ROSE: An Intent-Centered Evaluation Metric for NL2SQL

Wenqi Pei[1,2,*], **Shizheng Hou**[2,*], **Boyan Li**[1,*], **Han Chen**[2], **Zhichao Shi**[1], **Yuyu Luo**[1,†]

[1] The Hong Kong University of Science and Technology (Guangzhou)
[2] National University of Singapore

Abstract

Execution Accuracy (EX), the widely used metric for evaluating the effectiveness of Natural Language to SQL (NL2SQL) solutions, is becoming increasingly unreliable. It is sensitive to syntactic variation, ignores that questions may admit multiple interpretations, and is easily misled by erroneous ground-truth SQL. To address this, we introduce **ROSE**, an intent-centered metric that focuses on whether the predicted SQL answers the question, rather than consistency with the ground-truth SQL under the reference-dependent paradigm. ROSE employs an adversarial Prover-Refuter cascade: SQL Prover assesses the semantic correctness of a predicted SQL against the user's intent independently, while Adversarial Refuter uses the ground-truth SQL as evidence to challenge and refine this judgment. On our expert-aligned validation set **ROSE-VEC**, ROSE achieves the best agreement with human experts, outperforming the next-best metric by nearly 24% in Cohen's Kappa. We also conduct a large-scale re-evaluation of 19 NL2SQL methods, revealing four valuable insights. We release ROSE and ROSE-VEC to facilitate more reliable NL2SQL research[1].

1 Introduction

Natural Language to SQL (NL2SQL) translates user questions into executable SQL queries over a given database (Luo et al., 2025; Liu et al., 2025a; Zhu et al., 2026) and supports a range of data analysis tasks (Zhang et al., 2020; Luo et al., 2018a,b). Recent progress in NL2SQL has been remarkable, driven by increasingly capable large language models (LLMs) (Li et al., 2026; Zhu et al., 2025; Zhang et al., 2025; Pei et al., 2025). However, this rapid growth has exposed a key bottleneck: existing evaluation metrics struggle to capture whether the predicted SQL is semantically correct with respect to the user's intent (Li et al., 2024a; Hou et al., 2026).

The core metric in the field, Execution Accuracy (EX), deems a predicted SQL correct only when its execution result matches a single ground-truth SQL. This surface-level check causes EX to be unreliable in three recurring cases: (i) Realization variance when the underlying logic is identical (e.g., projection order, value formatting). EX cannot accommodate such predictions across different output representations. Structural audits report false negatives up to 28.9% caused by non-canonical but correct forms (Ascoli et al., 2024); (ii) Multiple valid interpretations for ambiguous questions. EX misses these reasonable predictions that deviate from the ground-truth SQL but still correctly answer the question. Ambrosia (Saparina and Lapata, 2024) reports that more than half of failures arise from ambiguity, and Sphinteract (Zhao et al., 2025b) further underscores the severity of the problem; (iii) Erroneous ground-truth SQL in large-scale benchmarks. EX propagates these annotation errors to all evaluated predictions. NL2SQL-BUGs (Liu et al., 2025b) reports a 6.91% ground-truth SQL error rate on BIRD Dev. In our audit, approximately 25% of sampled items were flagged as wrong by at least one annotator. These issues indicate that EX favors matching a particular reference SQL rather than achieving the intent behind the question.

Several improved metrics have been proposed. Structure-aware measures (Ascoli et al., 2024; Zhan et al., 2025) allow for syntax-level variants via complex normalization, which reduces superficial mismatches, but still judges proximity to a reference rather than whether the prediction answers the question's semantics. LLM-based judges (Kim et al., 2025; Zhao et al., 2025a) improve alignment with expert assessments, but most protocols remain reference-dependent, checking consistency primarily against a single ground-truth SQL. They are insufficiently tolerant of legitimate ambiguity

[*] Equal Contribution.
[†] Corresponding author.
[1] https://github.com/CedricPei/ROSE

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 5682–5709
July 2-7, 2026

and are easily misled by the flawed ground-truth SQL. In light of these limitations, a paradigm shift toward less reference-dependent evaluation is becoming advisable.

