL	Selection	552
	Decomposition	836
	Revision	174
Feather-SQL	Schema Pruning	267
	Schema Linking	287
	Generation	190
	Correction	106
	Selection	271

Table 1: Prompt length comparison.

3.2 Prompting Strategies

Extraction. We propose an extraction strategy to avoid rigid structural outputs by allowing SLMs to freely generate responses. We have two methods to achieve that: (i) Lexical Matching: identify schema elements by matching table/column names in the response. For instance, when the SLM outputs "required tables include customer and orders", the framework extracts customer/orders if they exist in the schema. (ii) Pattern Matching: extract the final answer by identifying predefined patterns in the model's output, such as "answer is" or "Answer:". For example, if the model generates “The answer is 128", the framework extracts "128" as the final result.

Method	Qwen2.5-0.5B		Yi-Coder-1.5B		DeepSeek-Coder-1.3B	
	EX (%)	EP (%)	EX (%)	EP (%)	EX (%)	EP (%)
DR	6.71	<u>26.99</u>	15.84	54.82	29.27	64.41
FEQ	<u>9.65</u>	29.14	<u>18.71</u>	<u>73.60</u>	<u>30.38</u>	67.67
MAC-SQL	2.54	26.40	7.63	59.52	29.99	<u>77.64</u>
CHESS	0.91	4.82	2.48	7.82	18.12	32.97
Feather-SQL (Ours)	**12.52**	**30.46**	**25.23**	**90.61**	**36.64**	**83.70**
Method	**MiniCPM3-4B**		**Prem-SQL-1.3B**		**CodeS-3B**	
	EX (%)	EP (%)	EX (%)	EP (%)	EX (%)	EP (%)
DR	27.57	69.30	47.07	88.14	24.19	<u>59.32</u>
FEQ	29.34	63.89	**51.63**	<u>92.70</u>	25.03	57.50
MAC-SQL	<u>37.35</u>	<u>81.68</u>	8.67 (*8.87**)	17.01 (*19.23**)	10.10 (*13.23**)	40.87 (*56.26**)
CHESS	28.42	54.43	24.64	43.22	<u>26.53</u>	56.91
Feather-SQL (Ours)	**40.09**	**87.02**	<u>49.28</u>	**98.04**	**33.96**	**85.31**

Table 2: Comparison of EX (Execution Accuracy) and EP (Execution Proportion) across different methods on the BIRD DEV dataset. The best and second-best results are highlighted by **bold** and <u>underline</u>, respectively. * denotes results with the extraction strategy.

Simplification. We reduce computational overhead by minimizing prompt verbosity while keeping the task unambiguous. In Feather-SQL, we achieve this by removing superfluous details and using concise instructions with the fewest effective examples. This approach refines the input by eliminating unnecessary complexity, avoiding the need for SLMs to process lengthy inputs while maintaining the clarity of the task.

As shown in Table 1, CHESS uses instruction-heavy prompts, and MAC-SQL also exceeds 500 words in 2 stages. Only Feather-SQL stays concise across all stages, balancing context and complexity without burdening SLMs with lengthy inputs.

3.3 1+1 Model Collaboration Paradigm

Our paradigm categorizes NL2SQL pipeline tasks into two types: reasoning-intensive tasks and SQL generation tasks. Reasoning tasks require contextual understanding and adaptability, while SQL generation demands precision in query synthesis. We employ two specialized models to leverage complementary strengths: the general-purpose chat model for reasoning tasks and the SQL fine-tuned model for SQL generation.

General-purpose Chat Model. This model leverages broad linguistic and contextual comprehension without domain-specific fine-tuning, which helps prevent catastrophic forgetting. Compared to the SQL fine-tuned model, it is more effective in schema linking and candidate evaluation, ensuring that the generation process is guided by accurate and well-structured contextual information.

SQL Fine-tuned Model. Optimized exclusively for SQL generation, this model is extensively trained on large-scale NL2SQL datasets, allowing it to achieve superior performance on SQL-specific tasks. Its focused training reduces hallucinations and enhances both query executability and accuracy.

4 Experiments

4.1 Settings

4.1.1 Datasets

BIRD (Li et al., 2023b) encompasses databases over 37 professional domains. Due to the proprietary nature of the BIRD TEST dataset, we conduct our experiments using the publicly accessible BIRD DEV subset, which contains 1,534 unique question-SQL pairs.

Spider (Yu et al., 2019) is another large-scale benchmark dataset for cross-domain SQL generation, covering 138 different domains. Our experiments utilize the Spider TEST set, comprising 2,147 question-SQL pairs.

