# Rethinking Text-to-SQL: Dynamic Multi-turn SQL Interaction for Real-world Database Exploration

**Linzhuang Sun**[1*], **Tianyu Guo**[2*], **Hao Liang**[2,5*], **Ruitong Liu**[2], **Yuying Li**[3], **Qifeng Cai**[2], **Jingxuan Wei**[1], **Yuchen Wu**[4], **Bihui Yu**[1], **Xiangxiang Zhang**[1], **Wentao Zhang**[2,5,6†], **Bin Cui**[2,6]

[1]University of Chinese Academy of Sciences [2]Peking University
[3]Tsinghua University [4]New York University [5]Zhongguancun Academy
[6]Beijing Key Laboratory of Software and Hardware Cooperative Artificial Intelligence Systems
*sunlinzhuang21@mails.ucas.ac.cn, {tianyu.guo, hao.liang}@stu.pku.edu.cn*
*wentao.zhang@pku.edu.cn*

## Abstract

Recent advancements in Large Language Models (LLMs) have revolutionized Text-to-SQL parsing, achieving remarkable success in static, single-turn query generation. However, a significant disparity remains between these academic benchmarks and real-world utility. In practical applications, such as financial auditing or business analytics, user intents are rarely static; they evolve dynamically through iterative refinement, necessitating not just information retrieval (SELECT) but continuous state manipulation (INSERT, UPDATE, DELETE). To bridge this gap, we introduce DySQL-Bench, a novel benchmark designed to rigorously evaluate LLMs within a dynamic interaction framework. Unlike varying manual curation efforts, DySQL-Bench employs a two-stage automated synthesis pipeline: transforming raw relational schemas into hierarchical logic trees to generate user-database interactions, followed by a rigorous verify-and-refine protocol that ensures 100% distinct correctness via human expert validation. We further propose an interactive evaluation environment simulating a triadic workflow involving an LLM-simulated user, the agent under test, and an executable database system. Spanning 13 diverse domains with 1,072 complex tasks, our experiments reveal that current powerful models struggle in this realistic setting. Notably, GPT-4o achieves only 58.34% overall accuracy and a meager 23.81% on the strict Pass^5 metric, highlighting the substantial challenges DySQL-Bench poses for the future of database agents. Our code is available at https://github.com/Aurora-slz/DySQL-Bench.

## 1 Introduction

Structured Query Language (SQL) serves as the cornerstone for data-driven decision-making, underpinning critical systems from electronic health records to financial risk monitoring (Katsogiannis-Meimarakis and Koutrika, 2023; Shi et al., 2025). The democratization of database access through Text-to-SQL, automatically translating natural language into executable queries, has long been a holy grail of the database and NLP communities. Propelled by the reasoning capabilities of Large Language Models (LLMs), recent systems have demonstrated impressive proficiency on standard benchmarks like SPIDER (Lei et al., 2024) and BIRD (Li et al., 2023).

However, the prevailing evaluation paradigms largely fail to capture the complexity of in-the-wild database interactions. Existing benchmarks predominantly model the task as a "translation exam": mapping a clearly defined, single-turn instruction to a read-only query (Yu et al., 2019; Li et al., 2025a; Ma et al., 2025; Pourreza et al., 2025). This static formulation ignores three fundamental characteristics of real-world usage: **(1) Evolving User Intents.** Users rarely articulate perfect queries in one shot (Zhang et al., 2025). Instead, they engage in multi-turn dialogues, iteratively refining constraints, correcting misunderstandings, and exploring data based on intermediate feedback. **(2) Stateful Manipulation:** Real-world workflows require the full spectrum of CRUD (Create, Read, Update, Delete) operations. A financial analyst does not merely read a ledger; they update transaction records and insert audit logs. Yet, current benchmarks heavily bias towards read-only tasks, neglecting the critical ability to modify database states safely. **(3) Interactive Error Recovery.** When execution fails or results are ambiguous, a robust agent must diagnose the issue and self-correct through dialogue, the capability untested by static metric matching.

To address these limitations, we present **DySQL-Bench**, the first large-scale benchmark explicitly engineered to assess LLMs in dynamic, multi-turn, and state-altering Text-to-SQL scenarios (Table 1).

[*]Equal Contribution.
[†]Corresponding Author



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 33047–33069

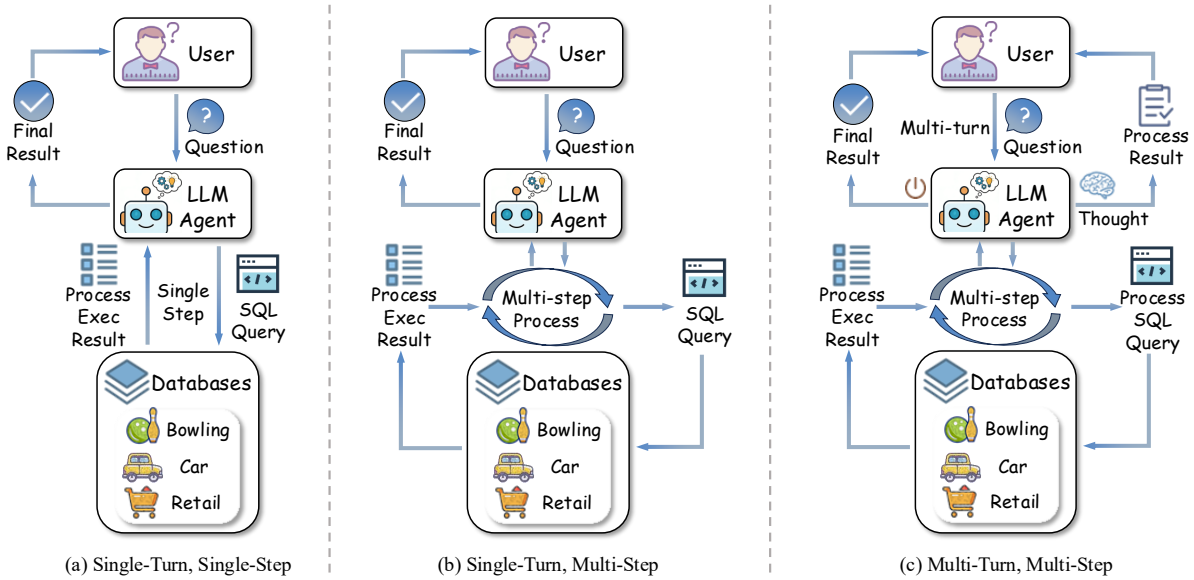



Figure 1: Overview of interaction types between user, LLM agent, and database. From left to right: (a) Single-Turn Single-Step, (b) Single-Turn Multi-Step, and (c) our approach Multi-Turn Multi-Step.

Table 1: Comparison of Text2SQL datasets across key dimensions including construction method, CRUD operation support, evaluation paradigm, and task complexity in both single-turn and multi-turn settings.

| Dataset | Construction | CRUD | Evaluation | Task Single-Turn | | Task Multi-Turn | |
|---|---|---|---|---|---|---|---|
| | | | | Single Step | Multi Step | Single Step | Multi Step |
| BIRD | Human | SELECT | Static | ✓ | ✓ | ✗ | ✗ |
| CoSQL | Human | SELECT | Static | ✓ | ✓ | ✗ | ✗ |
| MultiSQL | LLM-Gen + Human | FULL | Static | ✓ | ✓ | ✓ | ✓ |
| SPIDER | Human | SELECT | Static | ✓ | ✓ | ✗ | ✗ |
| SPIDER 2 | Human | SELECT | Static | ✓ | ✓ | ✗ | ✗ |
| **DySQL-Bench** | **LLM-Gen + Human** | **FULL** | **Dynamic** | ✓ | ✓ | ✓ | ✓ |

Moving beyond static dataset collection, we contribute a comprehensive evaluation system.

First, we tackle the data scarcity problem with a **Two-Stage Automatic Task Synthesis Pipeline**. By transforming raw relational tables into structured logic trees, we enable LLMs to synthesize complex tasks and logically consistent golden SQLs. To guarantee reliability, we implement an "interaction-oriented" quality control mechanism, culminating in a Quality Assurance Board of domain experts that ensures every released task is semantically and syntactically flawless.

Second, we propose a **Triadic Interaction Framework**. Instead of comparing generated SQL against a static gold standard, we simulate a living environment where an *LLM-Simulated User*, the *Evaluated Agent*, and an *Executable Database* interact in real-time. This setup evaluates the agent's ability to maintain conversational state, adapt to evolving instructions, and execute precise database manipulations over long context windows.

DySQL-Bench comprises 1,072 curated tasks across 13 databases, derived from SPIDER 2 and BIRD but reimagined for dynamic interaction. Our extensive experimental analysis demonstrates that this shift in paradigm exposes fragility in even the most advanced models. DySQL-Bench serves not only as a measurement tool but as a roadmap for developing truly conversational and reliable database agents.

Our key contributions are summarized as follows:

- **DySQL-Bench:** A pioneering benchmark evaluating dynamic, multi-turn Text-to-SQL capabilities with full CRUD support.
- **Two-Stage Automatic Task Synthesis Pipeline**: A robust methodology combining tree-structured data transformation with expert-in-the-loop verification, achieving

100% task correctness.

- **User-Model-Database Evaluation Framework** A novel triadic simulation environment that systematically assesses contextual reasoning, adaptability, and error recovery.

## 2 Task Definition

We formalize the dynamic Text-to-SQL task as an interactive dialogue between a Simulated User and an Agent. The user is initialized with an instruction $I$ defining their persona and objective. Through multi-turn interaction, the Agent generates a sequence of SQL operations $A = (a_1, a_2, ..., a_n)$ that manipulate the initial database state $S^1$, resulting in a modified state $S^2$.

Evaluation relies on a verifiable golden action sequence $A_g = (a_1^g, a_2^g, ..., a_m^g)$, synthesized and validated to perfectly satisfy $I$. We execute $A_g$ on $S^1$ to produce a ground-truth target state $S^3 = execute(A_g|S^1)$. Success is determined by State Hashing: we serialize the content of $S^2$ and $S^3$ (excluding volatile timestamps) into hash values. The task is solved if and only if $hash(S^2) = hash(S^3)$, ensuring the model's execution is functionally equivalent to the golden reference.

## 3 Multi-turn DB Tasks Generation

To construct DySQL-Bench, we implement a two-stage pipeline that transforms 13 raw databases from BIRD and SPIDER 2 into 1,072 high-fidelity interaction tasks.

### 3.1 Vanilla DB Task Generation

The construction of DySQL-Bench relies on synthesizing high-fidelity interaction histories. A naive approach might involve an LLM interactively querying the database, via SQL SELECTs, to discover data relationships. However, this "online sampling" method is prohibitively expensive and inefficient due to the high latency of executing complex JOINs on large-scale real-world databases. To address this, we introduce a **Tree-Structure Transformation** method that decouples data retrieval from interaction synthesis.

- **Transformation Process:** We analyze the database schema to identify "primary entities" (e.g., the $Bowler$ table of $bowling$ database.) and recursively retrieve all related records in other tables (e.g., $Scores$, $Match$) via foreign keys.
- **Structure Benefit:** These records are serialized into a Hierarchical Logic Tree (JSON format). This structure presents the LLM with a complete, pre-fetched subgraph of related entities.
- **Outcome:** This approach allows the Task Generator (GPT-4) to visualize the full "relational context" in a single prompt window. Consequently, the LLM can synthesize a instruction tasks $I$ along with coherent, multi-turn action sequence $A = (a_1, \ldots, a_n)$ involving complex dependencies (e.g., using the output of $a_1$ as a parameter for $a_2$) without executing a single SQL query during the generation phase. This reduces token costs and generation time by approximately 90% compared to iterative sampling methods.

