Agentar-Scale-SQL: Advancing Text-to-SQL through Orchestrated Test-Time Scaling

Pengfei Wang[1*], **Baolin Sun**[1*†], **Xuemei Dong**[1*], **Yaxun Dai**[2*‡], **Hongwei Yuan**[3*‡], **Mengdie Chu**[1], **Yingqi Gao**[1], **Xiang Qi**[1†], **Peng Zhang**[1†], **Ying Yan**[1]
[1]Ant Digital Technologies, Ant Group, [2]Soochow University, [3]Zhejiang University
{nanzhou.wpf, xuanfeng.sbl, dongxuemei.dxm, qixiang.qx, minghua.zp}@antgroup.com
https://github.com/antgroup/Agentar-Scale-SQL

Abstract

State-of-the-art (SOTA) Text-to-SQL methods still lag significantly behind human experts on challenging benchmarks like BIRD. Current approaches that explore test-time scaling lack an orchestrated strategy and neglect the model's internal reasoning process. To bridge this gap, we introduce Agentar-Scale-SQL, a novel framework leveraging scalable computation to improve performance. Agentar-Scale-SQL implements an Orchestrated Test-Time Scaling strategy that synergistically combines three distinct perspectives: i) Internal Scaling via RL-enhanced Intrinsic Reasoning, ii) Sequential Scaling through Iterative Refinement, and iii) Parallel Scaling using Diverse Synthesis and Tournament Selection. Agentar-Scale-SQL is a general-purpose framework designed for easy adaptation to new databases and more powerful language models. Extensive experiments show that Agentar-Scale-SQL achieves SOTA performance on the BIRD benchmark, reaching 81.67% execution accuracy on the test set and ranking first on the official leaderboard, demonstrating an effective path toward human-level performance.

1 Introduction

Democratizing access to data analytics by enabling users to query structured databases in their own natural language is a long-standing goal in human-computer interaction. This is the core objective of Text-to-SQL, a pivotal research area focused on translating natural language questions into executable SQL queries (Zhang et al., 2024; Li et al., 2024a; Liu et al., 2024; Luo et al., 2025). By bridging the gap between human language and structured data, Text-to-SQL empowers non-technical users to interact with complex databases effectively, garnering significant interest from both the Natural Language Processing (NLP) and database communities (Li et al., 2023; Yu et al., 2018; Pourreza et al., 2025a; Zhang et al., 2025b).

The ultimate vision for Text-to-SQL is to develop systems that can match, and eventually surpass, human expert performance. However, a significant gap persists between this ambition and the current state-of-the-art. On the challenging BIRD benchmark (Li et al., 2023), human experts achieve an execution accuracy (EX) of 92.96%, whereas even the top-performing methods lag considerably, with the top 5 approaches around 75% on the test set. Closing the vast human-machine performance divide urgently requires innovation.

Recent Text-to-SQL research falls into three main categories. The first consists of prompt-based methods (e.g., OpenSearch-SQL (Xie et al., 2025) and DAIL-SQL (Gao et al., 2024)). The second category is comprised of fine-tuning-based approaches, with representative works like Arctic-Text2SQL-R1-32B (Yao et al., 2025). The third category consists of hybrid methods like XiYan-SQL (Liu et al., 2025c), CHASE-SQL + Gemini (Pourreza et al., 2025a), and Contextual-SQL (Agrawal and Nguyen, 2025). These approaches primarily explore test-time scaling from different perspectives: Contextual-SQL adopts an ensemble strategy, while XiYan-SQL and CHASE-SQL investigate ensemble strategies and sequential refinement. However, these studies share a common limitation, as they neglect both an internal perspective on the model's reasoning process and the orchestrated scaling combination.

To further advance Text-to-SQL performance, this work contends that the most promising path forward lies in fully embracing the principle of "The Bitter Lesson" (Sutton, 2019): that general methods leveraging scalable computation ultimately triumph over those based on complex, human-specified knowledge. With this philosophy, we focus on test-time scaling and have not designed complex strate-

[*]Core contributors.

[‡]This work was partially done during the author's internship at Ant Digital Technologies, Ant Group.

[†]Corresponding authors.

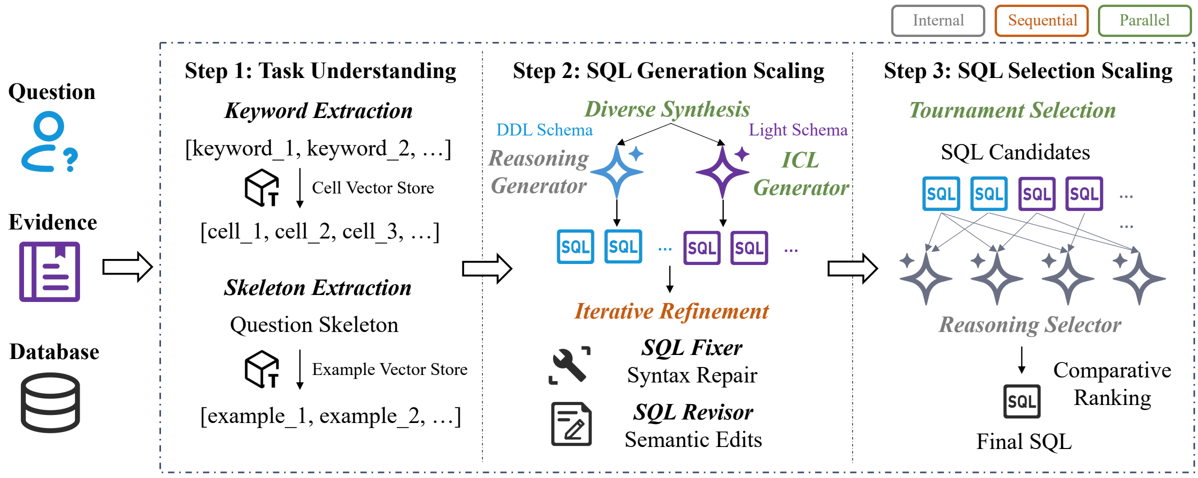


Figure 1: The proposed Agentar-Scale-SQL framework.

gies for schema linking, as we believe that scaling alone is sufficient to improve performance. To this end, we introduce Agentar-Scale-SQL, an *Orchestrated Test-Time Scaling* framework that demonstrates that the path to human-level performance lies not in crafting more intelligent heuristics but in building general-purpose frameworks that effectively leverage the test-time scaling (Snell et al., 2024; Zhang et al., 2025a). Agentar-Scale-SQL employs an orchestrated scaling strategy across three distinct perspectives: i) Internal Scaling (through RL-enhanced Intrinsic Reasoning), ii) Sequential Scaling (through Iterative Refinement), and iii) Parallel Scaling (through both Diverse Synthesis and Tournament Selection). This strategy is realized within a three-stage framework, as depicted in Figure 1. The objective of Step 1 (Task Understanding) is to establish a comprehensive understanding of the input and its context, which is essential for the scaling operations in subsequent steps. Then, Step 2 (SQL Generation Scaling) develops diverse synthesis and iterative refinement to obtain high-quality and diverse SQL candidates. Specifically, diverse synthesis employs two distinct generators (i.e., intrinsic reasoning generator and ICL generator) with generator-specific schema formats. Finally, Step 3 (SQL Selection Scaling) employs tournament selection and fine-tunes an intrinsic reasoning selector to ensure high accuracy. We summarize our contributions as follows:

- *Orchestrated Test-time Scaling.* We introduce an orchestrated test-time scaling framework that converts additional inference compute into accuracy gains by scaling across three distinct perspectives: i) Internal Scaling (RL-enhanced Intrinsic Reasoning), ii) Sequential Scaling (Iterative Refinement), and iii) Parallel Scaling (Diverse Synthesis and Tournament Selection).

- *General Purpose and Scalability.* The entire framework is fully general-purpose, plug-and-play, and effortlessly adaptable to any database. As future LLMs become more powerful and computing continues to get cheaper, Agentar-Scale-SQL's performance ceiling will lift on its own; we can simply allocate more compute to both generation and selection stages to achieve even higher accuracy.

- *Transparent and Actionable Insights.* We propose a structured three-stage Text-to-SQL framework, delineating the specific roles and objectives of each stage. Crucially, our work is the first to scale both the SQL generation and selection stages. Together, these contributions establish transparent and actionable insights to guide future research and development in the Text-to-SQL field.