Therefore, we introduce **ROSE** (**R**eas**O**ning **S**cor**E**). It is a new metric designed to measure the consistency between the underlying intent of the question and the reasoning process embodied by the prediction. This intent-centered evaluation is realized via a Prover-Refuter cascade powered by reasoning models. The *SQL Prover* makes an independent judgment using only the question and the database information, without accessing the ground-truth SQL. The *Adversarial Refuter*, with access to the ground-truth SQL, treats it as evidence to challenge the Prover's acceptance, contrasting it with the prediction to expose decisive mismatches. Through this analysis, it can also tag cases as ambiguous questions or ground-truth errors. This cascade reduces reference anchoring while tempering over-permissiveness and exposes dataset issues.

We construct **ROSE-VEC**, a **V**alidation dataset with **E**xpert **C**onsensus to assess metric validity. On this dataset, ROSE outperforms the closest competitor by 24% in agreement and 14% in accuracy, demonstrating better alignment with user intent.

Building on ROSE, we perform a large-scale re-evaluation of 19 NL2SQL methods. Our analysis yields four key insights: (i) Base model capability, not system-level engineering, is the primary performance driver; (ii) As models advance, a widening gap between semantic correctness and reference matching signals an evaluation crisis; (iii) This divergence largely stems from benchmark flaws, namely ground-truth errors and question ambiguities; and (iv) Fine-tuning narrows this gap by aligning models to a dataset's stylistic conventions.

We summarize our contributions as follows:

- We introduce ROSE, an intent-centered metric for NL2SQL evaluation that leverages an adversarial Prover-Refuter cascade, achieving the best agreement with expert judgment.
- We construct and release ROSE-VEC, a validation dataset of 585 expert-consensus samples, complete with detailed annotations, to enable rigorous validation of NL2SQL metrics.
- We conduct a large-scale re-evaluation of 19 methods, distilling four key insights that provide important guidance for future NL2SQL research.

2 Preliminary

2.1 Problem Formulation

The task of NL2SQL is to translate a NL question Q into SQL S, conditioned on a given database and its content, collectively denoted D. Given Q and D, an NL2SQL method generates a predicted SQL S_p. For evaluation, benchmark datasets provide a corresponding ground-truth SQL S_g. Executing these queries against the database D yields their respective result sets E_p and E_g.

After S_p passes syntactic checks, we evaluate semantic correctness under a set of acceptance criteria C that encode user-specific rules on schema validity and alignment with Q (e.g., tolerance for duplicates or NULL). Any executable S_p that violates C is considered incorrect.

2.2 Ideal Evaluation

The ideal evaluation of a predicted SQL S_p assesses whether it correctly captures the user's intent. We formalize this with a theoretical judgment function, I, which decomposes the problem into two conditions: syntactic validity and semantic correctness.

$$I(Q, S_p, D, C) = \sigma_{\text{syn}}(S_p \,|D) \;\wedge\; \sigma_{\text{sem}}(S_p, Q \,|C)$$

Here, σ_{syn} verifies the query's syntactic validity against the database D, while σ_{sem} assesses if the query's logic semantically captures the intent of the question Q according to the criteria C.

While σ_{syn} is trivial to implement, a perfect σ_{sem} is computationally infeasible (Abiteboul et al., 1995; He et al., 2024). Practical methods approximate this by measuring the similarity of S_p and S_g through structural or execution result comparison.

3 Related Work

The evaluation of NL2SQL has evolved significantly. We categorize the landscape of evaluation metrics into two primary types: deterministic metrics and LLM-based metrics. The formal mathematical definitions for the metrics are available in the Appendix A.

3.1 Deterministic Metrics

Deterministic metrics evaluate query correctness using predefined, rule-based algorithms, forming the foundation of standard NL2SQL benchmarks.

Exact Match (EM) (Yu et al., 2018), also known as String Match (SM), is the strictest metric, requiring S_p to be character-for-character identical to

S_g after normalization. To provide partial credit, **Component Match (CM)** (Yu et al., 2018) evaluates correctness at the clause level. It scores S_p by calculating the proportion of its components that correctly match S_g.

Execution Accuracy (EX) (Yu et al., 2018) verifies the equivalence of execution results. S_p is deemed correct if it produces the same result as S_g. **Enhanced Tree Match (ETM)** (Ascoli et al., 2024) operates on a structural level. It parses S_p and S_g into Abstract Syntax Trees (ASTs). A match is declared when their normalized forms are structurally equivalent. Other approaches include **Exact Set Matching (ESM)** (Yu et al., 2018), which compares unordered sets of keywords and arguments, and **Distilled Test Suite** (Zhong et al., 2020) designed to probe specific SQL capabilities.