4.1.2 Evaluation Metrics

Execution Accuracy (EX) (Li et al., 2023b) is a widely adopted metric in NL2SQL evaluations, measuring whether the result of executing the generated query matches the result of the ground-truth query. This metric allows for different query formulations that yield the same result. It is calculated as:

$$\mathrm{EX} = \frac{|\{n \in N \mid \mathrm{E}(Q_{gen}) = \mathrm{E}(Q_{\mathrm{gt}})\}|}{N} \times 100\%$$

Method	Qwen2.5-0.5B EX (%)	Qwen2.5-0.5B EP (%)	Yi-Coder-1.5B EX (%)	Yi-Coder-1.5B EP (%)	DeepSeek-Coder-1.3B EX (%)	DeepSeek-Coder-1.3B EP (%)
DR	28.50	56.45	45.23	87.24	49.28	90.68
FEQ	36.53	67.35	48.30	86.77	45.46	89.89
MAC-SQL	29.06	**89.61**	13.04	21.70	**52.12**	93.62
CHESS	15.42	29.16	3.68	10.29	30.18	46.30
Feather-SQL (Ours)	**36.98**	75.08	**49.56**	**92.04**	51.19	**94.13**
Method	**MiniCPM3-4B EX (%)**	**MiniCPM3-4B EP (%)**	**Prem-SQL-1.3B EX (%)**	**Prem-SQL-1.3B EP (%)**	**CodeS-3B EX (%)**	**CodeS-3B EP (%)**
DR	55.10	93.71	60.92	85.79	47.74	64.23
FEQ	55.75	89.52	64.23	85.75	49.60	64.65
MAC-SQL	25.01	38.47	0.14 (*67.91**)	0.14 (*100**)	0 (*74.48**)	0 (*100**)
CHESS	56.73	89.99	63.86	92.08	**66.65**	88.54
Feather-SQL (Ours)	**58.92**	**94.18**	**66.60**	**92.78**	63.25	**88.96**

Table 3: Comparison of EX (Execution Accuracy) and EP (Execution Proportion) across different methods on the Spider TEST dataset. The best and second-best results for EX are highlighted by **bold** and underline, respectively. * denotes results with the extraction strategy.

where N denotes the number of questions. Q_{gen} represents the SQL query generated by the model, while Q_{gt} is the ground-truth query. E is the execution function.

Execution Proportion (EP) is an auxiliary metric we proposed, evaluating the proportion of generated SQL queries that can be executed on the corresponding database without syntax errors. This metric reflects the model's upper-bound capability by assuming that any executable query is potentially correct. It is defined as:

$$\text{EP} = \frac{|\{n \in N \mid \text{E}(Q_{\text{gen}}) \neq \text{error}\}|}{N} \times 100\%$$

4.1.3 Baselines

Direct Response (DR) directly generates an SQL query from the natural language question without applying any refinement techniques. The process follows a single-turn interaction.

First Executable Query (FEQ) leverages the model's ability to generate multiple SQL candidates. Among candidates, the first executable query is selected without any refinement. This approach simulates multi-turn query generation.

MAC-SQL (Wang et al., 2024) is an LLM-based multi-stage framework, featuring a core Decomposer agent for SQL generation supported by auxiliary agents for sub-database acquisition and query refinement. It also utilizes few-shot chain-of-thought reasoning to enhance generation processes.

CHESS (Talaei et al., 2024) comprises four specialized agents: Information Retriever, Schema Selector, Candidate Generator, and Unit Tester. Notably, it employs locality-sensitive hashing and vector databases to efficiently retrieve relevant data from extensive database values and catalogs.

4.1.4 Implementation Details

Backbone Models. Our implementation leverages both general-purpose chat models and SQL fine-tuned models. The chat models include Qwen2.5-0.5B, Qwen2.5-1.5B, Qwen2.5-Coder-1.5B (Hui et al., 2024), Yi-Coder-1.5B (Young et al., 2025), DeepSeek-Coder-1.5B (DeepSeek-AI, 2024) and MiniCPM3-4B (Hu et al., 2024), while the SQL models consist of Prem-SQL-1.3B (Anindyadeep, 2024) and CodeS-3B (Li et al., 2024).

Candidate Size. In the multi-candidate generation stage, we generate 4 candidates per path, resulting in a total candidate pool of 16. During the correction stage, the candidate size is reduced to 2.

Selection Rounds. During the selection stage, we perform 3 rounds for each selection. The final choice is the majority vote across the three rounds, ensuring consistency of the selected candidate.

4.2 Main Results

4.2.1 Feather-SQL

To validate the general effectiveness of Feather-SQL for SLMs, we conducted experiments on two datasets across a range of models (all results here were obtained using a unified model without adopting the collaboration paradigm).