- *Extensive Experiments.* Extensive experiments confirm the SOTA performance of Agentar-Scale-SQL. Specifically, Agentar-Scale-SQL achieves an EX of 74.90% on the BIRD development set and 81.67% on the test set, along with an R-VES of 77.00%. With these results[1], Agentar-Scale-SQL **ranks first** on the BIRD leaderboard. Our findings indicate that scaling is a critical factor for achieving SOTA performance in Text-to-SQL.

[1] See the official BIRD leaderboard at `https://bird-bench.github.io/`. The rankings are dated November 27, 2025.

2 Related Work

Text-to-SQL. While LLMs have significantly advanced Text-to-SQL, their direct application remains challenging for complex queries (Liu et al., 2024, 2025b). Recent approaches address this via prompt-based methods (Xie et al., 2025; Gao et al., 2024; Dong et al., 2023), fine-tuning (Yao et al., 2025; Pourreza et al., 2025b; Li et al., 2025b, 2024b; Yang et al., 2024), or hybrid strategies that employ test-time scaling (Liu et al., 2025c; Pourreza et al., 2025a; Agrawal and Nguyen, 2025). Most relevant to our work are these scaling methods, which use techniques such as parallel synthesis and sequential refinement to improve generation quality. However, these approaches share a common limitation in that they lack an orchestrated combination of different scaling dimensions.

Test-time Scaling. Test-time Scaling (TTS) enhances an LLM's performance during inference by strategically expanding computation, rather than altering model weights (Zhang et al., 2025a; Snell et al., 2024; Kaplan et al., 2020). Existing research in TTS can be broadly classified into three paradigms. Parallel scaling focuses on generating multiple candidate solutions concurrently and aggregating them to improve the likelihood of finding a correct answer, with self-consistency (Wang et al., 2023) being a prominent example. In contrast, sequential scaling emulates a deliberative "System 2" reasoning process by iteratively building or refining a solution through a series of steps, as exemplified by Chain-of-Thought (CoT) (Wei et al., 2022) and iterative refinement (Madaan et al., 2023). More recently, internal scaling has emerged, where the model is trained to autonomously allocate its own computational budget and determine reasoning depth without external orchestration (DeepSeek-AI et al., 2025; Jaech et al., 2024). Our work, Agentar-Scale-SQL, builds upon these foundational concepts by introducing a novel orchestration framework that synergistically combines the three paradigms, specifically tailored to advance SOTA in the Text-to-SQL domain.

3 Methodology

3.1 Overview

Given a question Q_u, evidence E_u, and a target database D, our framework is divided into three stages, as depicted in Figure 1. An offline preprocessing phase precedes the operation of the online framework.

Offline Preprocessing. Our method involves three offline preprocessing steps before online inference. First, to increase the diversity of generation, we represent the database metadata in two formats (see Figure 2). We create a Markdown-based light schema designed for effective In-Context Learning (ICL) with general-purpose LLMs and use a standard DDL schema to fine-tune the code-specialized model, which capitalizes on its training background to achieve faster convergence. Second, we index all textual cell values from the database into a vector store VD_{cell}. Finally, we also index the training set as examples into a vector store $VD_{example}$, which allows us to retrieve relevant few-shot examples during inference by embedding the skeleton-extracted user question and performing a similarity search.

Online Framework. Agentar-Scale-SQL is an orchestrated test-time scaling framework that converts additional inference compute into accuracy gains by employing an orchestrated scaling strategy across three distinct perspectives: i) Internal Scaling (through RL-enhanced Intrinsic Reasoning), ii) Sequential Scaling (through Iterative Refinement), and iii) Parallel Scaling (through both Diverse Synthesis and Tournament Selection). Overall, Agentar-Scale-SQL consists of three stages: Step 1 (Task Understanding), Step 2 (SQL Generation Scaling), and Step 3 (SQL Selection Scaling).

- **Step 1 (Task Understanding)** focuses on comprehensively understanding the user's intent and retrieving relevant context.

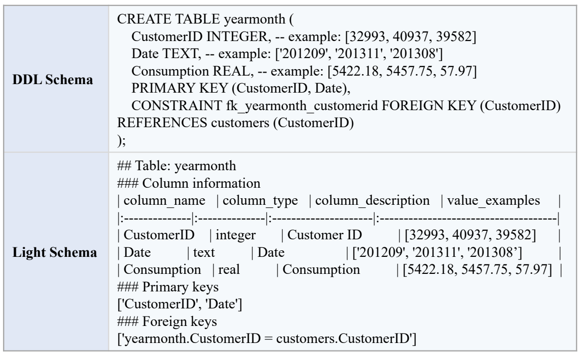
- **Step 2 (SQL Generation Scaling)** develops diverse synthesis and iterative refinement to obtain high-quality and diverse SQL candidates. The diverse synthesis component, in particular, utilizes two distinct generators operating on a specific schema format: a reasoning generator with DDL schema and an ICL generator with light schema.

- **Step 3 (SQL Selection Scaling)** utilizes a tournament selection method, enhanced by an intrinsic reasoning selector, to achieve high selection accuracy.

The following sections delve into the details of each component.

3.2 Task Understanding

Database cells are crucial (Pourreza et al., 2025a; Liu et al., 2025c), as they provide the specific val-



DDL Schema

```
CREATE TABLE yearmonth (
   CustomerID INTEGER, -- example: [32993, 40937, 39582]
   Date TEXT, -- example: ['201209', '201311', '201308']
   Consumption REAL, -- example: [5422.18, 5457.75, 57.97]
   PRIMARY KEY (CustomerID, Date),
   CONSTRAINT fk_yearmonth_customerid FOREIGN KEY (CustomerID)
REFERENCES customers (CustomerID)
);
```

Light Schema

```
## Table: yearmonth
### Column information
| column_name   | column_type   | column_description   | value_examples    |
|:--------------|:--------------|:---------------------|:-------------------------------------|
| CustomerID    | integer       | Customer ID          | [32993, 40937, 39582]     |
| Date          | text          | Date                 | ['201209', '201311', '201308’]       |
| Consumption   | real          | Consumption          | [5422.18, 5457.75, 57.97]  |
### Primary keys
['CustomerID', 'Date']
### Foreign keys
['yearmonth.CustomerID = customers.CustomerID']
```

Figure 2: An example of a database schema represented in both DDL schema and light schema formats.

ues needed for SQL clauses like WHERE and HAVING. Similarly, well-chosen few-shot examples are known to significantly improve the performance of In-Context Learning (ICL) (Gao et al., 2024). Therefore, the primary objective of the Task Understanding step is to identify and retrieve these two critical forms of context: relevant database cells and effective demonstration examples. This is achieved through two parallel sub-processes: i) keyword extraction that extracts keywords from the question Q_u and evidence E_u to retrieve relevant cells using embedding-based similarity from VD_{cell}, ii) skeleton extraction from the Q_u to retrieve relevant examples using embedding-based similarity from $VD_{example}$.

3.3 SQL Generation Scaling

This stage employs two complementary generators, operating on dual schema views, to produce high-quality and diverse candidates. The first generator, $M_{reasoning}$, is an intrinsic reasoner trained via reinforcement learning (RL). The second, M_{ICL}, is an in-context learning (ICL) model driven by a large, proprietary LLM. Then, an iterative refinement loop for syntax repair and semantic edits. As a result, a set of n SQL candidates, denoted as $C = \{c_1, c_2, ..., c_n\}$, is generated. The following sections detail each component.

3.3.1 Intrinsic Reasoning SQL Generator

Inspired by Arctic-Text2SQL-R1 (Yao et al., 2025), we pursue robust intrinsic reasoning Text-to-SQL generation via a simple, execution-grounded RL framework.

Overview of RL Pipeline. We adopt GRPO (Shao et al., 2024) due to its proven efficiency and effectiveness on structured reasoning tasks. Unlike Arctic-Text2SQL-R1, we exclusively use the training data from the BIRD dataset, without any extra data. Formally, let π_θ denote our policy model parameterized by θ. For any given input question Q_u, its associated evidence and schema, the model generates a set of N candidate SQL queries (i.e., rollouts), $\{o_{Q,1}, \ldots, o_{Q,N}\}$. Each candidate is then evaluated to yield an explicit reward signal, as described later. This per-input batch of rollouts allows for computing relative advantages, thereby stabilizing learning and fostering robust policy updates.