The reliance of deterministic metrics on a single and sometimes flawed S_g penalizes valid alternatives and misrepresents method performance, thus necessitating more flexible evaluation metrics.

3.2 LLM-based Metrics

A new evaluation type has emerged that employs Large Language Models (LLMs) as semantic judges to assess query correctness. Unlike deterministic metrics, this approach aims to leverage the reasoning capabilities of LLMs to provide a more realistic measure of utility.

LLM-SQL-Solver (Zhao et al., 2025a) directly prompts an LLM to determine whether a predicted SQL is equivalent to S_g. It employs prompting strategies such as Miniature & Mull to find counterexamples and Explain & Compare to analyze logic. To reduce false positives and negatives of EX, **FLEX** (Kim et al., 2025) leverages meticulously crafted prompts that provide complete context, guiding an LLM to deliver adequacy judgments. Frameworks from industry such as Defog.ai and Arize also use LLMs for evaluation (Defog.ai, 2023; Arize AI, 2024).

However, existing LLM-based metrics still rely on S_g, which can be misleading and unfairly penalize valid alternatives. Our work addresses this by transforming the role of S_g from a standard reference into an adversarial challenge.

4 Methodology

4.1 SQL Prover

To overcome the limitations of reference-dependent metrics, a mechanism is required to validate the reasoning of a predicted SQL against the user's intent without reliance on S_g. To this end, we introduce SQL Prover, designed to assess the semantic correctness of a query independently. It is invoked only when S_p is syntactically valid and E_p differs from E_g. It evaluates whether S_p satisfies the user's intent expressed in Q under the acceptance criteria C. Formally, SQL Prover function P outputs a boolean judgment j_p and a rationale R:

$$Pro(Q, S_p, E_p \,|D, C) \rightarrow (j_p, R)$$

Detailed acceptance criteria and instructions are provided in Appendix O.1.

4.2 Adversarial Refuter

However, SQL Prover's complete independence from the ground-truth risks being overly permissive, failing to leverage signals (albeit noisy) within S_g. Hence, we introduce Adversarial Refuter. It uses S_g not as a reference to match, but as a source of evidence to challenge and potentially refute SQL Prover's affirmative judgment on S_p.

4.2.1 $E_p \equiv E_g$: Suppressing False Positives

When execution results match, SQL Prover is bypassed, and the Refuter acts as a critical safeguard. By directly comparing the reasoning of S_p and S_g, it identifies cases of coincidental correctness. This prevents false positives from a flawed prediction and, just as importantly, labeling erroneous S_g.

4.2.2 $E_p \not\equiv E_g$: Challenging SQL Prover

In cases where SQL Prover approves S_p despite E_p conflicting with E_g, Adversarial Refuter arbitrates the conflict. It pinpoints the semantic divergence between the reasoning of S_p and S_g. It then re-evaluates this divergence against the user's intent in Q to determine which logic is more faithful. This arbitration may refute SQL Prover's approval, flag S_g as erroneous, or accept both queries as valid interpretations of an ambiguous question Q.

Adversarial Refuter's function Ref produces a boolean judgment j_r (overturn for true, uphold for false) on S_P and a diagnostic label L_q for Q. The function is defined as follows:

$$Ref(Q, S_p, S_g; E_p^*, E_g^*, R^* \,|D, C) \rightarrow (j_r, L_q)$$

The conditional arguments (E_p^*, E_g^*, R^*) are invoked only when $E_p \not\equiv E_g$.

4.3 ROSE

ROSE is determined by the workflow illustrated in Figure 1. A predicted SQL must first be executable.

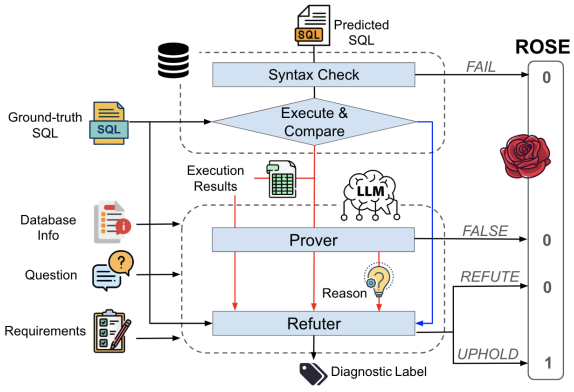

Figure 1: ROSE Scoring Cascade. The red path indicates the workflow when execution results do not match, while the blue path is followed when they do.