BIRD Results. As shown in Table 2, Feather-

Chat Model	SQL Model	EX (%)	EP (%)
–	Prem-SQL	49.28	98.04
Qwen	Prem-SQL	52.44 ↑	94.08
Qwen Coder	Prem-SQL	52.83 ↑	98.31
Yi Coder	Prem-SQL	54.76 ↑	93.94
–	CodeS	33.96	83.31
Qwen	CodeS	35.79 ↑	80.05
Qwen Coder	CodeS	37.03 ↑	81.10
Yi Coder	CodeS	39.43 ↑	80.44

(a) Feather-SQL

Chat Model	SQL Model	EX (%)	EP (%)
–	Prem-SQL	24.64	43.22
Qwen	Prem-SQL	49.28 ↑	82.07
Qwen Coder	Prem-SQL	49.61 ↑	79.60
Yi Coder	Prem-SQL	47.65 ↑	79.79
–	CodeS	26.53	56.91
Qwen	CodeS	28.55 ↑	56.19
Qwen Coder	CodeS	28.88 ↑	63.04
Yi Coder	CodeS	27.44 ↑	55.22

(b) CHESS

Table 4: Paradigm performance on the BIRD DEV dataset. When no chat model is specified, the SQL model is used as the chat model. Qwen refers to Qwen2.5-1.5B, Qwen Coder refers to Qwen2.5-Coder-1.5B, Yi Coder refers to Yi-Coder-1.5B, Prem-SQL refers to Prem-SQL-1.3B, and CodeS refers to CodeS-3B.

SQL demonstrates superior performance across all general-purpose chat models, achieving the highest scores in both EX and EP, with EX showing an average increase of approximately 10% and EP exceeding a 20% improvement compared to FEQ. For SQL fine-tuned models, Feather-SQL combined with CodeS achieves substantial gains in both EX and EP, while Prem-SQL shows notable improvements specifically in EP, with an average increase of around 5% compared to FEQ. Besides, we explored the upper bound of Feather-SQL on this dataset (Appendix C).

Moreover, we observe that CHESS and MAC-SQL do not perform effectively on SLMs, with their results on Qwen2.5 and Yi-Coder showing even lower EX and EP scores compared to DR. Their performance also falls behind that of FEQ.

Spider Results. Table 3 highlights the results on Spider TEST. Although MAC-SQL and CHESS show inconsistent performance across models, MAC-SQL generally performs well. Notably, for SQL fine-tuned models, MAC-SQL could achieve the best EX if extraction is applied, highlighting the necessity of this step. This may be attributed to MAC-SQL's Selector mechanism, which also employs schema pruning. Unlike our table pruning approach, MAC-SQL adopts column pruning, which may be more effective for Spider's relatively simple schema structures.

4.2.2 1+1 Model Collaboration Paradigm

As observed in Table 2, although Feather-SQL improves the EP of Prem-SQL, its EX shows a 2% decrease compared to FEQ. This decline is primarily due to Prem-SQL's inability to handle auxiliary reasoning tasks. To address this limitation, we propose a division of tasks where the general-purpose chat model handles auxiliary reasoning, while the SQL fine-tuned model focuses on SQL generation.

As shown in Table 4a, our 1+1 collaboration paradigm under Feather-SQL achieves a 3–6% improvement in EX for both Prem-SQL and CodeS. However, we observe a decline in EP when paired with a chat model. This is because when the SQL model is also used as the chat model during schema pruning, it returns a query instead of the expected answer. But our extraction strategy still retrieves table names from the output, often resulting in an overly pruned schema containing only one or two tables. While a simplified schema can occasionally boost EP, it frequently leads to lower overall EX.

Additionally, Table 4b shows that our paradigm improves both Prem-SQL and CodeS in CHESS, with EX increasing by ~20% and EP by over ~35% for Prem-SQL, while CodeS sees a smaller but consistent EX gain with no clear trend in EP.

However, the two models benefit differently due to their handling of auxiliary tasks. Prem-SQL attempts to answer linking questions but often does so incorrectly, whereas CodeS, due to severe catastrophic forgetting, fails to provide valid responses. As a result, CHESS defaults to using the original schema with CodeS, reducing linking errors.

Furthermore, since CHESS constructs long prompts without schema pruning, introducing a chat model increases input length and complexity. While this improves reasoning, it does not fully offset CodeS's limitations in processing extended inputs, restricting its EX improvement.