The clipped surrogate objective for each sample i is defined as:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \min\left(r_i A_i,\right. \qquad (1)$$
$$\left.\text{clip}(r_i, 1-\epsilon, 1+\epsilon)A_i\right)$$

The full GRPO objective:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}[L(\theta)] - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \qquad (2)$$

where r_i is the likelihood ratio $\frac{\pi_\theta(o_i|Q)}{\pi_{\theta_{\text{old}}}(o_i|Q)}$, A_i is the advantage, and D_{KL} is a KL-divergence penalty that keeps the policy close to a reference (supervised fine-tuned) model (Ouyang et al., 2022). The parameters ϵ and β are tuned in practice to balance exploration and stability.

Reward Design. We define a reward function R_G solely on final execution correctness and basic syntax validity following Arctic-Text2SQL-R1:

$$R_G = \begin{cases} 1, & \text{if results match the ground truth;} \\ 0.1, & \text{if the SQL is executable;} \\ 0, & \text{otherwise.} \end{cases} \qquad (3)$$

3.3.2 Diverse Synthesis

The diverse synthesis strategy is designed to generate a high-quality and varied pool of SQL candidates. This strategy involves two parallel and complementary generators: a fine-tuned reasoning generator and an ICL generator. We create a Markdown-based light schema designed for effective ICL with general-purpose LLMs and use a standard Definition Language (DDL) schema to fine-tune the code-specialized model, which capitalizes on its training background to achieve faster convergence.

Reasoning Generator. This generator utilizes the DDL schema to conduct deep, step-by-step reasoning. On the one hand, guided by our Internal Scaling principle, it is engineered to construct complex queries with the primary goal of achieving high accuracy. On the other hand, it can be fine-tuned to align with the specific characteristics and requirements of the target benchmark.

ICL Generator. In parallel, the ICL Generator utilizes a Markdown-based light schema (see Figure 2) and the few-shot examples retrieved during the Task Understanding stage. To enhance the diversity of the ICL generator, we employ a multi-faceted strategy: varying the input prompts, randomizing the order of in-context examples, utilizing a range of LLMs, and adjusting the temperature settings. Typically, the varied prompts are categorized into three distinct styles: direct prompting (without explicit reasoning), Chain-of-Thought (CoT) prompting (Wei et al., 2022), and problem decomposition. This generator excels at rapidly producing a broad range of plausible queries by leveraging pattern recognition from the provided in-context examples.

By combining the deep, analytical approach of the reasoning generator with the example-driven approach of the ICL generator, we maximize the diversity of the candidate pool. This synergy significantly increases the probability that at least one correct or near-correct query is present before the selection phase.

3.3.3 Iterative Refinement

To further improve the quality of SQL candidates, we introduce the iterative refinement module to refine errors. SQL queries can contain both syntactic and semantic errors (Yang et al., 2025; Xu et al., 2025). We employ a two-pronged approach to refine errors. For syntactic errors (the former), we use the SQL fixer, an LLM-based component that is conditionally activated to repair invalid syntax. For semantic errors (the latter), we employ the SQL revisor, an LLM agent designed to identify and refine logical flaws in the query. To streamline the revision process, we first group queries by their execution outcomes. Subsequently, we randomly select one query from each group for refinement.

3.4 SQL Selection Scaling

The primary limitation of majority voting is its underlying assumption that the most frequent answer is also the correct one, a premise that does not always hold. Instead, we employ a tournament selection process where a reasoning selector, enhanced by reinforcement learning (RL), evaluates candidates through pairwise comparisons. The top-ranked SQL query from this process is selected as the final SQL. We detail these modules in the following sections.

3.4.1 Tournament Selection

We select an optimal SQL query in a two-stage process. First, we consolidate an initial pool of queries by grouping them based on identical execution results on a database D. A single representative is chosen from each group to form a candidate set $C' = \{c_1, \ldots, c_m\}$. Second, these candidates compete in a pairwise round-robin tournament. For each pair (c_i, c_j), a reasoning selector $\mathcal{M}_{selection}$ determines a winner based on the question, light schema, and execution results, incrementing the winner's score W_i. The final query is the one with the maximum score: $c_{final} = \arg\max_{c_i \in C'} W_i$.

3.4.2 Intrinsic Reasoning SQL Selector

We apply Reinforcement Learning (RL) to SQL selector $\mathcal{M}_{selection}$, an approach analogous to the intrinsic reasoning used in the SQL generator.

Overview of RL Pipeline. Following the methodology described in Section 3.3.1, we apply GRPO to enhance the reasoning capabilities for SQL selection. Based on the training set, we construct 8.5k samples for reinforcement learning.

Reward Design. The objective of SQL selection is to identify the correct query from a set of candidates. To achieve this, we introduce a result-oriented reward function, R_S, designed to evaluate the correctness of the selection:

$$R_S = \begin{cases} 1, & \text{if the selection is correct;} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

4 Experiments

In this section, we experimentally evaluate the proposed Agentar-Scale-SQL on the large-scale dataset. We aim to answer the following research questions:

- **RQ1:** How does Agentar-Scale-SQL perform compared with the state-of-the-art methods?
- **RQ2:** How does each module affect the overall performance of Agentar-Scale-SQL?
- **RQ3:** What are the individual and complementary roles of the ICL and Reasoning generators?
- **RQ4:** How does performance scale with the number of candidates across different complexity levels?

4.1 Experimental Setup

Datasets. We evaluate our method on the BIRD benchmark (Li et al., 2023), a particularly challenging cross-domain dataset. It comprises over 12,751 question-SQL pairs across 95 large databases, simulating real-world complexity with messy data and intricate schemas across more than 37 professional domains.

Baselines. We compared several top-ranking baseline methods from the overall leaderboard and the single-model leaderboard. The former consists of fifteen baselines, including AskData + GPT-4o (Shkapenyuk et al., 2025), LongData-SQL, CHASE-SQL + Gemini (Pourreza et al., 2025a), JoyDataAgent-SQL, XiYan-SQL (Liu et al., 2025c), among others. The latter comprises eight leading methods, such as Gemini-SQL, Databricks RLVR 32B, Sophon-Text2SQL-32B, Arctic-Text2SQL-R1-32B (Yao et al., 2025).

Metrics. Following prior work (Pourreza et al., 2025a), we use Execution Accuracy (EX), the official metric for the respective leaderboard, as the primary evaluation metric to compare methods. Besides, we adopt the official Reward-based Valid Efficiency Score (R-VES) to evaluate the efficiency of the generated SQL.

Implementation Details. We implement Agentar-Scale-SQL with LangChain[2]/LangGraph[3] and chroma[4] retrieval using all-MiniLM-L6-v2[5] embeddings. The Task Understanding is powered by Gemini-2.5-Flash (Comanici et al., 2025) (temperature 0.2). The ICL SQL Generator utilizes Gemini-2.5-Pro (Comanici et al., 2025) (abbreviated as pro) with two temperature settings (0.5 and 1.8) and GPT-5 (OpenAI, 2025) (minimal reasoning effort). The Reasoning SQL Generator is fine-tuned based on Omni-SQL-32B (Li et al., 2025b). By default, candidates comprise 9 from the ICL SQL Generator and 8 from the Reasoning SQL Generator. The SQL Fixer and SQL Reviser both use pro. The base model of the Reasoning SQL Selector is Qwen2.5-Coder-32B-Instruct (Hui et al., 2024). The RL framework leverages verl (Sheng et al., 2025) on 32 NVIDIA A100 80GB GPUs.

4.2 Main Results (RQ1)

As presented in Table 1, our Agentar-Scale-SQL framework establishes a new SOTA on the BIRD benchmark, achieving 81.67% execution accuracy (EX) and 77.00% R-VES on the test set. This result surpasses the prior SOTA (AskData + GPT-4o) by 0.79% in test EX. It is worth noting that these approaches employ powerful models, such as Gemini 2.5 Pro and Claude 4.

Notably, the performance gain is even more pronounced when compared to single-trained models; Agentar-Scale-SQL outperforms the strongest single model (Gemini-SQL) by a substantial 5.54% in test EX. These results empirically validate the effectiveness of our orchestrated scaling strategy.

4.3 Ablation Study (RQ2)

Next, we study the effectiveness of each module in Agentar-Scale-SQL by comparing Agentar-Scale-SQL with its variants without the key module. The results are listed in Table 2.