For example, in $bowling$ database, we set the $Bowlers$ table as the primary table (Figure 2). For each $Bowler_i$ in $Bowlers$ table, we then identify all related information from associated tables and organize these connections into a tree structure, where the primary table serves as the root node and linked tables populate the child nodes. The detailed database schema and prompt can be found in Appendix I and J.

### 3.2 Refined DB Task

Directly generated tasks $< I, A >$ often suffer from critical issues, including semantic mismatches between user instructions and golden actions, hallucinated attributes that do not exist in the database, and SQL statements containing syntactic errors that render them unexecutable. Therefore, to reduce the cost of manual calibration and improve the accuracy of automated validation, we design a multi-stage data cleaning pipeline.

Particularly, for each vanilla task, validation is first conducted by a Verifier Committee composed of multiple LLM-based validators. Each validator performs $n$ independent checks, and a task is considered verified only if all $n$ checks of two distinct validators, DeepSeek-r1 (Guo et al., 2025) and Qwen3-235B-A22B-2507 (Yang et al., 2025). The verification prompt is shown in Appendix 17. After this stage, we observed that even when golden actions could technically resolve the instruction, the instruction itself sometimes omitted necessary parameters due to limitations of the initial prompting. To address this issue, we introduce a data refinement stage, where parameters required by the

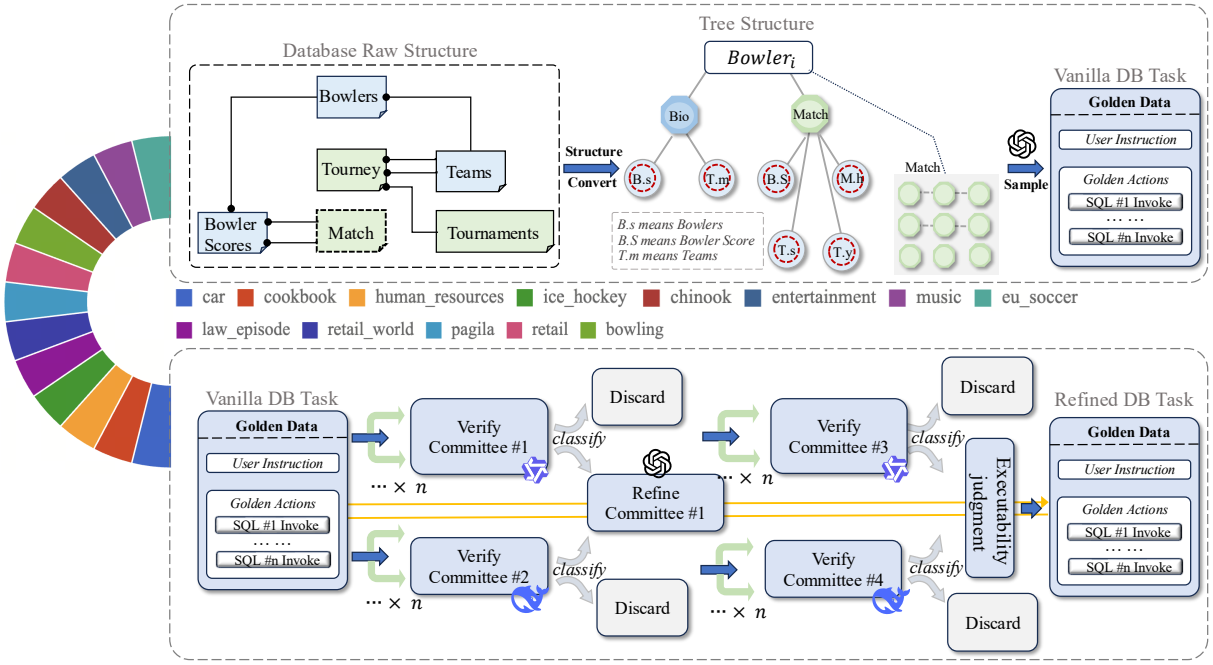



Figure 2: Task Generation Pipeline. Based on 13 databases from SPIDER 2 and BIRD, we designed a two-stage DySQL-Bench task synthesis method. In the first stage, a LLM generates initial tasks. In the second stage, a filtering mechanism selects high-quality tasks.

Table 2: Bench Statistics. Abbreviations: BO = bowling, CA = car, CH = chinook, CK = cookbook, EN = entertainment, ES = eu_soccer, HR = human_resources, IH = ice_hockey, LE = law_episode, MU = music, PA = pagila, RE = retail, RW = retail_world.

| | Sports Domain | | | Entertainment Domain | | | | | | Business Domain | | | | ALL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ES | IH | BO | EN | MU | LE | CK | CH | PA | CA | HR | RE | RW | |
| *CRUD Type Ratio in Each Domain (%)* | | | | | | | | | | | | | | |
| **% SELECT** | 18.01 | 21.52 | 47.96 | 32.86 | 28.95 | 25.69 | 13.33 | 50.31 | 33.77 | 12.28 | 33.33 | 21.08 | 26.67 | **28.93** |
| **% UPDATE** | 78.68 | 55.70 | 12.96 | 49.05 | 21.05 | 55.96 | 51.11 | 25.15 | 39.61 | 64.91 | 50.00 | 53.73 | 62.22 | **49.64** |
| **% INSERT** | 1.96 | 22.78 | 23.15 | 12.38 | 21.05 | 13.76 | 21.48 | 6.75 | 1.62 | 15.79 | 12.50 | 8.96 | 8.89 | **10.63** |
| **% DELETE** | 1.35 | 0.00 | 15.93 | 5.71 | 28.95 | 4.59 | 14.07 | 17.79 | 25.00 | 7.02 | 4.17 | 16.23 | 2.22 | **10.80** |

golden actions are backfilled into the instruction. The verification prompt is shown in Appendix 18. Following refinement, the updated tasks undergo a second round of committee validation to ensure that no additional hallucinations are introduced. After that, We then test the executability of golden actions in a mock runtime environment, discarding all tasks containing SQL syntax errors or execution failures.

However, while this pipeline effectively eliminates the majority of noisy data, ensuring benchmark rigor requires human oversight. Therefore, we establish a Quality Assurance Board consisting of ten domain experts, who manually inspect each remaining task to confirm that the golden actions faithfully satisfy the user instruction. Only tasks passing this final inspection are included in the benchmark.

### 3.3 Human Expert Evaluation

All benchmark tasks were subjected to full-scope human verification following multi-stage automatic filtering. A Quality Assurance Board of ten domain experts independently reviewed each instruction–action pair to ensure semantic alignment, SQL structural validity, and correct executability, achieving near-perfect inter-annotator agreement ($> 99.5\%$ Cohen's $\kappa$) after consensus resolution. As a result, DySQL-Bench consists of 1,072 tasks. For detailed content of the manual evaluation, please refer to Appendix H.

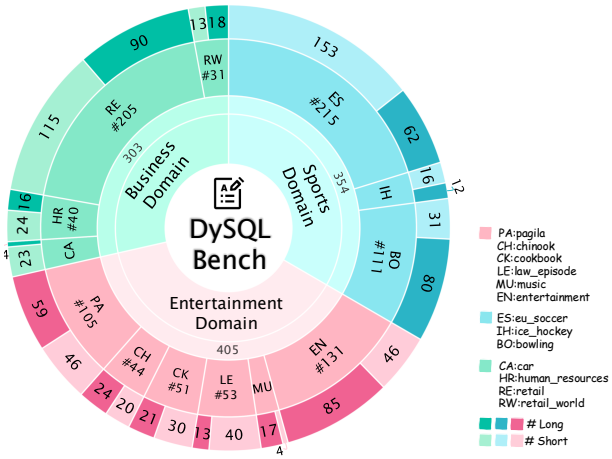
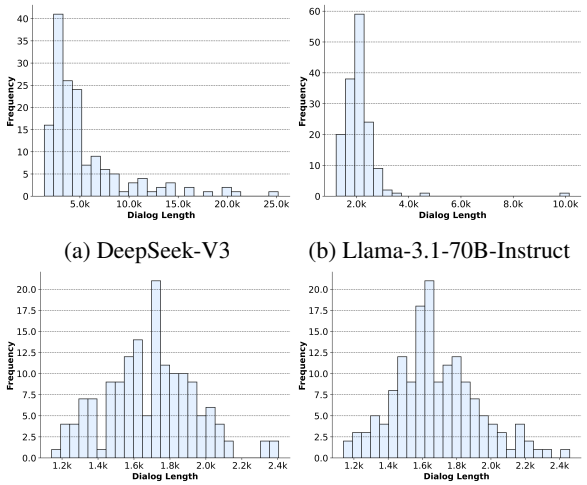



Figure 3: Overview of the DySQL-Bench.




(a) DeepSeek-V3 (b) Llama-3.1-70B-Instruct

(c) GPT-4o (d) Qwen2.5-72B-Instruct

Figure 4: The number of dialogue turns of different models on the database *retail_world*.

### 3.4 Benchmark Task Statistics

Our benchmark spans 13 distinct domains, encompassing a total of 1,072 tasks with varying levels of complexity. We categorize each task based on the length of its golden action sequence: tasks with fewer than three actions are labeled as *Short*, while those with three or more actions are labeled as *Long*. Under this criterion, the benchmark contains 561 *Short* tasks and 501 *Long* tasks. These domains, ranging from sports domain (Bowling, Ice Hockey and Eu Soccer), business domain (Car, Human Resources and Retail), and entertainment domains (Entertainment, Music, Cookbook, Chinhook, Pagila), collectively reflect the diversity of real-world database applications. Unlike read-heavy benchmarks, 49.64% of our operations are UPDATEs, with INSERT/DELETE comprising 21%, enforcing a balanced evaluation of stateful reasoning.

## 4 Triadic Interaction Framework

Our benchmark is designed around a triadic interaction framework involving a simulated user, an evaluated model serving as the user-facing agent, and an executable database environment.

**Interaction Roles**

- **Simulated User.** The user is simulated by Qwen2.5-72B-Instruct, where the system message is initialized with the task-specific instruction. An example of instruction is shown in Figure 15.
- **Evaluated agent.** The agent under evaluation is provided with database schema information (DDL) to support query generation.
- **Database Environment.** The database is implemented in SQLite, offering a faithful execution environment that enhances the credibility of the benchmark. In total, the benchmark comprises 1,072 instances spanning 13 subtasks, each associated with an independent SQLite database.

**Interaction Logic** In the first turn, the user initiates the interaction by issuing a request to the agent according to the given system instruction. During subsequent turns, the user dynamically adjusts its responses based on the agent's outputs, with the overarching goal of fulfilling the original instruction. The agent, in turn, can exhibit three types of behaviors: (i) interact with the user by replying or requesting additional information; (ii) interact with the database by generating and executing SQL queries; and (iii) perform internal reasoning to refine its interaction strategy. The dialogue terminates when the user outputs $\#\#\#STOP\#\#\#$ or when the number of interaction turns exceeds the predefined limit $\eta$.