SOTA within SLMs. To contextualize our re-

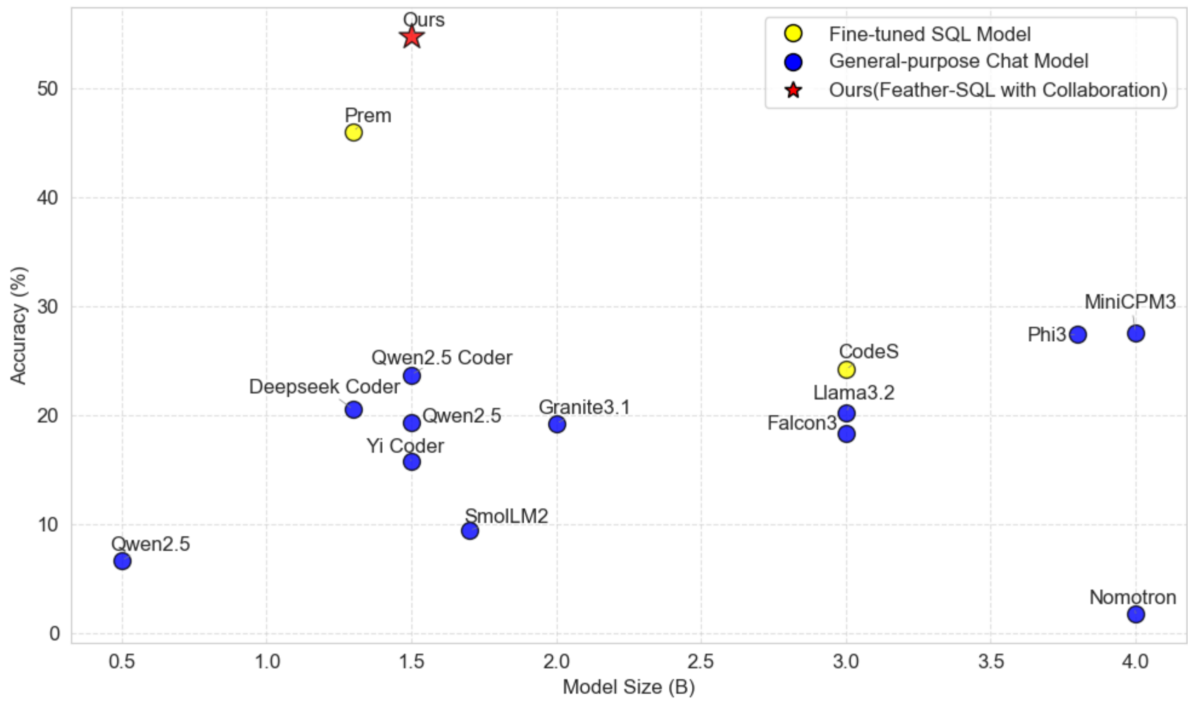

Figure 6: Accuracy (%) versus model size (billions of parameters) on BIRD DEV for small language models. Fine-tuned SQL models are shown in yellow, general-purpose chat models in blue, and ours (Feather-SQL + 1+1 Model Collaboration) is marked with a red star.

sults, we further compare against widely used open-source SLMs beyond our backbones—namely Granite-3.1B (Mishra et al., 2024), SmolLM2-1.7B, Llama3.2-3B (Dubey et al., 2024), Falcon-3B (Gao et al., 2025a), and Nemotron-4B (Nvidia et al., 2024). Figure 6 shows that combining Feather-SQL with the 1+1 Model Collaboration paradigm yields state-of-the-art accuracy among small language models.

4.3 Ablation Studies

4.3.1 Component Contribution

Framework	EX (%)	EP (%)
Full Model	31.81	88.33
–w/o Schema Pruning	-4.63 ↓	-20.34 ↓
–w/o Schema Linking	-3.45 ↓	-20.92 ↓
–w/o Multi-Candidate	-2.47 ↓	-17.99 ↓
–w/o Correction	-0.20 ↓	-12.58 ↓
–w/o Selection	-2.21 ↓	-10.36 ↓

Table 5: Ablation Study on Framework Components.

We conducted an ablation study to quantify the impact of each framework component by removing them one at a time and measuring changes in EX and EP on the BIRD DEV dataset, using Qwen2.5-1.5B (Table 5).

We can see from the ablation results that removing any of the components causes a drop in both EX and EP. This underscores that each step in our pipeline contributes to overall performance, and omitting even one module leads to noticeably reduced accuracy or executability.

Among these, schema pruning is shown to be the most critical: when it is removed, EX falls from 31.81% to 27.18%, the single largest drop in our study. This highlights how focusing on only the relevant tables and columns helps the model concentrate on essential schema elements, thereby yielding more accurate SQL generation. In contrast, removing correction only reduces EX by 0.20%, indicating that it has a relatively minor impact on the framework's effectiveness.