Agentar-Scale-SQL w/o Task Understanding. In this variant, the overall EX improves by 0.45, showing that the task-understanding module resolves ambiguities—both in the values required for SQL clauses and in the phrasing of the questions themselves.

Agentar-Scale-SQL w/o Reasoning SQL Generator. Removing the reasoning SQL generator leads to the most substantial performance drop, with total accuracy decreasing by 4.89 points. This result strongly demonstrates the effectiveness of the intrinsic scaling approach. It is essential for generating accurate SQL logic and aligning with the target data's preferences, which is an indispensable asset in solving complex problems.

Agentar-Scale-SQL w/o ICL SQL Generator. When the ICL SQL generator is excluded, the total accuracy falls by 3.78 points, the second-largest drop observed. Notably, the performance on challenging questions plummets from 64.14 to 55.86. This highlights the complementary nature of our two generators. The ICL generator excels at leveraging contextual examples to construct complex queries, providing an effective alternative pathway to a correct solution. The parallel scaling using two complementary generators ensures a diverse and high-quality pool of candidate SQL queries, which is crucial for achieving high performance.

Agentar-Scale-SQL w/o Iterative Refinement. We also analyzed the contribution of the iterative

[2] https://github.com/langchain-ai/langchain

[3] https://github.com/langchain-ai/langgraph

[4] https://github.com/chroma-core/chroma

[5] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Table 1: Evaluation results on the development and test sets.

Methods	EX (Dev)	EX (Test)	R-VES (%)
Overall			
Alpha-SQL (Li et al., 2025a)	69.70	70.26	-
OmniSQL-32B (Li et al., 2025b)	69.23	72.05	67.05
OpenSearch-SQL (Xie et al., 2025)	69.30	72.28	69.36
Reasoning-SQL 14B (Pourreza et al., 2025b)	72.29	72.78	68.67
ExSL + granite-34B-code	72.43	73.17	71.37
CSC-SQL (Sheng and Xu, 2025)	71.33	73.67	67.84
CYAN-SQL	73.47	75.35	-
XiYan-SQL (Liu et al., 2025c)	73.34	75.63	71.41
Contextual-SQL (Agrawal and Nguyen, 2025)	73.50	75.63	70.02
TCDataAgent-SQL	74.12	75.74	-
JoyDataAgent-SQL	74.25	75.85	70.16
CHASE-SQL + Gemini (Pourreza et al., 2025a)	74.90	76.02	69.94
LongData-SQL	74.32	77.53	71.89
AskData + GPT-4o (Shkapenyuk et al., 2025)	76.14	80.88	76.24
Single Trained Model			
SHARE (Qu et al., 2025)	64.14	-	-
Syn CoT + DPO (Liu et al., 2025a)	67.10	-	-
XiYanSQL-QwenCoder-32B (Liu et al., 2025c)	67.01	69.03	-
Jiayin-Pangu-Text2SQL-14B	71.10	73.45	-
Arctic-Text2SQL-R1-32B (Yao et al., 2025)	72.20	73.84	-
Sophon-Text2SQL-32B	72.43	74.79	-
Databricks RLVR 32B (Ali et al., 2025)	-	75.68	-
Gemini-SQL (Multitask SFT + Gemini-2.5-Pro)	72.62	76.13	-
Ours			
Agentar-Scale-SQL (Ours)	**74.90**	**81.67**	**77.00**

refinement module via an ablation study. Its removal caused a 0.52-point drop in performance, which we attribute to the module's ability to polish the SQL and correct syntactic and semantic errors. **Agentar-Scale-SQL w/o SQL Selection Scaling.** Agentar-Scale-SQL w/o SQL selection scaling denotes that we employ self-consistency (i.e., majority voting based on execution results) to select the best SQL. We can observe that our method outperforms the self-consistency baseline by 1.82 points in EX. This is because the most frequently executed result is not necessarily correct. Our selection strategy, which likely incorporates more signals than simple frequency, proves to be a more effective and robust method for identifying the correct query.

4.4 Analysis of Generator Components (RQ3)

As shown in Figure 3, the ICL generator achieves a notably higher upper bound accuracy (81.36%) than the reasoning generator (75.88%), indicating its strong potential for generating correct queries. However, combining their outputs (All) achieves the highest overall upper bound of 84.29%. This synergistic gain is explained by their complementary nature, as illustrated in Figure 4. While they share a large set of correct solutions, they also uniquely solve 47 and 12 samples, respectively. This expanded coverage holds across all difficulty levels (Figure 5). A breakdown by difficulty reveals that the reasoning generator holds an edge on simple and moderate tasks, while the ICL generator proves more effective in challenging problems.

Ultimately, this richer and more diverse candidate pool allows our final selection strategy to achieve its peak accuracy of 74.90%, demonstrating the crucial role of the dual-generator approach.

Table 2: Ablation results on the development set.

Methods	Simple	Moderate	Challenging	Total	ΔTotal
Agentar-Scale-SQL	79.35	69.40	64.14	74.90	-
w/o Task Understanding	79.14	68.32	64.14	74.45	-0.45
w/o Reasoning SQL Generator	74.92	63.79	58.62	70.01	-4.89
w/o ICL SQL Generator	75.89	66.38	55.86	71.12	-3.78
w/o Iterative Refinement	78.92	68.75	63.45	74.38	-0.52
w/o SQL Selection Scaling	77.95	66.59	62.76	73.08	-1.82

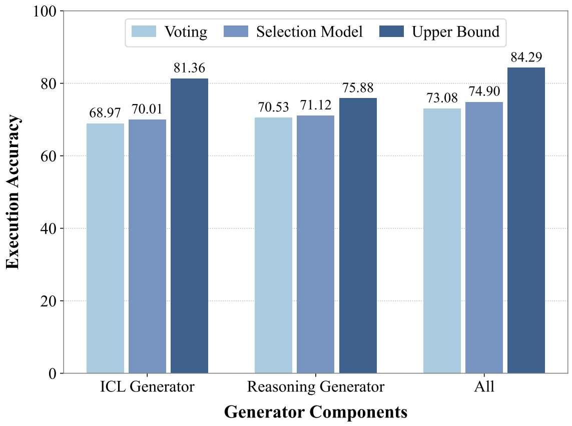


Figure 3: Execution accuracy of voting, selection model, and upper bound across generator components.

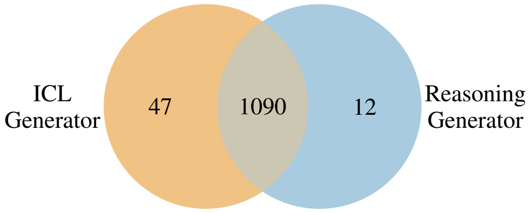


Figure 4: Shared and unique correct samples between ICL and reasoning generators.

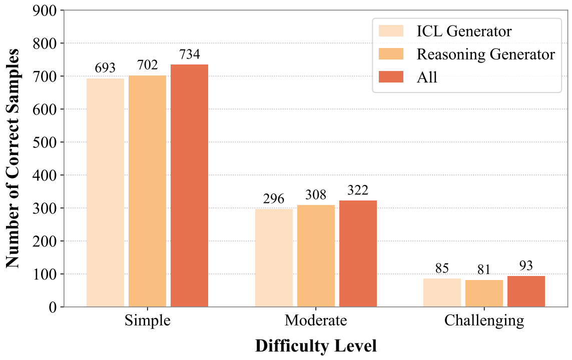


Figure 5: Number of correct samples by difficulty level for ICL, reasoning, and combined generators.

4.5 Impact of the Candidate Number (RQ4)

Finally, we investigate the impact of the candidate number by varying it from 1 to 16. The results are depicted in Figure 6, showing that increasing the number of candidates consistently boosts the Pass@k rate across all difficulty levels. The improvement is most significant for challenging queries and is most substantial when increasing the candidates up to 8, after which the gains diminish. This validates the effectiveness of our parallel scaling strategy.