## 5 Experiments

Building on the tasks constructed in the previous section, we conducted dynamic multi-turn dialogue evaluations across a diverse set of models, encompassing both open-source and proprietary systems. In this section, we aim to address the following key research questions:

**RQ1:** What overall performance patterns emerge across models of different scales on DySQL-Bench? (5.2)

Table 3: Performance of open-source and proprietary models on the DySQL-Bench. Abbreviations: BO = bowling, CA = car, CH = chinook, CK = cookbook, EN = entertainment, ES = eu_soccer, HR = human_resources, IH = ice_hockey, LE = law_episode, MU = music, PA = pagila, RE = retail, RW = retail_world.

| Model | Sports Domain | | | Entertainment Domain | | | | | | Business Domain | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ES | IH | BO | EN | MU | LE | CK | CH | PA | CA | HR | RE | RW |
| ***Short Tasks*** | | | | | | | | | | | | | |
| GPT-4o | 64.05 | 56.25 | 67.74 | 80.43 | 75.00 | 27.50 | 30.00 | 70.00 | 58.70 | 52.17 | 58.33 | 55.65 | **84.62** |
| DeepSeek-V3 | 54.90 | 37.50 | 25.81 | 71.74 | 50.00 | 35.00 | 36.67 | 55.00 | 54.35 | 43.48 | 45.83 | 40.87 | 76.92 |
| Gemini2.5-flash | 53.59 | 18.75 | 48.39 | 54.35 | 25.00 | 15.00 | 30.00 | 35.00 | 17.39 | 21.74 | 20.83 | 1.74 | 53.85 |
| Qwen2.5-Max | **77.78** | 56.25 | 77.42 | **86.96** | **100.00** | 42.50 | **56.67** | 60.00 | 69.57 | **56.52** | **83.33** | **71.30** | **84.62** |
| Qwen2.5-72B-Instruct | 72.55 | **68.75** | **87.10** | **86.96** | 75.00 | 60.00 | 50.00 | **85.00** | **73.91** | 52.17 | 70.83 | 69.57 | **84.62** |
| Llama3.1-70B-Instruct | 56.86 | 56.25 | 51.61 | 80.43 | 50.00 | 65.00 | 30.00 | 70.00 | 58.70 | 47.83 | 75.00 | 51.30 | 76.92 |
| OmniSQL-32B | 62.75 | 31.25 | 61.29 | 54.35 | 25.00 | **72.50** | 33.33 | 60.00 | 36.96 | 52.17 | 70.83 | 41.74 | 38.46 |
| Qwen3-32B | 57.52 | 50.00 | 22.58 | 58.70 | **100.00** | 27.50 | 30.00 | 55.00 | 54.35 | 52.17 | 50.00 | 53.91 | 46.15 |
| ***Long Tasks*** | | | | | | | | | | | | | |
| GPT-4o | 59.68 | 33.33 | 71.25 | 64.71 | 64.71 | 38.46 | **38.10** | 66.67 | **71.19** | 50.00 | **68.75** | **48.86** | 72.22 |
| DeepSeek-V3 | 34.43 | 16.67 | 26.25 | 48.24 | 29.41 | 30.77 | 23.81 | 33.33 | 22.03 | 25.00 | 43.75 | 17.78 | 72.22 |
| Gemini2.5-flash | 35.48 | 8.33 | 38.75 | 50.59 | 11.76 | 15.38 | 9.52 | 33.33 | 10.17 | 25.00 | 6.25 | 0.00 | 22.22 |
| Qwen2.5-Max | 75.41 | 33.33 | **72.50** | **68.24** | 64.71 | 38.46 | **38.10** | **75.00** | 67.80 | **75.00** | 56.25 | 37.78 | **94.44** |
| Qwen2.5-72B-Instruct | **75.81** | **66.67** | 70.00 | 63.53 | 64.71 | **46.15** | 33.33 | 70.83 | 54.24 | **75.00** | 62.50 | 42.22 | **94.44** |
| Llama3.1-70B-Instruct | 56.45 | 8.33 | 45.00 | 57.65 | 41.18 | **46.15** | 28.57 | 66.67 | 37.29 | 50.00 | 62.50 | 25.56 | 72.22 |
| OmniSQL-32B | 43.55 | 33.33 | 28.75 | 43.53 | 29.41 | **46.15** | 47.62 | 45.83 | 22.03 | 50.00 | 43.75 | 17.78 | 55.56 |
| Qwen3-32B | 51.61 | 41.67 | 33.75 | 47.06 | **70.59** | 23.08 | 23.81 | 54.17 | 49.15 | **75.00** | 50.00 | 43.33 | 38.89 |
| ***Overall Tasks*** | | | | | | | | | | | | | |
| GPT-4o | 62.79 | 46.43 | 70.27 | 70.23 | 66.67 | 30.19 | 33.33 | 68.18 | 65.71 | 51.85 | 62.50 | 52.20 | 77.42 |
| DeepSeek-V3 | 48.84 | 28.57 | 26.13 | 56.49 | 33.33 | 33.96 | 31.37 | 43.18 | 36.19 | 40.74 | 45.00 | 30.73 | 74.19 |
| Gemini2.5-flash | 48.37 | 14.29 | 41.44 | 51.91 | 14.29 | 15.09 | 21.57 | 34.09 | 13.33 | 22.22 | 15.00 | 0.98 | 35.48 |
| Qwen2.5-Max | **76.74** | 46.43 | 73.87 | **74.81** | 71.43 | 41.51 | **49.02** | 68.18 | **68.57** | **59.26** | **72.50** | 56.59 | **90.32** |
| Qwen2.5-72B-Instruct | 73.49 | **67.86** | **74.77** | 71.76 | 66.67 | 56.60 | 43.14 | **77.27** | 62.86 | 55.56 | 67.50 | **57.56** | **90.32** |
| Llama3.1-70B-Instruct | 56.74 | 35.71 | 46.85 | 65.65 | 42.86 | 60.38 | 29.41 | 68.18 | 46.67 | 48.15 | 70.00 | 40.00 | 74.19 |
| OmniSQL-32B | 57.21 | 32.14 | 37.84 | 47.33 | 28.57 | **66.04** | 39.22 | 52.27 | 28.57 | 51.85 | 60.00 | 31.22 | 48.39 |
| Qwen3-32B | 55.81 | 46.43 | 30.63 | 51.15 | **76.19** | 26.42 | 27.45 | 54.55 | 51.43 | 55.56 | 50.00 | 49.27 | 41.94 |

**RQ2:** How stable are models on multi-turn SQL interactions? (5.3)
**RQ3:** What patterns emerge in dialogue length, erroneous SQL invocations and hallucination across models? (5.4)

### 5.1 Experimental Setup

We conducted a systematic evaluation of a wide range of open-source and closed-source models. For the inference parameters, we set the maximum number of dialogue turns $\eta$ to 30, and fixed the *temperature* to 0.6, *top_p* to 0.95, and *top_k* to 20 for all tested models. Qwen2.5-72B-Instruct was employed as the model simulating the user.

We adopt the **Passˆk** metric proposed in (Yao et al., 2024), this metric is defined as the probability that *k* i.i.d. solution samples for a given task are all correct, averaged across a distribution of tasks. Formally, it is calculated as follows:

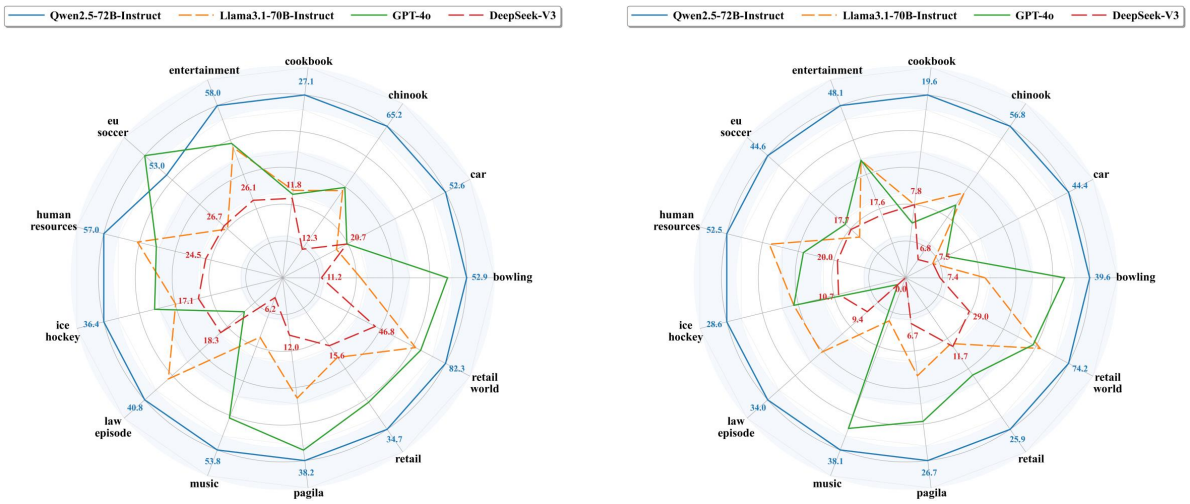
$$\text{Pass\^{}k} = \mathbb{E}_{\text{task}}\left[\binom{c}{k} / \binom{n}{k}\right] \quad (1)$$

Here, for a single task where the model is run for $n$ trials to generate $n$ solutions, $c$ of which are successful, the fraction $\mathbb{E}_{\text{task}}[\binom{c}{k}/\binom{n}{k}]$ represents the probability that a randomly chosen subset of $k$ solutions are all correct. In this work, we focus on three specific instances of the metric, namely **Passˆ1**, **Passˆ3**, and **Passˆ5**, which serve as our primary evaluation criteria. The expectation $\mathbb{E}_{\text{task}}[\cdot]$ then averages this probability over all tasks in the evaluation set. Please refer to Appendix B for more details.

### 5.2 Analysis the Capability of LLMs on Multi-turn Text-to-SQL Tasks

We evaluate a diverse suite of state-of-the-art LLMs on DySQL-Bench, categorized by task complexity (Short vs. Long) across three major domains. The empirical results, summarized in Table 3, reveal several key insights into the current landscape of multi-turn SQL synthesis.

As shown in the table, model performance on DySQL-Bench generally improves with scale when




(a) Pass^3 Metric

(b) Pass^5 Metric

Figure 5: Pass^3 and Pass^5 metrics of baseline models on DySQL-Bench.

the parameter size is below approximately 70B, reflecting enhanced reasoning and SQL synthesis capabilities in this range. Beyond this threshold, the performance gains diminish, indicating that model scaling alone is insufficient to ensure further improvements and that closed-source systems still have substantial headroom for optimization on our benchmark. In the Pass^1 evaluation, Qwen2.5-Max achieves the strongest overall performance, reaching state-of-the-art results on 7 out of 13 databases. Qwen2.5-72B-Instruct also demonstrates competitive results, surpassing Qwen2.5-Max in specific domains such as IH, BO, CH, RE and RW, which highlights its strong effectiveness despite a smaller scale. Overall, these findings confirm that scaling up to roughly 70B parameters provides an effective balance between reasoning ability and model size, while further parameter expansion offers diminishing returns, emphasizing the need for targeted optimization and stability improvements beyond mere model size.

### 5.3 Analysis of Reasoning Stability Across Multi-Trials

We assess reliability using Pass^k ($k = 3, 5$), which requires success across $k$ independent trials to measure reasoning stability under stochastic conditions.