If its execution result differs from the ground-truth SQL, it must pass SQL Prover's independent evaluation. Finally, it must withstand the adversarial challenge from Adversarial Refuter, which uses the ground-truth as counter-evidence. Failure at any of these stages results in a score of 0. Only a query that successfully navigates this entire cascade earns a final score of 1. The mathematical definition for ROSE is in Appendix B and its comparison with other NL2SQL metrics is in Appendix C.

5 Experiments: Validating ROSE

5.1 Setup

5.1.1 ROSE-VEC

To enable rigorous validation of NL2SQL metrics against expert judgments, we construct a human-labeled dataset ROSE-VEC consisting of 585 NL-SQL pairs. It includes 263 pairs from outputs of DAIL-SQL (Gao et al., 2024), RSL-SQL (Cao et al., 2024), and Super-SQL (Li et al., 2024a) on Spider Test (Yu et al., 2018) and 322 pairs from outputs of TA-SQL (Qu et al., 2024), CSC-SQL (Sheng and Xu, 2025), OpenSearch-SQL (Xie et al., 2025), Alpha-SQL (Li et al., 2025a), and RSL-SQL on BIRD Dev. Each output is independently judged by two of the five experts using our evaluation interface, and we retain only cases with exact agreement. The resulting dataset stores instances (Q, D, S_g, S_p, Y), where Y denotes the consensus semantic correctness of S_p for Q. Details of the annotators are in Appendix M, and the evaluation interface is shown in Appendix N. ROSE-VEC and the interface constitute valuable resources for follow-up studies.

5.1.2 Validation Metrics

We evaluate agreement between NL2SQL metrics and expert labels, treating "correct" as the positive class. We report four validation metrics, with formulations deferred to Appendix D.

Cohen's Kappa (κ) (Cohen, 1960) is our primary metric. It measures agreement beyond chance and is robust under skewed distributions.

Accuracy (ACC) is the proportion of correctly judged instances. It is simple to interpret but can be misleading when the classes are imbalanced.

Matthews Correlation Coefficient (MCC) (Matthews, 1975) is a correlation coefficient between ground-truth and predicted labels that is reliable when class sizes differ substantially.

F1 (van Rijsbergen, 1979) is the harmonic mean of precision and recall on the positive class, assessing how well a judge identifies valid SQL queries.

5.1.3 NL2SQL Metrics

We consider two categories of NL2SQL metrics. Deterministic metrics include EX, EM, and ETM. LLM-based metrics include LLM-SQL-Solver, FLEX, and our proposed ROSE. We also report an ablation baseline ROSE w/o Refuter, which drops the Refuter and uses only the SQL Prover. More ablation studies are provided in Appendix F.

5.1.4 Reasoning Models

Following the naming convention in Appendix E.1, we instantiate three reasoning models for LLM-based metrics: closed-source OpenAI o3-2504 (OpenAI, 2025) and Google Gemini-2.5 Pro-2506 (Google DeepMind, 2025), and open-source DeepSeek-R1-2505 (Guo et al., 2025). Results on more open-source backbones are in Appendix G.

5.2 Results

5.2.1 Metric Effectiveness

Table 1 reveals a significant gap between deterministic metrics and expert judgment. EX achieves only 25.56% in Cohen's κ, confirming that it is a poor proxy for semantic correctness.

Though existing LLM-based metrics offer an improvement, they still fall short of expert-level agreement. Our ground-truth independent ablation ROSE w/o Refuter already surpasses them. This reflects the limitations of reliance on the ground-truth SQL, as discussed in Section 3.2.

With three different backbones, ROSE consistently achieves the best performance across vali-

Table 1: Performance of NL2SQL metrics on ROSE-VEC. Within each block, **Bold** indicates the best metric, Underline indicates the second best. Open-source models are represented with 🔓, while closed-source models are represented with 🔒. Separate results on the BIRD and Spider splits are in Appendix J.1.