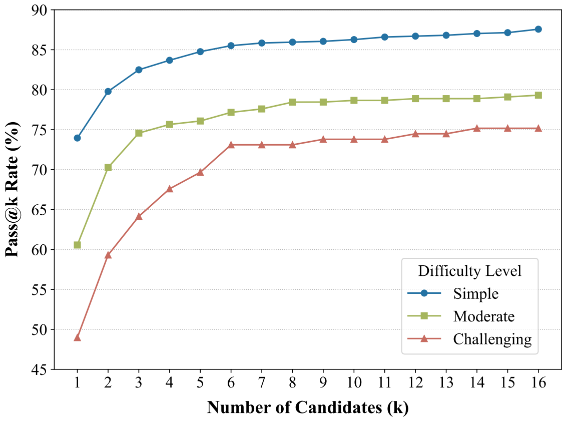


Figure 6: Pass@k rate with varying candidate numbers.

5 Conclusion

We introduced Agentar-Scale-SQL, a novel framework that significantly improves Text-to-SQL performance by synergistically combining internal, sequential, and parallel scaling strategies. Our method achieves SOTA results on the challenging BIRD benchmark, demonstrating an effective path toward human-level accuracy. We release codes and models to support future research in this area.

Building on the success of enhancing intelligence through Test-Time Scaling, we are pioneering our next endeavor: Exercise-Time Scaling. We will empower a new generation of agents to learn through action and evolve from experience.

Limitations

Despite its effectiveness, Agentar-Scale-SQL's reliance on orchestrated test-time scaling introduces several key limitations. The framework's primary drawback is its substantial computational overhead and high latency due to the multiple LLM calls for generation, refinement, and selection, making it less suitable for real-time applications. In our commercial practice (developing B2B ChatBI products), we observe that enterprise clients prioritize Accuracy above all else. In decision-making scenarios, a hallucinatory SQL query is unacceptable, whereas a latency of several seconds is often tolerable for complex analytics generation. Agentar-Scale-SQL is designed specifically to bridge the "last mile" of accuracy for these production requirements. Furthermore, its performance is fundamentally bounded by the capabilities of the underlying base LLM, and it is susceptible to cascading errors where a failure in an early stage, such as task understanding, can compromise the entire process.

References

Sheshansh Agrawal and Thien Nguyen. 2025. Opensourcing the best local text-to-sql system.

Alnur Ali, Ashutosh Baheti, Jonathan Chang, Ta-Chung Chi, Brandon Cui, Andrew Drozdov, Jonathan Frankle, Abhay Gupta, Pallavi Koppol, Sean Kulinski, Jonathan Li, Dipendra Misra, Krista Opsahl-Ong, Jose Javier Gonzalez Ortiz, Matei Zaharia, and Yue Zhang. 2025. A state-of-the-art sql reasoning model using rlvr. *Preprint*, arXiv:2509.21459.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit S. Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 81 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: zero-shot text-to-sql with chatgpt. *CoRR*, abs/2307.07306.

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.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-coder technical report. *CoRR*, abs/2409.12186.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, and 80 others. 2024. Openai o1 system card. *CoRR*, abs/2412.16720.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

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 & benchmark]. *Proc. VLDB Endow.*, 17(11):3318–3331.

Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025a. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. *CoRR*, abs/2502.17248.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025b. Omnisql: Synthesizing high-quality text-to-sql data at scale. *CoRR*, abs/2503.02240.

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, 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. 2023. 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*.

Hanbing Liu, Haoyang Li, Xiaokang Zhang, Ruotong Chen, Haiyong Xu, Tian Tian, Qi Qi, and Jing Zhang.

2025a. Uncovering the impact of chain-of-thought reasoning for direct preference optimization: Lessons from text-to-sql. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 21223–21261. Association for Computational Linguistics.

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? *CoRR*, abs/2408.05109.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025b. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Trans. Knowl. Data Eng.*, 37(10):5735–5754.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. 2025c. Xiyan-sql: A novel multi-generator framework for text-to-sql. *CoRR*, abs/2507.04701.

Yuyu Luo, Guoliang Li, Ju Fan, Chengliang Chai, and Nan Tang. 2025. Natural language to SQL: state of the art and open problems. *Proc. VLDB Endow.*, 18(12):5466–5471.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. 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*.

OpenAI. 2025. Introducing gpt-5. Blog post.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. 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*.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Ö. Arik. 2025a. CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-sql. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan Ö. Arik. 2025b. Reasoning-sql: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-sql. *CoRR*, abs/2503.23157.

Ge Qu, Jinyang Li, Bowen Qin, Xiaolong Li, Nan Huo, Chenhao Ma, and Reynold Cheng. 2025. SHARE: an slm-based hierarchical action correction assistant for text-to-sql. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 11268–11292. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM.

Lei Sheng and Shuai-Shuai Xu. 2025. CSC-SQL: corrective self-consistency in text-to-sql via reinforcement learning. *CoRR*, abs/2505.13271.

Vladislav Shkapenyuk, Divesh Srivastava, Theodore Johnson, and Parisa Ghane. 2025. Automatic metadata extraction for text-to-sql. *CoRR*, abs/2505.19988.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314.

Richard S. Sutton. 2019. The bitter lesson. Blog post. Accessed: 2024-08-28.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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*.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. *Proc. ACM Manag. Data*, 3(3):194:1–194:24.

Bo Xu, Shufei Li, Hongyu Jing, Ming Du, Hui Song, Hongya Wang, and Yanghua Xiao. 2025. Boosting text-to-sql through multi-grained error identification. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 4282–4292. Association for Computational Linguistics.

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.

Yicun Yang, Zhaoguo Wang, Yu Xia, Zhuoran Wei, Haoran Ding, Ruzica Piskac, Haibo Chen, and Jinyang Li. 2025. Automated validating and fixing of text-to-sql translation with execution consistency. *Proc. ACM Manag. Data*, 3(3):134:1–134:28.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2025. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *CoRR*, abs/2505.20315.

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.

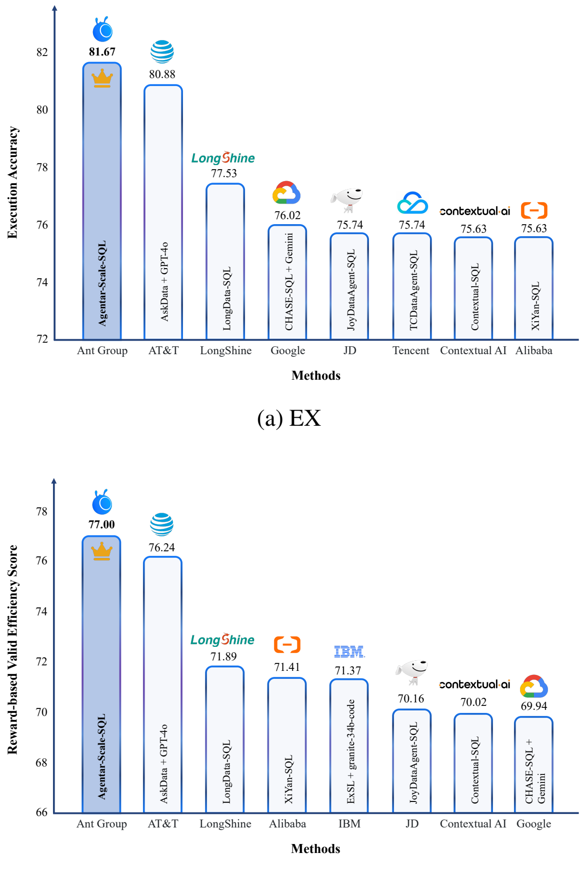
Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *CoRR*, abs/2403.02951.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. 2025a. What, how, where, and how well? A survey on test-time scaling in large language models. *CoRR*, abs/2503.24235.

Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. 2025b. Reward-sql: Boosting text-to-sql via stepwise reasoning and process-supervised rewards. *CoRR*, abs/2505.04671.

A BIRD Leaderboard: Bar Chart and Snapshot

We present the BIRD rankings as a bar chart (Figure 7) and a snapshot (Figure 8).



(a) EX

(b) R-VES

Figure 7: BIRD leaderboard (as of September 28, 2025): EX and R-VES performance comparison.

B Additional Results on the BIRD Test Set

Table 3: Additional results on the BIRD test set.

Metric	Simple	Moderate	Challenging	Total
Count	949	555	285	1789
EX	86.83	78.20	71.23	81.67
Soft F1	87.28	79.41	73.58	82.65
R-VES	81.90	73.23	68.03	77.00

C Definition of Evaluation Metrics

This section provides detailed definitions for the three primary metrics (Li et al., 2023) used to evaluate the performance of our text-to-SQL framework: Execution Accuracy (EX), Reward-based Valid Efficiency Score (R-VES), and the Soft F1-Score.