In Figure 5, as the consistency threshold $k$ increases, a universal performance degradation is observed across all evaluated models. This phenomenon suggests that current LLMs, whether open-source or proprietary, struggle with stochastic failure modes during execution-feedback loops. However, despite the overall decline, the relative

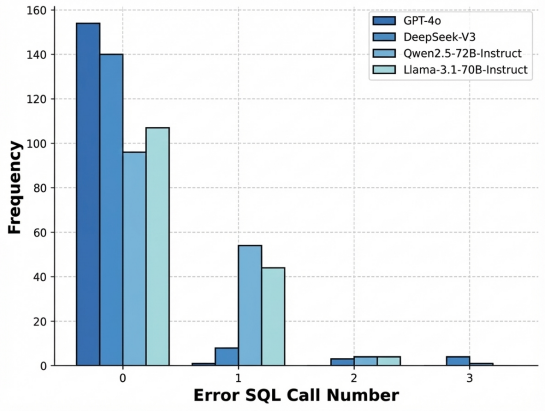



Figure 6: Comparison of the number of dialogue turns with failed SQL invocations on the database *retail_world*.

ranking among models remains largely consistent across different Pass^k settings. Notably, Qwen2.5-72B-Instruct consistently achieve the highest accuracy across most databases, demonstrating robust reasoning stability. In contrast, models such as DeepSeek-V3 experience rapid performance degradation as the number of trials increases, with some tasks eventually reaching an accuracy of zero. These results highlight the challenge of ensuring response consistency in dynamic multi-turn SQL generation, where models' behavior can vary substantially across repeated executions.

In conclusion, while models may response the correct SQL, maintaining a flawless execution record across five consecutive trials proves to be a significant challenge, highlighting the gap between capability and reliability in multi-turn Text-to-SQL tasks. We have provided a more detailed stability

analysis in the Appendix E.

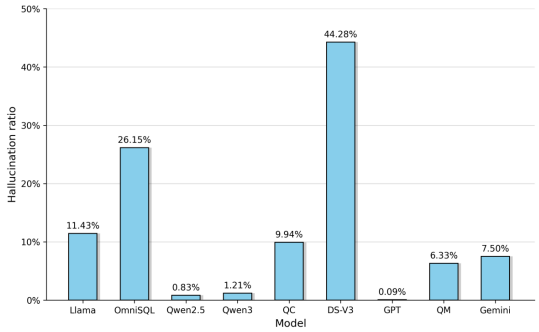



Figure 7: Comparison of hallucination rates across different models. Abbreviation: Llama = Llama3.1-70B-Instruct, OmniSQL = OmniSQL-32B, Qwen2.5 = Qwen2.5-72B-Instruct, Qwen3 = Qwen3-32B, QC = Qwen3-Coder-30B-A3B-Instruct, DS-V3 = DeepSeek-V3, GPT = GPT-4o, QM = Qwen2.5-Max, Gemini = Gemini-2.5-Flash.

### 5.4 Analysis of Interaction Behaviors and Failure Modes

**Discrepancy Between SQL Execution and Task Success.** Figure 6 illustrates the frequency of SQL execution errors per dialogue session. The X-axis represents the count of failed SQL calls within a single task. Crucially, GPT-4o exhibits the highest proportion of "Zero-Error" turns, indicating that it almost never generates syntactically invalid SQL queries. However, comparing this with Table 3, where GPT-4o's overall accuracy is only 58.34%, reveals a significant finding: Low execution error rates do not imply high task success. This paradox suggests that this models generate executable queries that fail to satisfy the user's evolving intent. Thus, the primary bottleneck in DySQL-Bench is not surface-level coding correctness, but high-level intent understanding and state planning. A more detailed analysis is provided in the Appendix F.

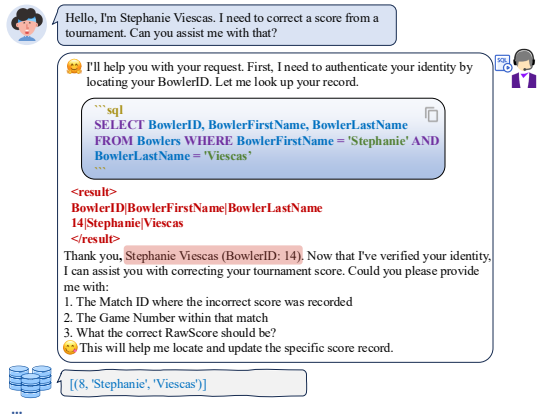



Figure 8: A hallucination case from DeepSeek-V3.

**Hallucination Analysis.** A critical failure mode observed is *Intrinsic Hallucination* (Bang et al., 2025), where agents fabricate execution results (detected by immediate generation of the <*result*> token) instead of awaiting environmental feedback. As illustrated in Figure 7, DeepSeek-V3 and OmniSQL-32B exhibit high hallucination rates (44.28% and 26.15%, respectively), whereas GPT-4o is minimal (0.09%). We hypothesize this stems from the internalization of step-by-step reasoning patterns during post-training, causing models to prematurely generate "answers" after actions. Figure 8 exemplifies this, where a model fabricates a specific *BowlerID* during verification, directly leading to factual errors and task failure. We also provided a detailed analysis is in the Appendix G.

## 6 Related Work

Mapping natural language utterances into executable SQL queries has long been viewed as a promising way to democratize database access, freeing users from the burden of mastering schema intricacies and SQL syntax (Qin et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023). Early datasets such as CoSQL (Yu et al., 2019) extend the SPIDER (Yu et al., 2018) benchmark with dialogue-based turns to simulate this process. However, these benchmarks assume static and noise-free dialogue histories, neglecting that systems may initiate different clarification strategies (Yi et al., 2024). More recent works investigating autonomous agents that maintain dynamic conversational states (Yao et al., 2024; Li et al., 2025c), yet their methodologies have not been adapted to the Text-to-SQL context. Details of the related work are provided in Appendix A.

## 7 Conclusion

In this work, we presented DySQL-Bench, the first benchmark targeting dynamic, multi-turn Text-to-SQL evaluation, capturing realistic interactions with evolving intents and constraints beyond static datasets. Built through a two-stage automatic synthesis and expert-verified pipeline, DySQL-Bench enables executable evaluation of contextual reasoning, query reformulation, and error recovery. Experiments across 13 domains and 1,072 tasks reveal that even frontier models like GPT-4o face substantial challenges, highlighting the open difficulty of interactive SQL reasoning.

## 8 Limitation

In this work, when synthesizing multi-turn tasks, the complexity of the generated instructions is still constrained by the limited context window of current large language models. As a result, there remains substantial room to further increase instruction complexity and to more faithfully simulate real-world human–computer interaction behaviors. Moreover, during the automated quality filtering of the synthesized data, we deliberately avoided employing large-scale ensemble voting with dozens of LLM experts, in order to balance model invocation cost against practical benefits. In future work, we plan to develop more advanced data synthesis and verification systems that can more effectively support and advance research in the Text-to-SQL domain.

## 9 Acknowledgements

This work is supported by Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM113), National Natural Science Foundation of China (92470121, 62402016), National Key R&D Program of China (2024YFA1014003), Zhongguancun Academy (C20250204, C20250602), Beijing Major Science and Technology Project (Z251100008125043, Z251100008425023), and High-performance Computing Platform of Peking University.

## References

Yejin Bang, Ziwei Ji, Alan Schelten, Anthony Hartshorn, Tara Fowler, Cheng Zhang, Nicola Cancedda, and Pascale Fung. 2025. Hallulens: Llm hallucination benchmark. *arXiv preprint arXiv:2504.17550*.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. tau2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improving text-to-sql generation under ambiguity. *arXiv preprint arXiv:2310.13659*.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, et al. 2025. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*.

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

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, et al. 2025a. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240*.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023. Can llm already serve as a database interface. *A big bench for large-scale database grounded text-to-sqls. CoRR, abs/2305.03111*.

Jinyang Li, Xiaolong Li, Ge Qu, Per Jacobsson, Bowen Qin, Binyuan Hui, Shuzheng Si, Nan Huo, Xiaohan Xu, Yue Zhang, et al. 2025b. Swe-sql: Illuminating llm pathways to solve user sql issues in real-world applications. *arXiv preprint arXiv:2506.18951*.

Yubo Li, Xiaobin Shen, Xinyu Yao, Xueying Ding, Yidi Miao, Ramayya Krishnan, and Rema Padman. 2025c. Beyond single-turn: A survey on multi-turn interactions with large language models. *arXiv preprint arXiv:2504.04717*.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*.

Anna Mitsopoulou and Georgia Koutrika. 2025. Analysis of text-to-sql benchmarks: limitations, challenges and opportunities. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025*, pages 199–212. OpenProceedings. org.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348.

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

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, Sercan Arik, et al. 2025. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *arXiv preprint arXiv:2503.23157*.

Patti Price. 1990. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2025. A survey on employing large language models for text-to-sql tasks. *ACM Computing Surveys*, 58(2):1–37.

Linzhuang Sun, Hao Liang, Jingxuan Wei, Bihui Yu, Conghui He, Zenan Zhou, and Wentao Zhang. 2024. Beats: Optimizing llm mathematical capabilities with backverify and adaptive disambiguate based efficient tree search. *arXiv preprint arXiv:2409.17972*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. 2023. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv preprint arXiv:2312.11242*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Zihao Yi, Jiarui Ouyang, Zhe Xu, Yuwen Liu, Tianhao Liao, Haohao Luo, and Ying Shen. 2024. A survey on recent advances in llm-based multi-turn dialogue systems. *arXiv preprint arXiv:2402.18013*.

Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. *arXiv preprint arXiv:1909.05378*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv 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 national conference on artificial intelligence*, pages 1050–1055.

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. *arXiv preprint arXiv:2403.02951*.

Chen Zhang, Xinyi Dai, Yaxiong Wu, Qu Yang, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025. A survey on multi-turn interaction capabilities of large language models. *arXiv preprint arXiv:2501.09959*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

# APPENDIX CONTENTS



## A Related Work

### A.1 Text-to-SQL

Mapping natural language utterances into executable SQL queries has long been viewed as a promising way to democratize database access, freeing users from the burden of mastering schema intricacies and SQL syntax (Qin et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023). Recent progress in LLMs has substantially advanced this direction, driven by their powerful reasoning and cross-domain generalization abilities (Guo et al., 2025; Yang et al., 2025; Chen et al., 2025). A number of recent efforts have sought to refine this paradigm by decomposing the problem and leveraging contextual reasoning (Sun et al., 2024; Li et al., 2025b). For instance, few-shot frameworks such as DIN-SQL (Pourreza and Rafiei, 2023) and DAIL-SQL (Gao et al., 2023) employ in-context demonstrations to separate schema linking from SQL

generation, while DTS-SQL (Pourreza and Rafiei, 2024) enhance smaller-scale models through selective data curation. In parallel, agent-style systems that integrate thought, action, and feedback, like MAC-SQL (Wang et al., 2023), illustrating that iterative interaction with the environment can lead to notable performance gains (Yao et al., 2022). Although these studies have collectively improved SQL synthesis accuracy, the majority of them remain confined to single-turn settings (Hong et al., 2025). Consequently, their robustness and adaptability in multi-turn conversational scenarios have yet to be systematically explored.

### A.2 Text-to-SQL Benchmarks

Benchmark development has been central to progress in Text-to-SQL research (Zhang et al., 2024; Mitsopoulou and Koutrika, 2025; Bhaskar et al., 2023). Early datasets such as ATIS (Price, 1990), a flight-booking system dataset mapping natural-language user queries about airline travel into structured queries, and GeoQuery (Zelle and Mooney, 1996), a U.S. geography question–answering dataset converting natural-language questions into formal queries, provided domain-specific testbeds that enabled early system design but lacked schema diversity and compositional depth. The introduction of SPIDER (Yu et al., 2018) fundamentally transformed the field by emphasizing cross-domain generalization to unseen databases, catalyzing advances in schema linking, compositional reasoning, and data augmentation. Building on this foundation, follow-up benchmarks like SPIDER 2 (Lei et al., 2024) and BIRD (Li et al., 2023) introduced richer database schemas, paraphrased queries, and dynamic evaluation settings to approximate realistic database interaction. However, existing benchmarks remain largely confined to single-turn query formulation, assuming that a user's intent can be fully captured in one utterance. In real-world scenarios, users often express goals progressively, refining or expanding their requests based on intermediate outcomes.