4.3.2 Path Contribution

We analyzed origins of SQL answers from four models to understand how each processing path affects the final output. As shown in Figure 7, our multi-path strategy includes four paths: one using both schema linking and pruning, one using only schema linking, one using only schema pruning,

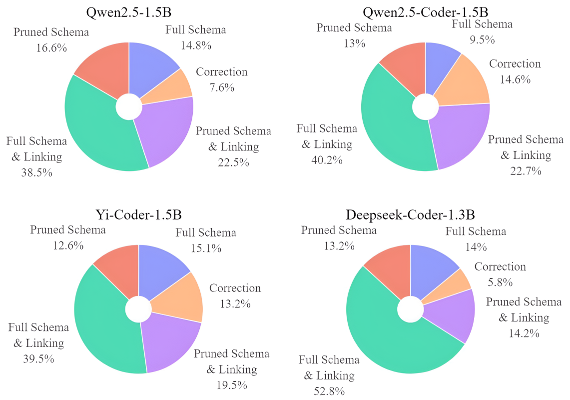

Figure 7: Distribution of correct SQL answers contributed by each path across four different SLMs.

and one without either.

For all four models, the path *Full Schema & Linking* is consistently the largest contributor, followed by *Pruned Schema & Linking*. Additionally, we find that schema pruning collectively accounts for over 25%. These observations further underscore the critical roles of schema linking and pruning.

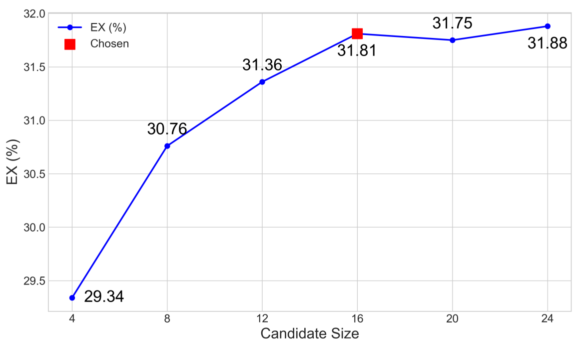

Figure 8: Effect of candidate size on EX performance.

4.3.3 Candidate Size

We further investigated the impact of different candidate sizes. Figure 8 presents the results based on our four paths. In our experiments, the total candidate size increases from 4 to 24, which corresponds to the number of candidates generated per path increasing from 1 to 6. The figure illustrates how EX changes as the overall candidate size grows from 4 to 24.

We observe a concave trend, consistent with Figure 5: EX steadily increases as the candidate size rises from 4 to 16 but then plateaus from 16 to 24. Once the model reaches its approximate upper bound, further increases in candidate size result in only a marginal difference in performance. Therefore, we select a candidate size of 16, as it is the earliest point at which EX saturates, thus balancing computational efficiency and model performance.

4.3.4 Selection Rounds

Rounds R	EX (%)
1	29.40
3	**31.81**
5	31.23
7	31.49

Table 6: Effect of selection rounds on EX.

We repeat the selection step multiple times and choose the SQL that appears the most frequently (mode). As shown in Table 6, EX improves from a single round to three rounds, after which it largely plateaus. Accordingly, we use three rounds by default.

5 Conclusion

Small language models (SLMs) are appealing for NL2SQL because they are easier to deploy and safer for data privacy. However, they are underexplored and often struggle with low executability and incompatibility with existing LLM-based frameworks. To address these limitations, we propose **Feather-SQL**, a lightweight framework for SLMs that includes six key components: schema pruning, schema linking, multi-path generation, multi-candidate generation, correction, and selection. We further introduce a **1+1 Model Collaboration Paradigm** that pairs a general-purpose chat model for reasoning tasks with an SQL fine-tuned model for SQL generation, leveraging complementary strengths.

We conduct extensive experiments to evaluate our method. Across multiple models on BIRD DEV and Spider TEST, Feather-SQL alone yields consistent gains in Execution Accuracy (EX) and Execution Proportion (EP). Compared to the second-best method, EX improves by about 10% and EP by more than 20%. Furthermore, with the 1+1 Model Collaboration Paradigm, Feather-SQL attains state-of-the-art (SOTA) within SLMs on BIRD DEV, with EX reaching 54.76%.

Together, these results demonstrate the effectiveness of our method for NL2SQL with SLMs. Our work also provides a robust foundation for applying SLMs to other structured tasks and domains.

Limitations

Despite the promising performance gains achieved by Feather-SQL, our current framework does not yet reach very high absolute accuracy on datasets. For instance, the best cumulative accuracy on BIRD DEV is around 74% (Gao et al., 2025b; Pourreza et al., 2024). In fact, many LLM-based NL2SQL methods typically report accuracy in the 60+% range, while the SOTA results achieved by SLMs remain below 55%. However, our approach is the first to surpass all previous methods at the 1B-parameter scale. Feather-SQL with the Model Collaboration Paradigm lays a strong foundation for promoting the broader adoption of NL2SQL in real-world applications.

References

Anindyadeep. 2024. Premsql: End-to-end local-first text-to-sql pipelines. https://github.com/premAI-io/premsql. Accessed: 2024-12-10.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. Computational Linguistics, 47(2):309–332.

DeepSeek-AI. 2024. Deepseek llm: Scaling open-source language models with longtermism. Preprint, arXiv:2401.02954.