Backbone	Metric	κ (%)	Acc (%)	MCC (%)	F1 (%)
Deterministic	EM	0.51	27.86	5.07	1.86
	ETM	6.60	35.56	18.47	20.63
	EX	**25.56**	**55.90**	**37.23**	**57.00**
OpenAI o3 🔒	LLM-SQL-Solver	12.24	42.05	25.54	33.92
	FLEX	56.70	78.97	62.01	83.31
	ROSE w/o Refuter	60.74	85.47	61.46	90.40
	ROSE	**80.43**	**91.79**	**81.04**	**94.16**
Gemini-2.5 Pro 🔒	LLM-SQL-Solver	20.31	50.43	33.62	48.40
	FLEX	43.53	70.60	51.16	75.14
	ROSE w/o Refuter	48.28	82.39	51.68	88.84
	ROSE	**69.68**	**86.84**	**71.01**	**90.41**
DeepSeek-R1 🔓	LLM-SQL-Solver	15.00	46.32	26.04	42.91
	FLEX	29.09	59.32	39.64	61.86
	ROSE w/o Refuter	46.90	79.15	46.90	85.75
	ROSE	**64.49**	**84.62**	**65.68**	**88.81**

Table 2: Precision of ROSE diagnostic labelling for GoldX and AmbQ on ROSE-VEC. Separate results on the BIRD and Spider splits are in Appendix J.2.

Label	Model	Count	Precision (%)
GoldX	OpenAI o3	51	**84.32**
	Gemini-2.5 Pro	76	68.42
	DeepSeek-R1	76	61.84
AmbQ	OpenAI o3	57	**91.23**
	Gemini-2.5 Pro	63	73.02
	DeepSeek-R1	108	51.85

dation metrics. ROSE$_{o3\text{-}2504}$ reaches 80.43% in Cohen's κ, 24% improvement over FLEX$_{o3\text{-}2504}$. This demonstrates the superiority of our cascade. The substantial gain from ROSE w/o Refuter to the full cascade suggests that adversarial use of the ground-truth is key to peak performance. We analyze representative error cases in Appendix I.

5.2.2 Diagnostic Effectiveness

Beyond scoring, ROSE serves as an effective diagnostic tool for identifying erroneous ground-truth SQL and ambiguous questions. To validate this capability, we manually verified the questions flagged by Refuter as having erroneous ground-truth SQL (GoldX) or being ambiguous (AmbQ) and calculated the precision of these labels.

As detailed in Table 2, ROSE$_{o3\text{-}2504}$ demonstrates high reliability. It achieves a precision of 84.32% for identifying ground-truth SQL errors and 91.23% for ambiguous questions. Although precision varies across different backbones, the strong performance of ROSE$_{o3\text{-}2504}$ confirms that its diagnostic labels are reliable enough for automated dataset analysis and cleaning, highlighting its dual value as both a superior evaluation metric and a powerful benchmark-auditing tool.

5.3 Efficiency and Versioning

We optimize the time efficiency of ROSE through multi-thread parallelism in Appendix H.1. To reduce monetary cost, we utilize concise prompts and route calls across stages, observing that ROSE can be more cost-efficient than FLEX, as shown in Appendix H.2. In addition, to maintain long-term stability under rapidly evolving LLMs, we adopt a backbone versioning strategy with selection and update policy detailed in Appendix E.2.

6 Experiments: Benchmarking

We conduct a comprehensive re-evaluation on a representative set of NL2SQL methods on two metrics: EX and ROSE$_{o3\text{-}2504}$. ROSE in subsequent paragraphs all refers to ROSE$_{o3\text{-}2504}$.

Table 3: Benchmarking results of NL2SQL methods on BIRD mini-Dev with EX and ROSE$_{o3\text{-}2504}$. Underlined methods denote base models. The score gap is illustrated in Appendix L.