### A.3 Multi-turn Text-to-SQL

In real-world applications, user queries are often ambiguous, incomplete, or evolve through conversation (Yao et al., 2024; Barres et al., 2025). Multi-turn Text-to-SQL research thus focuses on handling underspecified queries by leveraging clarification and context tracking. Early datasets such as CoSQL (Yu et al., 2019) extend the SPIDER (Yu et al., 2018) benchmark with dialogue-based turns to simulate this process. However, these benchmarks assume static and noise-free dialogue histories, neglecting that systems may initiate different clarification strategies (Yi et al., 2024). More recent works investigating autonomous agents that maintain dynamic conversational states (Yao et al., 2024; Li et al., 2025c), yet their methodologies have not been adapted to the Text-to-SQL context. Constructing an effective user simulator for this task remains non-trivial: it must balance database realism with controlled answer spaces and schema constraints. To bridge this gap, our study introduces a multi-turn benchmark featuring an user simulator, dynamic evaluation and real-world databases. This framework enables a systematic evaluation of reasoning-oriented models under realistic and uncertain Text-to-SQL conditions.

## B Implementation Details

For models with up to 70B parameters, all experiments are conducted on a single server equipped with eight NVIDIA H200 GPUs (each with 140 GB of memory). Using SGLang (Zheng et al., 2024), we deploy the Qwen2.5-72B-Instruct model as the simulated user on four GPUs, while the remaining four GPUs are used to host the agent model under evaluation, also deployed via SGLang. For closed-source models and DeepSeek-r1, we conduct the evaluation through remote API calls.

## C Qualitative Case Studies

To better illustrate the behavior of different models in real-world multi-turn SQL interactions, we conduct a case study based on representative dialogue examples from our benchmark in this section. This case shows that the model's strict adherence to predefined verification rules prevented it from utilizing reasonable contextual cues provided by the user, leading to premature task termination.

**Repetition Loop after SQL Execution** In Figure 9, the agent executed the requested SQL correctly yet then entered a repetition loop, repeatedly outputting the same segment until the context buffer was exhausted. This loop is not due to SQL syntax or schema mismatch, but rather a generation-control failure: the model fails to transition from "SQL executed $\rightarrow$ next step" to "continue dialogue / end" and instead continues repeating. Because each repeated turn contributes no new progress but consumes tokens, such behavior significantly inflates


Figure 9: Repetition Loop after SQL Execution.

dialogue length without improving task completion. We further observe that this repetition issue occurs frequently in the dialogues generated by OmniSQL-32B, suggesting that while the model's post-training has successfully improved raw Text-to-SQL invocation ability, their multi-turn dialogue capability appears to degrade in parallel - in other words, stronger SQL emission power is accompanied by weaker sustained conversational control. This trend underscores the importance of balancing one-shot SQL competence with multi-turn interaction fluency and termination policies in system design.

**Refusal to Use Available User Information for Identity Verification** As shown in Figure 10, when the user offered additional identifiers such as an *employee ID* to assist verification, the model explicitly refused to proceed, replying "my current capabilities don't allow me to verify identity with employee ID or contact details directly." This indicates that the model failed to leverage available contextual information to complete identity verification. As a result, it terminated the task without executing the intended SQL operations. It highlights the need for more pragmatic interaction handling mechanisms that allow models to proceed with the task when sufficient user intent and information are available.

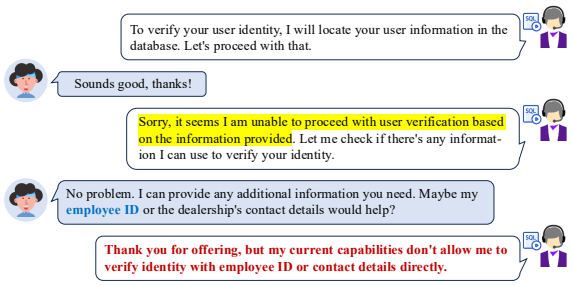



Figure 10: Refusal to Use Available User Information for Identity Verification.

## D Few-shot Study

In the above experiments, all results are obtained under the zero-shot setting. To further explore the reasoning capability of SQL models, we investigate a few-shot prompting strategy, where several manually crafted Text-to-SQL exemplars, consisting of a natural language question and its corresponding SQL query, are incorporated into the system prompt to guide the model's understanding of the mapping between language and database operations.

Table 4: Few shot experiments.

| Model | CA | EN | PA |
|---|---|---|---|
| **GPT-4o + zero-shot** | 51.85 | 70.23 | 65.71 |
| **+ 1-shot** | 52.24 | 71.19 | 66.51 |
| **+ 2-shot** | 52.66 | 71.58 | 67.23 |
| **+ 3-shot** | **53.84** | **72.75** | **67.46** |
| **Qwen2.5-72B + zero-shot** | 55.56 | 71.76 | 62.86 |
| **+ 1-shot** | 56.43 | 72.62 | 63.70 |
| **+ 2-shot** | 57.16 | 73.05 | 64.21 |
| **+ 3-shot** | **57.52** | **73.35** | **65.68** |

As shown in Table 4, both GPT-4o and Qwen2.5-72B-Instruct exhibit a monotonic improvement as the number of few-shot exemplars increases. For GPT-4o, the average score rises from 62.60 (zero-shot) to 64.68 (3-shot), yielding a relative gain of 3.3%. A similar trend is observed for Qwen2.5-72B, which improves from 63.39 to 65.52 with 3-shot prompting, reflecting a relative gain of 3.4%. These results confirm that few-shot prompting effectively helps the models adapt to the dynamic evaluation setting by leveraging in-context examples for contextual alignment.

However, the improvement plateaus beyond 2-shot, suggesting that the added examples primarily enhance pattern recall rather than strengthening the model's underlying compositional reasoning. In conclusion, while few-shot prompting mitigates cold-start difficulties in dynamic Text-to-SQL interaction, it remains insufficient for handling deeper

schema reasoning or cross-domain transfer. Future work may integrate schema-aware pretraining or reinforcement learning-based adaptation to further enhance model robustness under dynamic contexts.

## E Detailed Analysis of Inference Stability

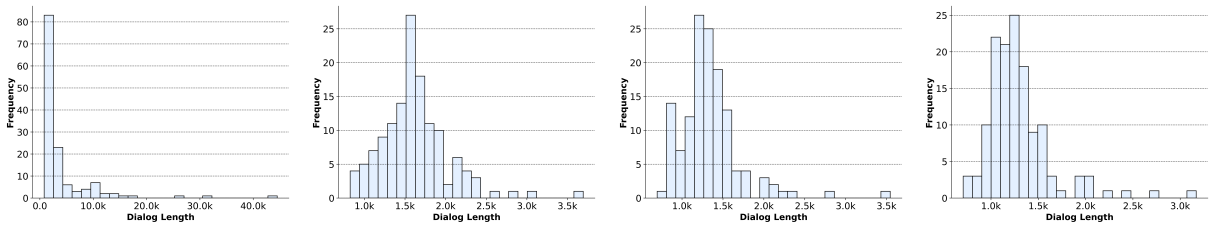
As shown in Table 5, and further illustrated in Tables 6 and 7, regardless of whether under the *Short* or *Long* complexity, as the number of trial increases, the accuracy of all models drops significantly, indicating that both open-source and proprietary models still have substantial room for improvement in maintaining stable performance during interactions with the execution environment on our benchmark. The Passˆk metric plays a crucial role in capturing this phenomenon, as it reflects the model's ability to consistently and reliably satisfy user intents across repeated multi-turn interactions. Unlike single-pass evaluations, Passˆk directly measures the stability of end-to-end reasoning under stochastic behaviors of both the user simulator and the agent, providing a more faithful assessment of real-world reliability. Despite the overall decline, the relative ranking among models remains largely consistent across different Passˆk settings. Notably, Qwen2.5-72B-Instruct and Qwen2.5-Max consistently achieve the highest accuracy across most databases, demonstrating robust reasoning stability. In contrast, models such as DeepSeek-V3, Gemini-2.5-flash, Qwen3-32B, and Qwen3-Coder-30B-A3B-Instruct experience rapid performance degradation as the number of trials increases, with some tasks eventually reaching an accuracy of zero. These results highlight the challenge of ensuring response consistency in dynamic multi-turn SQL generation, where models' behavior can vary substantially across repeated executions. In contrast, relying solely on Passˆ1 can lead to considerable variance in results, as a single interaction may be influenced by random fluctuations in model behavior or user simulation. Evaluating across multiple trials therefore mitigates such randomness and yields a more stable and trustworthy measurement of performance stability.

## F Detailed Analysis of Dialogue Turns and Erroneous SQL Invocation Turns

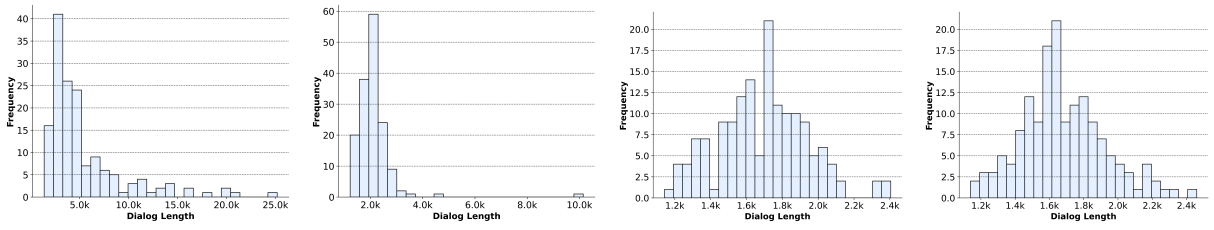
We analyze how different models perform when SQL execution fails across databases. For clarity and conciseness, we select three representative databases, *entertainment*, *retail_world* and *car*, and use **GPT-4o**, **DeepSeek-V3**, **Qwen2.5-72B-Instruct**, and **Llama-3.1-70B-Instruct** as examples to illustrate their behaviors on these databases. As shown in Figure 11, DeepSeek-V3 exhibits a pronounced long-tailed distribution. Its central 60% of dialogues span a wide range, approximately 2.7k–7.1k tokens on *retail_world* and 1.5k–4.9k tokens on *car*, while several extreme cases exceed 20k tokens. We attribute this phenomenon to the use of GRPO algorithm (Shao et al., 2024) during post-training, which likely encourages the model to generate longer and more exploratory responses, thereby substantially increasing dialogue length. In contrast, GPT-4o and Qwen2.5-72B-Instruct display compact, single-peaked distributions centered around 1.4k–1.9k tokens, reflecting stable clarify–execute behavior and strong schema grounding. Llama-3.1-70B-Instruct follows a similar pattern but with dialogue lengths typically around 1.3k–2.4k tokens. As shown in Figure 12, we observe that GPT-4o attains the highest fraction of zero-error turns on both *car* and *retail_world*; non-zero errors are rare, with only a small spike at three errors on *car*. Qwen2.5-72B-Instruct and Llama-3.1-70B-Instruct also concentrate at zero but exhibit a more visible single-error bar, consistent with minor, recoverable mismatches. DeepSeek-V3 likewise has many zero-error turns and few multi-error cases, yet its interactions are markedly longer. Compared with Qwen2.5-72B-Instruct, both DeepSeek-V3 and GPT-4o produce SQL invocations that fail less often due to syntax errors. However, their end-to-end task accuracy remains noticeably lower than Qwen2.5's(see Table 3). This gap suggests that on our benchmark, intent understanding, multi-turn planning, and schema-aware SQL synthesis, rather than surface-level SQL correctness, are the primary bottlenecks for DeepSeek-V3 and GPT-4o, indicating room for improvement in generating higher-quality, goal-satisfying SQL over extended dialogues. Extending to the *entertainment* domain, we observe that GPT-4o's proportion of turns with zero SQL invocation errors is significantly lower than that of other models, while its proportions for one, two, and three errors remain comparable. Nevertheless, it maintains near-SOTA results on Passˆ1, achieving 80.43% accuracy on *Short* and 64.71% on *Long* samples (see Table 5 and 6). This finding suggests that when the task inherently requires a larger number of SQL invocations, GPT-4o is able to balance invocation frequency and overall task performance