C.1 Execution Accuracy (EX)

Execution Accuracy (EX) is a strict binary metric that assesses whether the SQL query generated by the model produces the exact same result set as the ground-truth SQL query.

For each question, the predicted SQL is executed against the database. The resulting table is then compared to the table generated by executing the official ground-truth SQL. A prediction is considered correct (score of 1) only if the two result sets are identical. Any deviation, including differences in row or column order, mismatched values, or SQL execution errors, results in a score of 0.

The final EX score for the entire evaluation set is the average of these binary scores, calculated as:

$$\text{EX} = \frac{1}{N} \sum_{i=1}^{N} \text{score}_i \tag{5}$$

where N is the total number of questions in the evaluation set, and score_i is 1 if the predicted query for question i is correct, and 0 otherwise.

C.2 Reward-based Valid Efficiency Score (R-VES)

The Reward-based Valid Efficiency Score (R-VES) is designed to measure the execution efficiency of a correctly generated SQL query relative to its ground-truth counterpart. This metric is calculated only for queries that pass the Execution Accuracy (EX) evaluation.

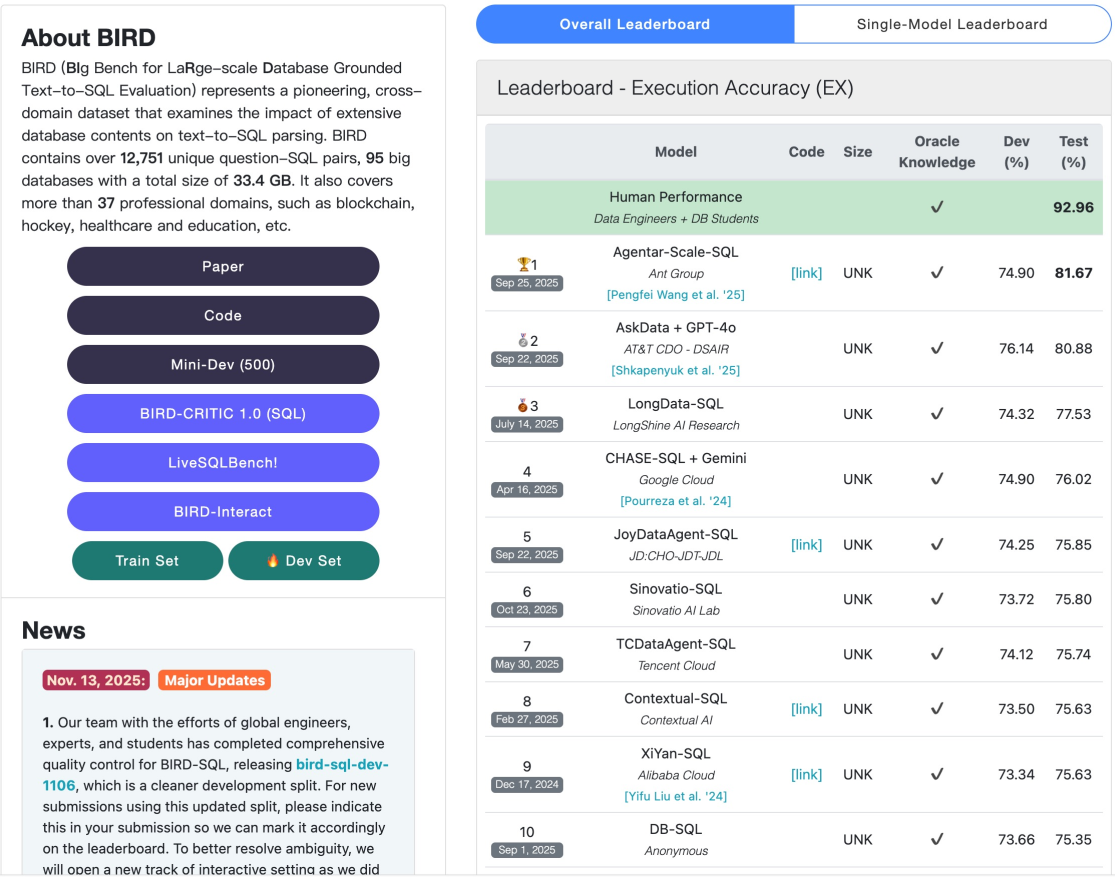
The score is based on a reward function that considers the ratio of execution times between the predicted query (T_{pred}) and the ground-truth query (T_{gold}). The reward for a single valid query is defined as:

$$\text{Reward}_i = \begin{cases} 1 & \text{if } T_{\text{pred}_i} < T_{\text{gold}_i} \\ \frac{2}{1+(T_{\text{pred}_i}/T_{\text{gold}_i})} & \text{if } T_{\text{pred}_i} \geq T_{\text{gold}_i} \end{cases} \tag{6}$$

This function assigns a full reward of 1 if the predicted query is more efficient than the ground truth. If it is slower, the reward decreases as the execution time ratio increases, penalizing inefficient queries.

The final R-VES is the average reward over all validly executed queries:

$$\text{R-VES} = \frac{1}{N_{\text{valid}}} \sum_{i=1}^{N_{\text{valid}}} \text{Reward}_i \tag{7}$$



About BIRD

BIRD (**BI**g Bench for La**R**ge-scale **D**atabase Grounded Text-to-SQL Evaluation) represents a pioneering, cross-domain dataset that examines the impact of extensive database contents on text-to-SQL parsing. BIRD contains over **12,751** unique question-SQL pairs, **95** big databases with a total size of **33.4 GB**. It also covers more than **37** professional domains, such as blockchain, hockey, healthcare and education, etc.

News

Nov. 13, 2025: **Major Updates**

1. Our team with the efforts of global engineers, experts, and students has completed comprehensive quality control for BIRD-SQL, releasing **bird-sql-dev-1106**, which is a cleaner development split. For new submissions using this updated split, please indicate this in your submission so we can mark it accordingly on the leaderboard. To better resolve ambiguity, we will open a new track of interactive setting as we did

Leaderboard - Execution Accuracy (EX)

	Model	Code	Size	Oracle Knowledge	Dev (%)	Test (%)
	Human Performance *Data Engineers + DB Students*			✓		**92.96**
🏆1 Sep 25, 2025	Agentar-Scale-SQL *Ant Group* [Pengfei Wang et al. '25]	[link]	UNK	✓	74.90	**81.67**
🥈2 Sep 22, 2025	AskData + GPT-4o *AT&T CDO - DSAIR* [Shkapenyuk et al. '25]		UNK	✓	76.14	80.88
🥉3 July 14, 2025	LongData-SQL *LongShine AI Research*		UNK	✓	74.32	77.53
4 Apr 16, 2025	CHASE-SQL + Gemini *Google Cloud* [Pourreza et al. '24]		UNK	✓	74.90	76.02
5 Sep 22, 2025	JoyDataAgent-SQL *JD:CHO-JDT-JDL*	[link]	UNK	✓	74.25	75.85
6 Oct 23, 2025	Sinovatio-SQL *Sinovatio AI Lab*		UNK	✓	73.72	75.80
7 May 30, 2025	TCDataAgent-SQL *Tencent Cloud*		UNK	✓	74.12	75.74
8 Feb 27, 2025	Contextual-SQL *Contextual AI*	[link]	UNK	✓	73.50	75.63
9 Dec 17, 2024	XiYan-SQL *Alibaba Cloud* [Yifu Liu et al. '24]	[link]	UNK	✓	73.34	75.63
10 Sep 1, 2025	DB-SQL *Anonymous*		UNK	✓	73.66	75.35

Figure 8: A snapshot of the BIRD benchmark’s dynamic leaderboard as of November 27, 2025.

where N_{valid} is the total number of queries that passed the EX evaluation. To ensure stability, execution times are measured with a timeout, and the evaluation is repeated multiple times, with the highest score being reported.

C.3 Soft F1-Score

The Soft F1-Score offers a more lenient evaluation than EX by comparing the content similarity of the result tables produced by the predicted and ground-truth queries. This metric is insensitive to column order and robust to missing values.