Defog. Sqlcoder. https://github.com/defog-ai/sqlcoder. Accessed: 2024-12-10.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. Preprint, arXiv:1810.04805.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, and Austen Gregerson. 2024. The llama 3 herd of models. Preprint, arXiv:2407.21783.

Xiangxiang Gao, Weisheng Xie, Yiwei Xiang, and Feng Ji. 2025a. Falcon: Faster and parallel inference of large language models through enhanced semi-autoregressive drafting and custom-designed decoding tree. Preprint, arXiv:2412.12639.

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. 2025b. A preview of xiyan-sql: A multi-generator ensemble framework for text-to-sql. Preprint, arXiv:2411.08599.

Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. 2025. Msc-sql: Multi-sample critiquing small language models for text-to-sql translation. In Proceedings of the Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics, pages 2145–2160.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4524–4535.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, and Yuxiang Huang. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. Preprint, arXiv:2404.06395.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, and Jiajun Zhang. 2024. Qwen2.5-coder technical report. Preprint, arXiv:2409.12186.

Marcelo Archanjo Jose and Fabio Gagliardi Cozman. 2023. A multilingual translator to sql with database schema pruning to improve self-attention. Preprint, arXiv:2306.14256.

Suhas Kotha, Jacob Mitchell Springer, and Aditi Raghunathan. 2024. Understanding catastrophic forgetting in language models via implicit inference. In The Twelfth International Conference on Learning Representations.

Fei Li and Hosagrahar V Jagadish. 2014. Nalir: an interactive natural language interface for querying relational databases. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, page 709–712.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. Preprint, arXiv:2402.16347.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. 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.

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

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. Preprint, arXiv:2308.08747.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, Manish Sethi, Xuan-Hong Dang, Pengyuan Li, Kun-Lung Wu, Syed Zawad, and Andrew Coleman. 2024. Granite code models: A family of open foundation models for code intelligence. Preprint, arXiv:2405.04324.

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. Preprint, arXiv:2410.20011.

Nvidia, :, Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H. Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, Sirshak Das, Ayush Dattagupta, Olivier Delalleau, Leon Derczynski, Yi Dong, Daniel Egert, Ellie Evans, Aleksander Ficek, and Denys Fridman. 2024. Nemotron-4 340b technical report. Preprint, arXiv:2406.11704.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, and Igor Babuschkin. 2024. Gpt-4 technical report. Preprint, arXiv:2303.08774.

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.

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. Preprint, arXiv:2402.01117.

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. Preprint, arXiv:2405.15307.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21(140):1–67.

Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. Athena: an ontology-driven system for natural language querying over relational data stores. Proceedings of the VLDB Endowment, 9(12):1209–1220.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. Preprint, arXiv:2405.16755.

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

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.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. arXiv:1711.04436.

01.AI: Alex Young, Bei Chen, Chao Li, and Chengen Huang. 2025. Yi: Open foundation models by 01.ai. Preprint, arXiv:2403.04652.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. Preprint, arXiv:1809.08887.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, pages 1050–1055.

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

A Experimental Settings

All experiments were conducted on 4 NVIDIA A6000 GPUs using the vLLM inference acceleration framework to improve model efficiency. For stages that produce multiple answers, such as candidate generation and selection, we primarily used a `temperature` of 0.2 and a `top_p` of 0.8 to balance diversity and accuracy. In contrast, for tasks requiring a single answer, such as schema pruning and schema linking, we employed greedy search to ensure deterministic outputs.

B Multi-Candidate Motivation

Top-N	**Yi-Coder-1.5B**		**MiniCPM3-4B**		**Prem-SQL-1.3B**	
	ACC (%)	**EXE (%)**	**ACC (%)**	**EXE (%)**	**ACC (%)**	**EXE (%)**
1	15.65	46.26	26.53	65.31	55.78	92.52
3	24.49	70.75	35.37	76.87	59.86	97.28
5	30.61	78.91	36.05	82.31	62.59	97.96
7	33.33	82.31	37.41	84.35	65.31	97.96
Top-N	**CodeS-3B**		**GPT-4o**		**Claude-3.5-Sonnet**	
	ACC (%)	**EXE (%)**	**ACC (%)**	**EXE (%)**	**ACC (%)**	**EXE (%)**
1	24.49	61.90	51.70	93.20	40.82	86.39
3	27.21	68.71	53.74	94.56	41.50	87.76
5	29.93	72.11	56.46	94.56	42.18	88.44
7	29.93	73.47	56.46	94.56	42.18	88.44

Table 7: Comparison of Accuracy (ACC) and Execution (EXE) on the BIRD DEV Subset from CHESS using multi-candidate generation strategy.