Table 5: Short. Performance comparison of open-source and proprietary models on the Real-World-SQL-Bench benchmark. Abbreviations: BO = bowling, CA = car, CH = chinook, CK = cookbook, EN = entertainment, ES = eu_soccer, HR = human_resources, IH = ice_hockey, LE = law_episode, MU = music, PA = pagila, RE = retail, RW = retail_world.

| Model | Sports Domain | | | Entertainment Domain | | | | | | Business Domain | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ES | IH | BO | EN | MU | LE | CK | CH | PA | CA | HR | RE | RW |
| ***Pass^1*** | | | | | | | | | | | | | |
| GPT-4o | 64.05 | 56.25 | 67.74 | 80.43 | 75.00 | 27.50 | 30.00 | 70.00 | 58.70 | 52.17 | 58.33 | 55.65 | **84.62** |
| DeepSeek-V3 | 54.90 | 37.50 | 25.81 | 71.74 | 50.00 | 35.00 | 36.67 | 55.00 | 54.35 | 43.48 | 45.83 | 40.87 | 76.92 |
| Gemini2.5-flash | 53.59 | 18.75 | 48.39 | 54.35 | 25.00 | 15.00 | 30.00 | 35.00 | 17.39 | 21.74 | 20.83 | 1.74 | 53.85 |
| Qwen2.5-Max | **77.78** | 56.25 | 77.42 | **86.96** | **100.00** | 42.50 | **56.67** | 60.00 | 69.57 | **56.52** | **83.33** | **71.30** | **84.62** |
| Qwen2.5-72B-Instruct | 72.55 | **68.75** | **87.10** | **86.96** | 75.00 | 60.00 | 50.00 | **85.00** | **73.91** | 52.17 | 70.83 | 69.57 | **84.62** |
| Llama3.1-70B-Instruct | 56.86 | 56.25 | 51.61 | 80.43 | 50.00 | 65.00 | 30.00 | 70.00 | 58.70 | 47.83 | 75.00 | 51.30 | 76.92 |
| OmniSQL-32B | 62.75 | 31.25 | 61.29 | 54.35 | 25.00 | **72.50** | 33.33 | 60.00 | 36.96 | 52.17 | 70.83 | 41.74 | 38.46 |
| Qwen3-32B | 57.52 | 50.00 | 22.58 | 58.70 | **100.00** | 27.50 | 30.00 | 55.00 | 54.35 | 52.17 | 50.00 | 53.91 | 46.15 |
| ***Pass^3*** | | | | | | | | | | | | | |
| GPT-4o | 33.99 | 37.50 | 38.71 | 63.04 | 75.00 | 5.00 | 13.33 | 35.00 | 39.13 | 21.74 | 41.67 | 29.57 | **76.92** |
| DeepSeek-V3 | 36.60 | 18.75 | 16.13 | 36.96 | 25.00 | 17.50 | 10.00 | 15.00 | 15.22 | 21.74 | 20.83 | 22.61 | 53.85 |
| Gemini2.5-flash | 21.57 | 0.00 | 22.58 | 8.70 | 0.00 | 7.50 | 6.67 | 0.00 | 8.70 | 0.00 | 0.00 | 0.00 | 53.85 |
| Qwen2.5-Max | **56.86** | **43.75** | 54.84 | 60.87 | **100.00** | 25.00 | 23.33 | 55.00 | 41.30 | 30.43 | 50.00 | 40.87 | 69.23 |
| Qwen2.5-72B-Instruct | **56.86** | **43.75** | **64.52** | **71.74** | 75.00 | **45.00** | **30.00** | **65.00** | **54.35** | **43.48** | **58.33** | **52.17** | 69.23 |
| Llama3.1-70B-Instruct | 30.72 | 37.50 | 32.26 | 67.39 | 25.00 | 32.50 | 6.67 | 55.00 | 34.78 | 21.74 | 45.83 | 24.35 | **76.92** |
| OmniSQL-32B | 42.48 | 31.25 | 35.48 | 34.78 | 0.00 | 37.50 | 20.00 | 35.00 | 19.57 | 17.39 | 45.83 | 21.74 | 23.08 |
| Qwen3-32B | 30.07 | 25.00 | 3.23 | 28.26 | 50.00 | 7.50 | 3.33 | 35.00 | 19.57 | 26.09 | 20.83 | 30.43 | 46.15 |
| ***Pass^5*** | | | | | | | | | | | | | |
| GPT-4o | 23.53 | 31.25 | 25.81 | 41.30 | 25.00 | 2.50 | 6.67 | 25.00 | 23.91 | 8.70 | 33.33 | 17.39 | 61.54 |
| DeepSeek-V3 | 22.22 | 12.50 | 12.90 | 23.91 | 0.00 | 10.00 | 10.00 | 10.00 | 10.87 | 8.70 | 16.67 | 15.65 | 38.46 |
| Gemini2.5-flash | 8.50 | 0.00 | 9.68 | 0.00 | 0.00 | 5.00 | 3.33 | 0.00 | 6.52 | 0.00 | 0.00 | 0.00 | 46.15 |
| Qwen2.5-Max | 43.79 | **37.50** | 38.71 | 50.00 | **75.00** | 7.50 | 13.33 | 45.00 | **32.61** | 17.39 | 41.67 | 26.09 | 53.85 |
| Qwen2.5-72B-Instruct | **49.67** | 31.25 | **54.84** | **63.04** | 50.00 | **32.50** | **26.67** | **65.00** | **32.61** | **43.48** | **58.33** | **35.65** | **69.23** |
| Llama3.1-70B-Instruct | 16.34 | 31.25 | 29.03 | 47.83 | 25.00 | 22.50 | 6.67 | 45.00 | 21.74 | 8.70 | 37.50 | 15.65 | **69.23** |
| OmniSQL-32B | 30.72 | 31.25 | 25.81 | 23.91 | 0.00 | 22.50 | 13.33 | 25.00 | 13.04 | 13.04 | 25.00 | 9.57 | 23.08 |
| Qwen3-32B | 17.65 | 0.00 | 3.23 | 19.57 | 50.00 | 5.00 | 3.33 | 15.00 | 10.87 | 8.70 | 8.33 | 17.39 | 46.15 |




(a) DeepSeek-V3 (b) Llama-3.1-70B-Instruct (c) GPT-4o (d) Qwen2.5-72B-Instruct




(e) DeepSeek-V3 (f) Llama-3.1-70B-Instruct (g) GPT-4o (h) Qwen2.5-72B-Instruct

Figure 11: The number of dialogue turns of different models on the databases *car* (top row) and *retail_world* (bottom row).

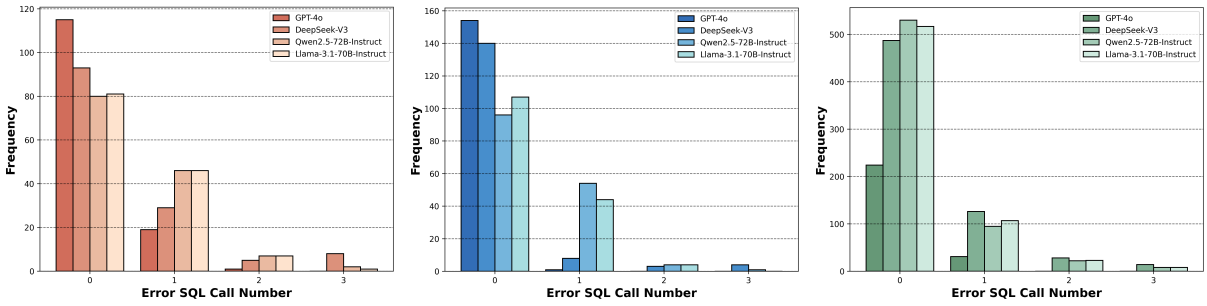
more effectively, avoiding unnecessary SQL calls while maintaining strong end-to-end accuracy.

Table 6: Long. Performance comparison of open-source and proprietary models on the Real-World-SQL-Bench benchmark.