The calculation proceeds by comparing the tables on a row-by-row, cell-by-cell basis. For each row in the predicted table and the ground-truth table, we compute the following three quantities:

- **Matched** ($tp_{\textbf{row}}$): The number of common cell values between a predicted row and its best-matching ground-truth row.
- **Pred_only** ($fp_{\textbf{row}}$): The number of cell values present in the predicted row but not in its matched ground-truth row.
- **Gold_only** ($fn_{\textbf{row}}$): The number of cell values present in the ground-truth row but not in its matched predicted row.

These row-level counts are then aggregated across all rows to compute the total True Positives (tp), False Positives (fp), and False Negatives (fn) for the entire table:

$$\text{tp} = \sum_{\text{all rows}} tp_{\text{row}}$$
$$\text{fp} = \sum_{\text{all rows}} fp_{\text{row}}$$
$$\text{fn} = \sum_{\text{all rows}} fn_{\text{row}}$$

Finally, standard Precision, Recall, and F1-Score are calculated using these aggregated values:

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} \tag{8}$$

$$\text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}} \tag{9}$$

$$\text{Soft F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

D Discussion of Schema Linking

Given the universal and pluggable architecture of our Agentar-Scale-SQL framework, we did not develop a built-in schema linking strategy for the BIRD benchmark. The framework is explicitly designed to be modular, facilitating the seamless integration of components like a schema linker. This approach is particularly advantageous for large-scale databases with numerous tables, as a dedicated schema linking module can be easily incorporated as needed, without altering the core system.

E Extended Analysis: Scaling, Efficiency, and Comparisons

In this section, we provide further elaboration on the core contributions of our method regarding test-time scaling orchestration, a detailed comparison of scaling capabilities against SOTA baselines, and a quantitative analysis of computational costs versus utility.

E.1 Orchestrated Test-Time Scaling

Our work distinguishes itself from naive combinations of existing modules through a systematic orchestration of Test-Time Scaling across three distinct dimensions. Unlike simple stacking, we introduce specific designs to maximize synergy, such as *Diverse Synthesis* (combining breadth via ICL with depth via RL-tuned reasoning) and *Tournament Selection* (an RL-enhanced pairwise judge). This orchestration allows Agentar-Scale-SQL to validate the "Scaling Hypothesis" in Text-to-SQL, outperforming the strongest single model (Gemini-SQL) by 5.54% and the previous SOTA (AskData + GPT-4o) by 0.79%.

E.2 Comparison of Scaling Capabilities

Table 4 categorizes existing SOTA methods based on their supported scaling dimensions. While most frameworks support only one or two dimensions, our method is the first to fully integrate all three, offering a comprehensive optimization space.

E.3 Computational Cost vs. Practical Utility

While test-time scaling inherently introduces higher latency compared to single-pass models, this is a deliberate trade-off designed for high-stakes enterprise scenarios. In B2B ChatBI products, accuracy is paramount; a hallucinated query in a financial report is unacceptable, whereas a latency of 10–30 seconds is often tolerable for complex analytics.

Furthermore, our framework offers high configurability. Users can dynamically adjust the number of candidates (N) or skip the refinement stage to

Table 4: Comparison of scaling capabilities across SOTA methods. Our method uniquely supports Internal, Sequential, and Parallel scaling simultaneously.

Methods	**Internal Scaling**	**Sequential Scaling**	**Parallel Scaling**
Overall			
XiYan-SQL	-	✓	✓
Contextual-SQL	-	-	✓
CHASE-SQL + Gemini	-	✓	✓
AskData + GPT-4o	-	-	✓
Single Trained Models			
Arctic-Text2SQL-R1-32B	✓	-	-
Databricks RLVR 32B	✓	-	-
Gemini-SQL	✓	-	-
Agentar-Scale-SQL (Ours)	✓	✓	✓

Table 5: Comparison of inference costs (estimated number of LLM calls).

Methods	**Task Understanding**	**Generation**	**Refinement**	**Selection**	**Total Calls**
Overall					
Contextual-SQL	-	1024	-	1024	2048
CHASE-SQL + Gemini	32	16	32	10	297
Single Trained Models					
Sophon-Text2SQL-32B	-	8 - 32	-	-	8 - 32
Databricks RLVR 32B	-	1 - 7	-	-	1 - 7
Gemini-SQL	-	1 - 7	-	-	1 - 7
Agentar-Scale-SQL (Ours)	1	17	1 - 51	1 - 561	20 - 630

reduce costs for simpler queries, bridging the "last mile" of accuracy only when necessary.

Table 5 provides an approximate comparison of inference costs using the number of LLM calls as a metric. Although our method involves more steps than single-pass models, it significantly optimizes the total calls compared to other multi-stage frameworks (e.g., Contextual-SQL) while achieving SOTA performance.

F Prompts

We have listed a selection of prompts. For the complete list, please refer to the code repository.

- The prompt for the Task Understanding.
- The prompt for the Reasoning Generator.
- The prompt for the ICL Generator with CoT prompting.
- The prompt for the SQL fixer.
- The prompt for the Reasoning Selector.

G Further Work

Agentar-Scale-SQL marks a significant milestone on our journey toward Artificial General Intelligence (AGI) and Artificial Superintelligence (ASI). By leveraging *Orchestrated Test-Time Scaling*, we have substantially advanced the state-of-the-art in Text-to-SQL. Looking ahead, we plan to explore the following directions:

- **Self-Exploration:** We will enable the agent to autonomously explore and accumulate experience in an offline phase, thereby shifting the computational burden from Test-Time to a pre-computation phase we term Exercise-Time.

- **Agentic SQL:** We aim to evolve our current workflow-based approach into a fully autonomous agent, moving beyond predefined structures.
- **Generalization:** We intend to extend the *Orchestrated Test-Time Scaling* methodology to a broader range of code generation and reasoning tasks.

~~~~
# Task Description
You are a text-to-SQL expert who is excellent in analysing text-to-SQL question. You
 are given ['Database Schema', 'Question', 'Evidence'], you need to **
comprehensively analyse the question** by:
1. Identifying the database literals appeared in the question or the evidence.
Database literal refers to a specific value that belongs to a column (e.g., ”Japan”,
 ”Chinese Grand Prix”). These values are used in the WHERE clause to conduct
effective filtering.
2. Generating a question skeleton. The question skeleton contains the question
structure while ignoring the detailed database information (entity names, column
names).

# Guideline for identifying database literals
1. Taking both the question and the evidence into account.

# Guideline for generating a question skeleton
1. Replace **all the database literals** in the question with the database column it
 belongs to.
2. Replace **all the database columns** with the placeholder <COLUMN>.
3. Keep SQL keywords such as ”average”, ”total”, ”difference”, ”count” in the
question skeleton.

# Example of the question mapping to skeleton
Question: Name movie titles released in year 1945. Sort the listing by the
descending order of movie popularity.
Skeleton: Name <COLUMN> released in <YEAR>. Sort the listing by the descending order
 of <COLUMN>.

Question: In August of 1996, how many orders were placed by the customer with the
highest amount of orders?
Skeleton: In <MONTH> of <YEAR>, how many <COLUMN> were placed by the <COLUMN> with
the the highest amount of <COLUMN>?

Question: Calculate the total production for each product which were supplied from
Japan.
Skeleton: Calculate the total <COLUMN> for each <COLUMN> which were supplied from <
COUNTRY>.

Question: Calculate the difference in the average number of low-priority orders
shipped by truck in each month of 1995 and 1996.
Skeleton: Calculate the difference in the average number of <COLUMN> in each month
of <YEAR> and <YEAR>.

# About Output
**Do not return explanations or reasoning process.**
The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {”properties”: {”foo”: {”title”: ”Foo”, ”description”:
 ”a list of strings”, ”type”: ”array”, ”items”: {”type”: ”string”}}}, ”required”: [”
foo”]}
the object {”foo”: [”bar”, ”baz”]} is a well-formatted instance of the schema. The
object {”properties”: {”foo”: [”bar”, ”baz”]}} is not well-formatted.

Here is the output schema:
```
{”properties”: {”database_literals”: {”description”: ”The database literals
extracted from the question and evidence.”, ”items”: {”type”: ”string”}, ”title”: ”
Database Literals”, ”type”: ”array”}, ”question_skeleton”: {”description”: ”the
generated question skeleton”, ”title”: ”Question Skeleton”, ”type”: ”string”}}, ”
required”: [”database_literals”, ”question_skeleton”]}
```
~~~~