The results demonstrate that SLMs exhibit a performance gap between TOP-1 and TOP-7 results. This indicates that employing a multi-candidate generation strategy can effectively improve the accuracy and execution rates by selecting the best result. In contrast, larger models already perform robustly with TOP-1 outputs, and therefore, the additional benefit from multi-candidate generation is limited. Additionally, the fine-tuned SQL model CodeS-3B shows some improvement, but the gains are not as pronounced as those observed in the other SLMs.

C Framework Upper Bound

To explore the upper bound of the Feather-SQL framework, we also evaluated its performance using cumulative accuracy, which measures whether the correct SQL query is present within the Top-n generated results. Specifically, we retained the top 4 candidates after the selection ranking in this experiment, rather than solely selecting the top 1 candidate in default.

As indicated in Table 8, Top-3 is approximately 10% higher than Top-1 (EX). This suggests that there is room for further improvement in the selection mechanism. If the selection can be refined to accurately identify the optimal SQL query, the performance gap between Top-N and Top-1 could be considerably reduced.

Model	**Top-1(%)**	**Top-2(%)**	**Top-3(%)**
Qwen	31.8	39.0	40.5
Yi Coder	25.2	32.6	34.5
Prem-SQL	49.2	60.2	62.6

Table 8: Cumulative Accuracy on BIRD DEV.

D Prompts

D.1

Schema Pruning Prompt

```
prompt_pruning_system = """
You are an agent designed to find all related tables to generate SQL query
for question based on the database schema and hint.

## Requirements
1. You don't need to answer the question, your task is only finding all related tables .
2. Consider all constraints of each table, including primary keys, foreign keys, and data
     types.
3. You can generate chain of thoughts, but ensure all tables mentioned truly exist.
4. Successfully answer related columns could help you win $100000 dollars.
"""

prompt_pruning = """
## Instructions
1. Prioritize the table that most directly contains the information needed to answer the
     question, considering:
    - Table relationships such as foreign keys.
    - Whether the table has columns directly related to the entities or actions in the
     question.
2. Reasoning like two shown examples.

----------Example----------
## Database Schema
CREATE TABLE Employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(100),
    salary DECIMAL(10, 2)
);

CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100),
    location VARCHAR(100)
);

## Question
What is the salary of the employee named 'Alice'?

## Relevant Tables
This table directly contains the columns name and salary, which are the only necessary fields
     to answer the question.
The name column is used to locate the specific employee named 'Alice', and the salary column
     provides the required
salary information. The Departments table is irrelevant because it does not store employee-
     level data like salaries
or names, and its information is unrelated to this specific query.
The relevant table is Employees.

----------Task----------
## Database Schema
You are provided with the structure of the database "{database_name}":
{database_schema}

## Question
{question}

## Hint
{hint}

Among the following tables: {tables}, which tables are relevant for addressing the question?
## Relevant Tables
```

```
"""
```

D.2

Schema Linking Prompt

```
prompt_linking_system="""
You are an agent designed to find all related columns to generate SQL query for question based
      on the database schema and the hint.

## Requirements
1. You don't need to answer the question, your task is only finding all related columns.
2. Hint could help you to find the correct related columns.
3. Consider all constraints of each table, including primary keys, foreign keys, and data
     types.
4. You can generate chain of thoughts, but ensure all columns mentioned truly exist.
7. Successfully answer related columns could help you win $100000 dollars.
"""

prompt_linking="""
## Instructions
1. Select columns that relates to information requested by the question, considering:
    - Whether the column is key to filtering results (used in WHERE clauses).
    - Whether the column should be part of the SELECT statement to fulfill the user query.
    - The relationship of the column to other parts of the question, such as groupings,
     aggregations, or direct match to entities mentioned.
2. Reasoning like two shown examples.

----------Example----------
## Database Schema
CREATE TABLE Employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(100),
    salary DECIMAL(10, 2)
);

CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100),
    location VARCHAR(100)
);

## Question
What is the salary of the employee named 'Alice'?

## Relevant Columns
The name column is essential to filter the employee named 'Alice' in the WHERE clause,
     ensuring we identify the correct individual. The salary column is needed to extract the
     requested information, which is the employee's salary. Since the question does not
     involve departments, the Departments table and its columns are irrelevant.
The related columns are Employees.name and Employees.salary.

----------Task----------
## Database Schema
You are provided with the structure of the database "{database_name}":
{schema}

## Question
{question}

## Hint
{hint}

Among the columns, which are relevant for addressing the question?
## Relevant Columns
```

```
"""
```