| Model | Sports Domain | | | Entertainment Domain | | | | | | Business Domain | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ES | IH | BO | EN | MU | LE | CK | CH | PA | CA | HR | RE | RW |
| ***Pass^1*** | | | | | | | | | | | | | |
| GPT-4o | 59.68 | 33.33 | 71.25 | 64.71 | 64.71 | 38.46 | **38.10** | 66.67 | **71.19** | 50.00 | **68.75** | **48.86** | 72.22 |
| DeepSeek-V3 | 34.43 | 16.67 | 26.25 | 48.24 | 29.41 | 30.77 | 23.81 | 33.33 | 22.03 | 25.00 | 43.75 | 17.78 | 72.22 |
| Gemini2.5-flash | 35.48 | 8.33 | 38.75 | 50.59 | 11.76 | 15.38 | 9.52 | 33.33 | 10.17 | 25.00 | 6.25 | 0.00 | 22.22 |
| Qwen2.5-Max | 75.41 | 33.33 | **72.50** | **68.24** | 64.71 | 38.46 | **38.10** | **75.00** | 67.80 | **75.00** | 56.25 | 37.78 | **94.44** |
| Qwen2.5-72B-Instruct | **75.81** | **66.67** | 70.00 | 63.53 | 64.71 | **46.15** | 33.33 | 70.83 | 54.24 | **75.00** | 62.50 | 42.22 | **94.44** |
| Llama3.1-70B-Instruct | 56.45 | 8.33 | 45.00 | 57.65 | 41.18 | **46.15** | 28.57 | 66.67 | 37.29 | 50.00 | 62.50 | 25.56 | 72.22 |
| OmniSQL-32B | 43.55 | 33.33 | 28.75 | 43.53 | 29.41 | **46.15** | 47.62 | 45.83 | 22.03 | 50.00 | 43.75 | 17.78 | 55.56 |
| Qwen3-32B | 51.61 | 41.67 | 33.75 | 47.06 | **70.59** | 23.08 | 23.81 | 54.17 | 49.15 | **75.00** | 50.00 | 43.33 | 38.89 |
| ***Pass^3*** | | | | | | | | | | | | | |
| GPT-4o | 20.97 | 8.33 | 48.75 | 37.65 | 41.18 | 15.38 | 9.52 | 41.67 | 33.90 | 25.00 | 37.50 | **25.56** | 55.56 |
| DeepSeek-V3 | 16.13 | 8.33 | 5.00 | 20.00 | 0.00 | 15.38 | 4.76 | 4.17 | 8.47 | 0.00 | 31.25 | 7.78 | 27.78 |
| Gemini2.5-flash | 16.13 | 0.00 | 15.00 | 5.88 | 0.00 | 0.00 | 4.76 | 0.00 | 3.39 | 0.00 | 6.25 | 0.00 | 5.56 |
| Qwen2.5-Max | 41.94 | 16.67 | **52.50** | 40.00 | 23.53 | 23.08 | 9.52 | 54.17 | **49.15** | 25.00 | 43.75 | 20.00 | 72.22 |
| Qwen2.5-72B-Instruct | **45.16** | **25.00** | 50.00 | **47.06** | **52.94** | **38.46** | **28.57** | **58.33** | 28.81 | **50.00** | **50.00** | 21.11 | **94.44** |
| Llama3.1-70B-Instruct | 17.74 | 0.00 | 18.75 | 32.94 | 11.76 | 15.38 | 9.52 | 25.00 | 15.25 | 25.00 | **50.00** | 8.89 | 61.11 |
| OmniSQL-32B | 22.58 | 8.33 | 13.75 | 20.00 | 0.00 | 23.08 | 23.81 | 16.67 | 8.47 | 0.00 | 31.25 | 7.78 | 38.89 |
| Qwen3-32B | 16.13 | 16.67 | 3.75 | 15.29 | 29.41 | 7.69 | 4.76 | 25.00 | 15.25 | 0.00 | 18.75 | 15.56 | 5.56 |
| ***Pass^5*** | | | | | | | | | | | | | |
| GPT-4o | 9.68 | 0.00 | **37.50** | 28.24 | **35.29** | 0.00 | 4.76 | 29.17 | 18.64 | 25.00 | 25.00 | **15.56** | 55.56 |
| DeepSeek-V3 | 6.45 | 8.33 | 5.00 | 14.12 | 0.00 | 7.69 | 4.76 | 4.17 | 3.39 | 0.00 | 25.00 | 6.67 | 22.22 |
| Gemini2.5-flash | 3.23 | 0.00 | 7.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 3.39 | 0.00 | 6.25 | 0.00 | 0.00 |
| Qwen2.5-Max | 29.03 | 8.33 | 35.00 | 24.71 | 17.65 | 15.38 | 4.76 | 37.50 | **35.59** | 25.00 | **43.75** | 13.33 | 66.67 |
| Qwen2.5-72B-Instruct | **32.26** | **25.00** | 33.75 | **40.00** | **35.29** | **38.46** | 9.52 | **50.00** | 22.03 | **50.00** | **43.75** | 13.33 | **77.78** |
| Llama3.1-70B-Instruct | 11.29 | 0.00 | 12.50 | 24.71 | 5.88 | 15.38 | 9.52 | 20.83 | 8.47 | 0.00 | **43.75** | 5.56 | 55.56 |
| OmniSQL-32B | 9.68 | 8.33 | 7.50 | 15.29 | 0.00 | 7.69 | **14.29** | 4.17 | 5.08 | 0.00 | 31.25 | 5.56 | 38.89 |
| Qwen3-32B | 8.06 | 8.33 | 2.50 | 8.24 | 11.76 | 7.69 | 4.76 | 20.83 | 8.47 | 0.00 | 6.25 | 8.89 | 5.56 |

Table 7: ALL. Performance comparison of open-source and proprietary models on the Real-World-SQL-Bench benchmark.

| Model | Sports Domain | | | Entertainment Domain | | | | | | Business Domain | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ES | IH | BO | EN | MU | LE | CK | CH | PA | CA | HR | RE | RW |
| ***Pass^1*** | | | | | | | | | | | | | |
| GPT-4o | 62.79 | 46.43 | 70.27 | 70.23 | 66.67 | 30.19 | 33.33 | 68.18 | 65.71 | 51.85 | 62.50 | 52.20 | 77.42 |
| DeepSeek-V3 | 48.84 | 28.57 | 26.13 | 56.49 | 33.33 | 33.96 | 31.37 | 43.18 | 36.19 | 40.74 | 45.00 | 30.73 | 74.19 |
| Gemini2.5-flash | 48.37 | 14.29 | 41.44 | 51.91 | 14.29 | 15.09 | 21.57 | 34.09 | 13.33 | 22.22 | 15.00 | 0.98 | 35.48 |
| Qwen2.5-Max | **76.74** | 46.43 | 73.87 | **74.81** | 71.43 | 41.51 | **49.02** | 68.18 | **68.57** | **59.26** | **72.50** | 56.59 | **90.32** |
| Qwen2.5-72B-Instruct | 73.49 | **67.86** | **74.77** | 71.76 | 66.67 | 56.60 | 43.14 | **77.27** | 62.86 | 55.56 | 67.50 | **57.56** | **90.32** |
| Llama3.1-70B-Instruct | 56.74 | 35.71 | 46.85 | 65.65 | 42.86 | 60.38 | 29.41 | 68.18 | 46.67 | 48.15 | 70.00 | 40.00 | 74.19 |
| OmniSQL-32B | 57.21 | 32.14 | 37.84 | 47.33 | 28.57 | **66.04** | 39.22 | 52.27 | 28.57 | 51.85 | 60.00 | 31.22 | 48.39 |
| Qwen3-32B | 55.81 | 46.43 | 30.63 | 51.15 | **76.19** | 26.42 | 27.45 | 54.55 | 51.43 | 55.56 | 50.00 | 49.27 | 41.94 |
| ***Pass^3*** | | | | | | | | | | | | | |
| GPT-4o | 30.23 | 25.00 | 45.95 | 46.56 | 47.62 | 7.55 | 11.76 | 38.64 | 36.19 | 22.22 | 40.00 | 27.80 | 64.52 |
| DeepSeek-V3 | 30.70 | 14.29 | 8.11 | 25.95 | 4.76 | 16.98 | 7.84 | 9.09 | 11.43 | 18.52 | 25.00 | 16.10 | 38.71 |
| Gemini2.5-flash | 20.00 | 0.00 | 17.12 | 6.87 | 0.00 | 5.66 | 5.88 | 0.00 | 5.71 | 0.00 | 2.50 | 0.00 | 25.81 |
| Qwen2.5-Max | 52.56 | 32.14 | 53.15 | 47.33 | 38.10 | 24.53 | 17.65 | 54.55 | **45.71** | 29.63 | 47.50 | 31.71 | 70.97 |
| Qwen2.5-72B-Instruct | **53.49** | **35.71** | **54.05** | **55.73** | **57.14** | **43.40** | **29.41** | **61.36** | 40.00 | **44.44** | **55.00** | **38.54** | **83.87** |
| Llama3.1-70B-Instruct | 26.98 | 21.43 | 22.52 | 45.04 | 14.29 | 28.30 | 7.84 | 38.64 | 23.81 | 22.22 | 47.50 | 17.56 | 67.74 |
| OmniSQL-32B | 36.74 | 21.43 | 19.82 | 25.19 | 0.00 | 33.96 | 21.57 | 25.00 | 13.33 | 14.81 | 40.00 | 15.61 | 32.26 |
| Qwen3-32B | 26.05 | 21.43 | 3.60 | 19.85 | 33.33 | 7.55 | 3.92 | 29.55 | 17.14 | 22.22 | 20.00 | 23.90 | 22.58 |
| ***Pass^5*** | | | | | | | | | | | | | |
| GPT-4o | 19.53 | 17.86 | 34.23 | 32.82 | 33.33 | 1.89 | 5.88 | 27.27 | 20.95 | 11.11 | 30.00 | 16.59 | 58.06 |
| DeepSeek-V3 | 17.67 | 10.71 | 7.21 | 17.56 | 0.00 | 9.43 | 7.84 | 6.82 | 6.67 | 7.41 | 20.00 | 11.71 | 29.03 |
| Gemini2.5-flash | 6.98 | 0.00 | 8.11 | 0.00 | 0.00 | 3.77 | 1.96 | 0.00 | 4.76 | 0.00 | 2.50 | 0.00 | 19.35 |
| Qwen2.5-Max | 39.53 | 25.00 | 36.04 | 33.59 | 28.57 | 9.43 | 9.80 | 40.91 | **34.29** | 18.52 | 42.50 | 20.49 | 61.29 |
| Qwen2.5-72B-Instruct | **44.65** | **28.57** | **39.64** | **48.09** | **38.10** | **33.96** | **19.61** | **56.82** | 26.67 | **44.44** | **52.50** | **25.85** | **74.19** |
| Llama3.1-70B-Instruct | 14.88 | 17.86 | 17.12 | 32.82 | 9.52 | 20.75 | 7.84 | 31.82 | 14.29 | 7.41 | 40.00 | 11.22 | 61.29 |
| OmniSQL-32B | 24.65 | 21.43 | 12.61 | 18.32 | 0.00 | 18.87 | 13.73 | 13.64 | 8.57 | 11.11 | 27.50 | 7.80 | 32.26 |
| Qwen3-32B | 14.88 | 3.57 | 2.70 | 12.21 | 19.05 | 5.66 | 3.92 | 18.18 | 9.52 | 7.41 | 7.50 | 13.66 | 22.58 |




(a) *car*. (b) *retail_world*. (c) *entertainment*.

Figure 12: Comparison of the number of dialogue turns with failed SQL invocations for GPT-4o, DeepSeek-V3, Qwen2.5-72B-Instruct, and Llama-3.1-70B-Instruct on the *entertainment*, *car*, and *retail_world* databases.

## G Detailed Analysis of Hallucination in Multi-turn Tasks

During dynamic multi-turn interactions between the agent model and the user model, we found that the agent often exhibits hallucination after generating an SQL query, namely by fabricating the query results on its own. According to the taxonomy of hallucinations defined in prior work (Bang et al., 2025), this phenomenon falls under *Intrinsic Hallucination*. Specifically, the model fails to properly interpret the contextual constraints specified in the system prompt: after generating the SQL query, it directly fabricates the corresponding execution results instead of awaiting the environment's response in a subsequent interaction. Such hallucinations typically arise from the model's misinterpretation of the task requirements themselves. We manually inspected a subset of the model's dialogue trajectories and observed instances of *Intrin-*

*sic Hallucination*. To systematically identify such hallucinations, we check whether a model's dialogue trajectory output contains an SQL code block that is immediately followed by the special token <*result*>, which is used in the agent model's system prompt to denote the SQL execution result returned from the environment. As illustrated in Figure 7, DeepSeek-V3 and OmniSQL-32B demonstrate the highest hallucination rates, at 44.28% and 26.15%, respectively. The hallucination rates of Llama3.1-70B-Instruct, Qwen3-Coder-30B-A3B-Instruct, Qwen2.5-Max, and Gemini-2.5-Flash are approximately 10%, whereas GPT-4o yields the lowest hallucination rate of 0.09%. We hypothesize that the high hallucination rates observed in DeepSeek-V3 and OmniSQL-32B stem from the models' tendency to learn step-by-step problem-solving patterns during post-training. In particular, these models appear to internalize a reasoning-and-planning style in which intermediate results are explicitly produced after each step. When transferring this learned pattern to DySQL-Bench, they may incorrectly extend the same behavior by generating presumed query outputs immediately after producing SQL code, rather than waiting for the environment to return the execution result. We present an example in Figure 8. After receiving the user's instruction, the model first performs identity verification. However, after generating a query to retrieve the user's ID (*BowlerID*), it fabricates an incorrect result (*BowlerID = 14*), whereas the actual query output should be *BowlerID = 8*. This behavior demonstrates that the model produced a factually incorrect output due to hallucination.

## H Details of Human Expert Evaluation

To ensure the reliability and correctness of the benchmark, every task in the final dataset underwent rigorous human verification. Specifically, after passing the multi-stage automatic validation pipeline, all tasks were manually inspected by a Quality Assurance Board composed of ten domain experts with extensive experience in database management and SQL semantics.