Figure 9: The system prompt for the Task Understanding.

```
# Database Schema
{Database Schema}

# Question
{Question}

# Evidence
{Evidence}

Begin! Take a deep breath and think logically.
[no prose][output json Only]
```

Figure 10: The user prompt for the Task Understanding.

~~~
Task Overview:
You are a data science expert. Below, you are provided with a database schema and a
natural language question. Your task is to understand the schema and generate a
valid SQL query to answer the question.

Database Engine:
{dialect}

Database Schema:
{Database Schema}
This schema describes the database's structure, including tables, columns, primary
keys, foreign keys, and any relevant relationships or constraints.
Matched contents:
{matched_contents}
Matched contents presents values related to the question, together with their source
 table and column, for your reference in SQL generation.
Question:
{evidence}
{question}

Instructions:
- If Matched contents is provided, you can use it as reference when generating the
SQL query.
- Make sure you only output the information that is asked in the question. If the
question asks for a specific column, make sure to only include that column in the
SELECT clause, nothing more.
- The generated query should return all of the information asked in the question
without any missing or extra information.
- Before generating the final SQL query, please think through the steps of how to
write the query.

Output Format:
In your answer, please enclose the generated SQL query in a code block:
```sql
-- Your SQL query
```

Take a deep breath and think step by step to find the correct SQL query.
~~~

Figure 11: The prompt for the Reasoning Generator.

```
1. **SELECT Clause:**
    - Only select columns mentioned in the user's question.
    - Avoid unnecessary columns or values.
2. **Aggregation (MAX/MIN):**
    - Always perform JOINs before using MAX() or MIN().
3. **ORDER BY with Distinct Values:**
    - Use `GROUP BY <column>` before `ORDER BY <column> ASC|DESC` to ensure distinct
 values.
4. **Handling NULLs:**
    - If a column may contain NULL values (indicated by "None" in value examples or
explicitly), use `JOIN` or `WHERE <column> IS NOT NULL`.
    - When a field is sorted in ascending order, also apply a NOT NULL filter to it.
    - When using the MIN() function on a column, also include a WHERE clause to
filter NULL values from that column.
5. **FROM/JOIN Clauses:**
    - Only include tables essential to answer the question.
6. **Strictly Follow evidences:**
    - Adhere to all provided evidences.
7. **Thorough Question Analysis:**
    - Address all conditions mentioned in the question.
8. **DISTINCT Keyword:**
    - Use `SELECT DISTINCT` when the question requires unique values (e.g., IDs,
URLs).
    - Refer to column statistics ("Value Statics") to determine if `DISTINCT` is
necessary.
9. **Column Selection:**
    - Carefully analyze column descriptions and evidences to choose the correct
column when similar columns exist across tables.
10. **String Concatenation:**
    - Never use `|| ' ' ||` or any other method to concatenate strings in the `
SELECT` clause.
11. **JOIN Preference:**
    - Prioritize `INNER JOIN` over nested `SELECT` statements.
12. **SQLite Functions Only:**
    - Use only functions available in SQLite.
13. **Date Processing:**
    - Utilize `STRFTIME()` for date manipulation (e.g., `STRFTIME('%Y', SOMETIME)`
to extract the year).
14. **Formatting:**
    - Pay close attention to any formatting requirements in the question, such as
specific decimal places or percentage representation. These are not just suggestions
; they are critical parts of the final answer and must be implemented using
appropriate SQL functions (e.g., ROUND() and multiplying by 100).
    - Use `ROUND()` to round the result to a specific number of decimal places.
    - Use `* 100` to convert a fraction to a percent (%).
```

Figure 12: The database admin instruction.

~~~
# Your Role
You are an experienced database expert. As a SQL expert, formulate a precise SQL
query to resolve the user's question. You must carefully analyze the database schema
, the question's intent, and any provided examples or experience to ensure
correctness and efficiency.
The database stores millions of cell values, you need to write a SQL query that can
be executed in **less than 20.0 seconds**. Otherwise the business will be effected
and lose millions of dollars.

# Database Engine
{dialect}

# Database Admin Instruction
{Database Admin Instruction}

# Input Information
You are provided with ['Examples', 'Database Schema', 'Question', 'Evidence'].
**Examples** are the similar question-SQL pairs which are annotated by other
experienced database experts.
**Database schema** are the database structure. It contains table names, column
names, column types, primary key and foreign key relationship. It also contains some
 relevant database cell values to the question.
**Question** is your task to answer.
**Evidence** contains expert experience about the question. It is **very important
reference** to answer the question.

# About Output
Please return your analysis and then write the SQL query in a code block:
1. Decompose the question and evidence, then map the question and evidence to the
schema.
2. [Important] If the SQL query does not require any subqueries, write the complete
SQL query directly.
3. [Important] If the SQL query does require subqueries, **first write the subquery
(or subqueries), then write the main (outer) query that uses them**.
4. You need to enclose the final SQL query in a code block using the following
format:
```sql

```

## Here is an example about the process of analysing question and generating SQL
query
### Question
How many users have rated the most popular movie?
### Evidence

### Answer
Step 1: Decompose the question.
- We are asked to count the number of users who have rated the most popular movie.
- This requires us to first identify the most popular movie (a subquery is needed),
then count the ratings for that movie.

Step 2: Write the subquery to find the most popular movie.
```sql
SELECT movie_id
FROM movies
ORDER BY movie_popularity DESC
LIMIT 1
```

Step 3: Write the main query to count the ratings for that movie.
```sql
SELECT COUNT(rating_id)
FROM ratings
WHERE movie_id = (
    SELECT movie_id
    FROM movies
    ORDER BY movie_popularity DESC
    LIMIT 1
)
```
~~~

Figure 13: The system prompt for the ICL Generator with CoT prompting.

````
# Examples
{Examples}

# Database Schema
{Database Schema}

# Question
{Question}

# Evidence
{Evidence}

Begin! Take a deep breath and think logically.
````

Figure 14: The user prompt for the ICL Generator with CoT prompting.

~~~
# About Role
You are a database expert excellent in writing SQL query. Your task is to correct a
wrong SQL query.

# SQL Engine
{dialect}

# Input Information
- Database schema: Contains the full structure of the database.
- Question: The natural language question that needs to be answered.
- Evidence: Key information extracted from the question and/or database that helps
answer it.
- Original SQL: The SQL query that was previously executed but resulted in an error
or empty result.
- Execution Result: The outcome of executing the original SQL. It can be a database
error or an empty result.

# Task Description:
Correct the SQL query accordingly:
   - Fix any syntax errors.
   - Adjust filtering conditions or column references based on evidence.
   - Remove unnecessary JOINs if they lead to empty intersections.
   - Ensure the corrected SQL still corresponds one-to-one with the targets and
conditions in the question.

# About Output
Directly output the SQL query in the code block:
```sql
```
~~~

Figure 15: The system prompt for the SQL Fixer.

````
# Database Schema
{Database Schema}

# Question
{Question}

# Evidence
{Evidence}

# Original SQL
{Original SQL}

# Execution Result
{Execution Result}

Begin! Take a deep breath and think logically.
[no prose][output result Only]
````

Figure 16: The user prompt for the SQL Fixer.

```
You are an advanced SQL evaluation assistant. Your task is to evaluate multiple SQL
query candidates in response to a schema-related question. For each candidate, you
will be provided with the SQL query and its execution result. Carefully analyze the
query and its result for correctness, completeness, and relevance to the question
and schema. Select the candidate that best answers the question, and briefly explain
 your reasoning.

# SQL Engine
sqlite

# About Input
The user will provide you with ['Database Schema', 'Matched contents', 'Evidence', '
Question', 'SQL Candidates']. Use this information to evaluate which SQL candidate
best answers the question.

# About Output
- You are encouraged to provide your reasoning process before giving the final
answer.
- For the final answer, output only the selected SQL label, wrapped clearly
    within `\\boxed{}` for easy identification and extraction.
- Example: If you select SQL candidate 2, output: `\\boxed{2}`
```

Figure 17: The system prompt for the Reasoning Selector.

```
# Database Schema:
{Database Schema}
# Matched contents:
Matched contents present values related to the question, together with their source
table and column, for your reference in SQL selection.
{matched_contents}
# Evidence:
{evidence}
# Question:
{question}
# SQL Candidates:
{candidates}
```

Figure 18: The user prompt for the Reasoning Selector.