D.3

Multi-path Generation Prompt

```
system_prompt_sql_generation = """
You are an expert SQL assistant tasked with generating precise SQL queries based on given
    database schemas, questions, and hint.

## Responsibilities
1. Analyze the **database schema** and **hint** to determine relationships, including **
    primary keys, foreign keys, data types, and constraints**.
2. Generate a single, valid **SQLite SQL query** to answer the question, using provided schema
     linking information for table and column selection.
3. Your response should contain only the **SQL query**, using standard SQL syntax with correct
     use of table/column names and SQL clauses.

## Requirements
- Respond with only one SQL query, formatted as ```SQL```.
- Use clauses like **SELECT**, **FROM**, **WHERE**, **JOIN**, **GROUP BY**, **ORDER BY**, etc.
- Ensure SQL is efficient and respects **Important Columns**, table relationships, and
    relevant constraints.
"""

prompt_generation_with_linking = """
You are given a database schema, question, important columns and hint. Generate a valid SQLite
     query that answers the question.

## Instructions
1. Your response should only contain one SQL query, in standard SQL syntax.
2. Consider all **table relationships**, **primary/foreign keys**, **data types**, and **
    Important Columns** while generating the query.

## Database Schema
Database "{database_name}":
{database_schema}

## Important Columns
{schema_linking}

## Question
{question}

## Hint
{hint}

## Output Requirement
Format the response as:
```sql
[SQL query]
```
"""

prompt_generation_without_linking = """
You are given a database schema, question, and hint. Generate a valid SQLite query that
    answers the question.

Instructions
1. Your response should only contain one SQL query, in standard SQL syntax.
2. Consider all **table relationships**, **primary/foreign keys**, **data types** while
    generating the query.

Database Schema
Database "{database_name}":
{database_schema}
```

```
## Question
{question}

## Hint
{hint}

## Output Requirement
Format the response as:
```sql
[SQL query]
```
"""
```

D.4

Correction Prompt

```
prompt_answer_correction_system ="""
Suppose you are an expert in SQLite and database management.

## Instructions
1. Based on the database structure provided, previous answer and its error messages, generate
    one SQL query that answers the question.
2. You should try to fix the error of the previous answer and avoid it from happening again.

## Requirements
1. Your response should consist of only one SQL query, don't generate anything else.
3. Consider all constraints of each table, including primary keys, foreign keys, and data
    types.
4. Provide your query in standard SQL format with appropriate use of SQL functions, joins, and
     conditions.
"""

prompt_answer_correction = """
## Database Schema
Given the structure of database:
{schema}

## Question
{question}

## Hint
{hint}

## Previous answer
{prev_ans}

## Error
{errorMsg}

## New Answer
"""
```

D.5

Selection Prompt

```
system_prompt_query_selection = """
You are an expert in analyzing SQL queries and determining their relevance to a given question.
     Your task is to evaluate multiple SQL queries and select the one that best answers the
    question based on the provided database schema and context.

## Responsibilities
```

```
1. Analyze the given question: Understand the intent of the question and its expected output.
2. Evaluate each SQL query: Consider the correctness, relevance, and completeness of each
    query in relation to the question.
3. Select the best query: Choose the query that most accurately answers the question, while
    considering database structure, table relationships, and query efficiency.

## Requirements
- Respond with the most relevant SQL query, and nothing else.
- Ensure the selected query is valid for the given database schema and directly addresses the
    question.
"""

query_selection_prompt = """
You are given a question, a database schema, and multiple SQL queries. Your task is to select
    the SQL query that is most relevant and best answers the question.

## Instructions
1. Analyze the Question: Understand what the user is asking and identify the information that
    needs to be extracted from the database.
2. Evaluate SQL Queries: For each provided SQL query, determine its relevance based on:
    - Accuracy: Does the query correctly match the question's intent?
    - Completeness: Does the query retrieve all the necessary information without omitting
    important details?
    - Efficiency: Is the query optimized for the task, avoiding unnecessary joins or
    conditions?
3. Select the Most Relevant Query: Choose the query that is the best match for the question.

## Database Schema
Database "{database_name}":
{database_schema}

## Question
The question is:
{question}

## Hint
{hint}

## SQL Queries
{queries}

## Output Requirement
Reply the query Index in the format of "Index: ".

## Output
"""

query_with_response_selection_prompt = """
You are given a question, a database schema, multiple SQL queries, and their execution results.
     Your task is to select the SQL query that best answers the question based on the query
    and its result.

## Instructions
1. Understand the Question: Determine what the user is asking and identify the specific
    information that needs to be retrieved.
2. Evaluate Each Query and Response Pair: For each provided SQL query and its result,
    determine:
    - Query Accuracy: Does the query correctly represent the user's intent?
    - Result Relevance: Does the result contain the data needed to answer the question
    completely and correctly?
    - Efficiency: Is the query optimized, avoiding unnecessary complexity?

## Database Schema
Database "{database_name}":
{database_schema}

## Question
{question}
```

```
## Hint
{hint}

## SQL Queries and Execution Results
{queries}

## Output Requirement
Only reply the query Index in the format of "Index: ".
"""
```