Each expert independently reviewed the paired instruction–action tuples $\langle I, A \rangle$ to confirm three aspects: (1) the *semantic fidelity* between the user instruction and the corresponding SQL action; (2) the *structural validity* of the SQL syntax and its logical coherence with the database schema; and (3) the *executability and outcome correctness* within the simulated database environment.

Any disagreements among annotators were resolved through cross-review and consensus discussions, ensuring inter-annotator consistency. The committee reported a near-perfect agreement rate ($> 99.5\%$ Cohen's $\kappa$), and all discrepancies were manually corrected before release.

As a result, every one of the $1{,}072$ benchmark tasks has been verified to be semantically correct, executable, and free from hallucinated or ill-formed SQL queries. This full-scope human validation guarantees that the benchmark can serve as a high-fidelity, error-free foundation for evaluating interactive database reasoning and manipulation in large language models.

Database schema

```
Bowler
├── Bowlers_info
│   ├── BowlerID
│   ├── cust_first_name
│   ├── ...
│   ├── BowlerCurrentHcp
│   └── Teams_info
│       ├── TeamID
│       ├── TeamName
│       ├── ...
│       └── CaptainID
├── Match
│   ├── Bowler_Scores
│   │   ├── MatchID
│   │   ├── GameNumber
│   │   ├── BowlerID
│   │   ├── ...
│   │   └── WonGame
│   ├── Match_Games
│   │   ├── MatchID
│   │   ├── GameNumber
│   │   ├── ...
│   │   └── WinningTeamID
│   ├── Tournaments
│   │   └── ...
│   ├── Tourney_Matches
│   │   └── ...
└── ...
```

Figure 13: Database Schema Representation.

## I Database Schema Representation

The example *bowling* database schema is shown in Figure 13.

# J Prompt

## J.1 Logic Tree Construction Pipeline

The prompt is shown in figure 14.

## J.2 User System

The prompt is shown in figure 15.

## J.3 Task Generation Prompt

The prompt is shown in figure 16.

## J.4 Task Verification Prompt

The prompt is shown in figure 17.

## J.5 Task Refine Prompt

The prompt is shown in figure 18.

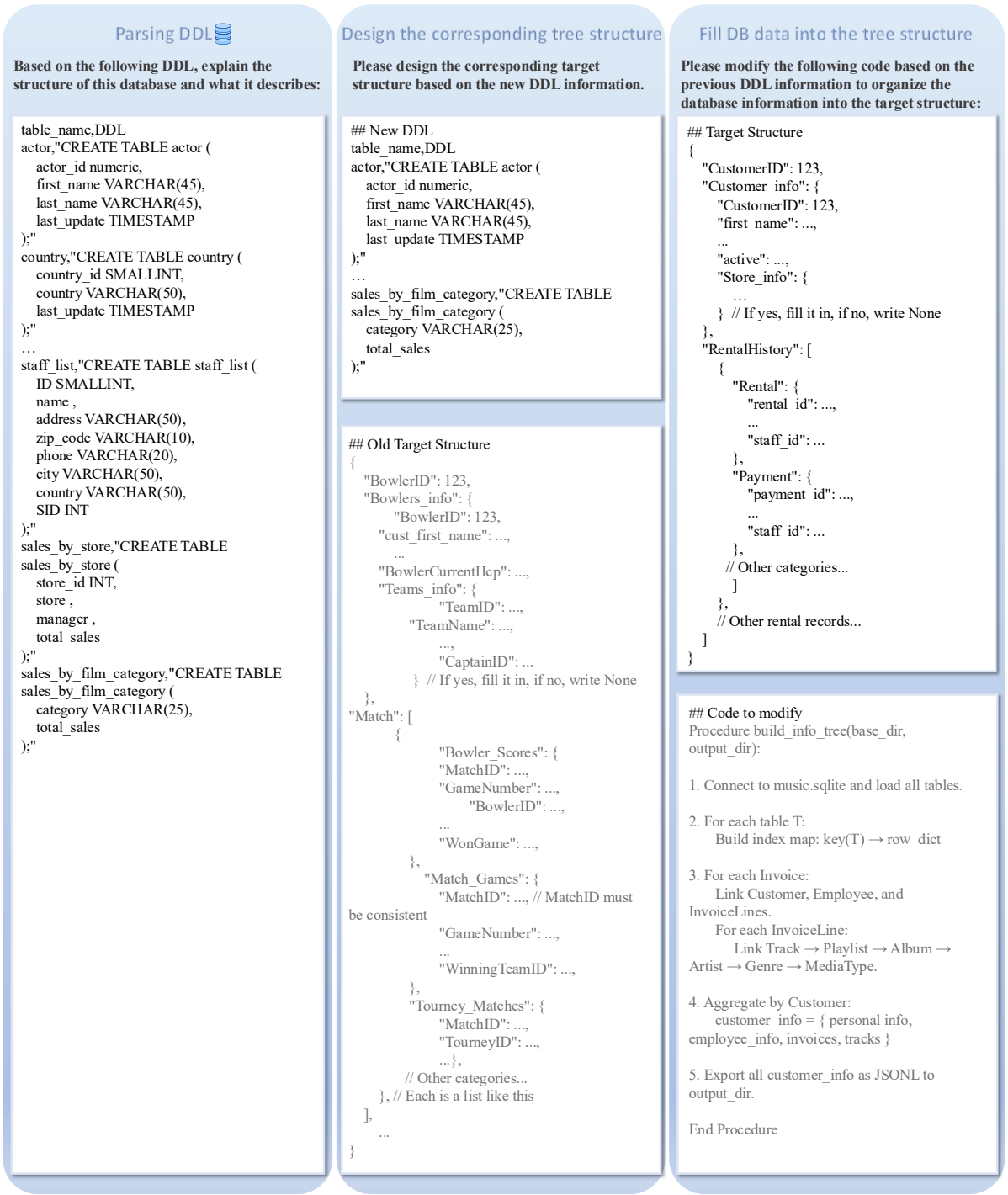



Figure 14: Transfer from raw database to logic tree structure.

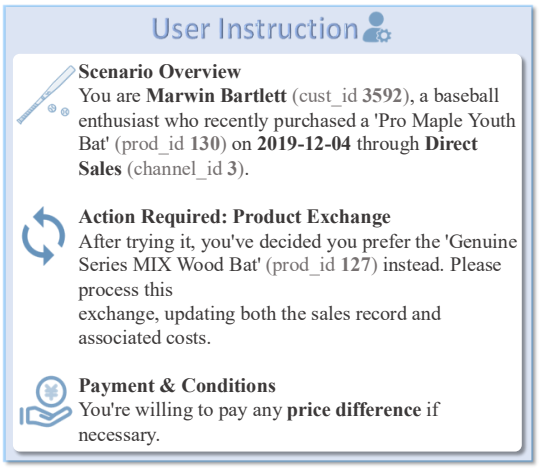



Figure 15: User model system prompt

## DySQL-Bench Task Generation Prompt

Generate a NEW task instruction that mimics realistic human users and their intentions, such as with different personality and goals. The task instruction should be followed by 'actions' which is a list of the sql to be taken to solve this task and 'outputs' which is a list of the answers to specific information requests made by the user. Think step by step to come up with the action(s) and the corresponding sql(s) translating this thought that would be necessary to fulfill the user's request or solve their intentions. The new user instruction should have all the parameters for the SQL calls in Actions.

**Guidelines for generating NEW taks instruction and Groundtruth Actions**
1. You must generate a new user instruction according to the <Input Database>.
2. The main focus is to generate actions that can modify the underlying database.
3. For actions that do not modify the database like specific information requests, scan the provided User Data directly and append only the answer in 'outputs'. **Do not make separate sql calls for this in 'actions'**.
4. Include multiple SQL calls when the scenario requires multiple steps or modifications.
5. Provide precise SQL calls with all necessary parameters for each action according to the given Dataset schema, and ALL the parameters should be Explicitly given in the new user instruction.

**Principles for generating SQL calls**
- At the beginning of the conversation, you have to authenticate the user identity by locating their user.
- Once the user has been authenticated, you can provide the user with information, e.g. help the user look up order id.
- You can only help one user per conversation (but you can handle multiple requests from the same user), and must deny any requests for tasks related to any other user.
- You **should not** make up any information or knowledge or procedures not provided from the user, or give subjective recommendations or comments.
- You should **at most make one sql call at a time**, and if you take a sql call, you should not respond to the user at the same time. If you respond to the user, you should not make a sql call.
- You should transfer the user to a human agent if and only if the request cannot be handled within the scope of your actions.

**Output Format**
Generate your response according to the following <Instruction Example> and <Format Example> format. Enclose the thought process within '<thought></thought>' tags, and the final structured **response within <answer></answer>** tags. The structured response should be in **strict JSON format,** without any additional comments or explanations.

**Instruction Example**
{example_instruction}

**Format Example (only for reference)**
{{"annotator": ..., "user_id": "...", "instruction": "...", "actions": [{{"sql": "..."}}, {{"sql": "..."}}, …], "outputs": []}}

The new user instruction must have all the parameters for the SQL calls in Actions. Do not directly copy instruction and the action patterns from the examples. Ground the generation from the above provided data.

Figure 16: Task generation prompt.

## DySQL-Bench Task Verification Prompt

**Verification Task Overview**
Please help me to **verify** whether the assistant has solved the user's problem based on the provided user and assistant interactions.
The user has outlined specific requirements, and the assistant's response should address all of these needs. The output should indicate whether the assistant has fully addressed the user's request, with a detailed check of the Agent Policy and assistant's sql call validity, correctness of invocation.

**Principles for Verification**
You have four principles to do this.
1. *[Verification]* The output should thoroughly verify whether the assistant's responses and tool calls have correctly addressed all of the user's requests step by step.
2. *[SQL Call Accuracy]* The output should check whether the assistant used the appropriate sql calls, with correct invocation **and parameters**, to solve the user's task.
3. *[Consistency Check]* The output should ensure that the data provided by the user is **consistent throughout the interaction**, without any discrepancies or hallucinations.
4. *[Correctness]* The verification should confirm if all of the user's requirements have been fully addressed and that no crucial aspect of the problem was overlooked.

**Response format**
The response should include reasoning process **step by step**, and ending with: "**Verification: Is the answer correct (Yes/No)?**" followed by "Yes" or "No".

Figure 17: Task verification prompt.

## DySQL-Bench Task Refine Prompt

REFINE the given task to confirm the the refined instruction have all the parameters in the SQL calls. Think step-by-step to improve the instruction and corresponding SQL queries.

**Guidelines for Refining Task Instruction and Groundtruth Actions**
1. Start with an existing task instruction and enhance its realism or clarity.
2. Focus on ensuring actions can properly modify the underlying database.
3. For non-modifying actions (e.g., information requests), scan the provided User Data directly and append only the answer in 'outputs'—**no separate tool calls needed in 'actions'.**
4. Adjust SQL calls if the refined scenario requires additional or modified steps.
5. Ensure SQL calls include all necessary parameters based on the Dataset schema, and **ALL parameters must be explicitly given** in the refined instruction.
6. Refined SQL parameters **MUST align** with the updated user instruction.

**Output Format**
Provide your response in the following format. Enclose the thought process within <thought></thought> tags, and the final structured response within '<answer></answer>' tags. The structured response should be in **strict JSON format**, without additional comments.

The refined instruction must include all parameters for the SQL calls in Actions.

Figure 18: Task refine prompt.