Method	Date	Model	Simple		Moderate		Challenge		Overall	
			EX	ROSE	EX	ROSE	EX	ROSE	EX	ROSE
Prompting Methods										
GPT-5	Aug 2025	–	69.86	91.10	51.03	87.24	46.46	89.90	55.74	88.93
Alpha-SQL-32B	Feb 2025	Qwen2.5-Coder	81.02	91.24	69.47	79.65	52.58	70.10	69.35	81.09
OpenSearch-SQL	Sep 2024	DeepSeek-Chat	77.94	91.18	69.78	80.44	56.25	67.71	69.37	80.96
RSL-SQL	Oct 2024	DeepSeek-Chat	74.29	86.43	59.83	71.79	52.04	62.24	62.50	74.15
DeepSeek-Chat	Mar 2025	–	72.60	84.25	45.93	69.92	37.62	60.40	52.13	72.21
RSL-SQL	Oct 2024	GPT-4o	80.15	91.18	64.60	80.09	53.61	73.20	66.88	81.92
GPT-4o	May 2025	–	68.92	79.45	48.40	69.01	60.40	65.35	52.20	71.37
SuperSQL	Jul 2024	GPT-4	67.65	72.79	48.66	69.38	45.83	48.96	53.73	61.18
TA-SQL	May 2024	GPT-4	66.67	71.74	49.15	54.66	33.67	44.90	51.06	57.63
DAIL-SQL	Nov 2023	GPT-4	62.50	72.06	44.64	52.68	38.95	38.95	48.79	55.60
GPT-4	Apr 2024	–	65.75	76.03	44.03	67.90	36.63	49.50	48.98	66.53
CoT	Mar 2023	GPT-3.5	42.65	50.00	21.05	28.07	13.54	19.79	25.87	32.83
C3-SQL	Jul 2023	GPT-3.5	61.03	62.50	36.89	43.11	28.87	30.93	42.36	46.29
GPT-3.5	Jan 2024	–	56.16	68.49	36.21	44.03	31.68	40.59	41.22	50.61
Fine-tuned Methods										
CSC-SQL-32B	May 2025	XiYan-Qwen2.5	85.00	87.14	71.19	77.54	53.06	67.35	71.52	78.27
OmniSQL-32B	Mar 2025	Qwen2.5-Coder	80.00	86.43	67.23	78.30	57.14	72.45	68.92	79.49
CodeS-15B	Feb 2024	StarCoder	64.96	67.15	46.90	44.69	39.13	32.61	50.77	49.01
CHESS$_{(IR,SS,CG)}$	May 2024	DeepSeek-Coder	70.50	70.50	58.23	62.87	45.92	48.98	59.28	62.24
RESDSQL-3B	Feb 2023	T5	56.30	54.07	31.44	31.00	17.71	15.62	35.87	34.57

6.1 Setup

6.1.1 Dataset

BIRD Mini-Dev is used for evaluation. The split contains 500 NL-SQL pairs from 11 databases. Data remain unchanged compared with BIRD Dev.

6.1.2 NL2SQL Methods

Our evaluation considers both prompting methods and fine-tuned methods in NL2SQL. We prioritize systems with publicly accessible prediction outputs on the BIRD leaderboard[2].

Prompting Methods. This category utilizes LLMs via prompt engineering, forgoing task-specific fine-tuning. It can be divided into two sub-categories:

- *Base Models*. We use several LLMs in a zero-shot setting directly. These include OpenAI GPT series (GPT-5, GPT-4o, GPT-4, and GPT-3.5) and DeepSeek-Chat.
- *Engineered Systems*. This consists of systems that employ multi-step pipelines to decompose the task into sub-problems. The evaluated systems include Alpha-SQL-32B, OpenSearch-SQL, RSL-SQL, SuperSQL, TA-SQL, DAIL-SQL, C3-SQL (Dong et al., 2023), and CoT (Li et al., 2023b).

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

Fine-tuned Methods. This category comprises models that are specifically fine-tuned on NL2SQL corpora to specialize their capabilities. The included methods are CSC-SQL, OmniSQL-32B (Li et al., 2025b), CodeS-15B (Li et al., 2024b), CHESS$_{(IR,SS,CG)}$ (Talaei et al., 2024), and RESDSQL-3B (Li et al., 2023a).

Details of the engineered systems and fine-tuned methods are in Appendix K.

6.2 Insights

6.2.1 Base Model Dominance

We reveal a foundational principle:

NL2SQL performance advancement relies largely on the capability of base models rather than engineering designs.

This hierarchy is visually evident in Figure 2, where systems cluster into distinct performance tiers defined by their base model. Performance scales monotonically with model generation, from GPT-3.5, through GPT-4 and GPT-4o, to GPT-5, and engineered systems inherit this ordering. For example, RSL-SQL based on GPT-4o outperforms C3-SQL based on GPT-3.5 in both metrics. This pattern also holds for fine-tuned models: at a sim-

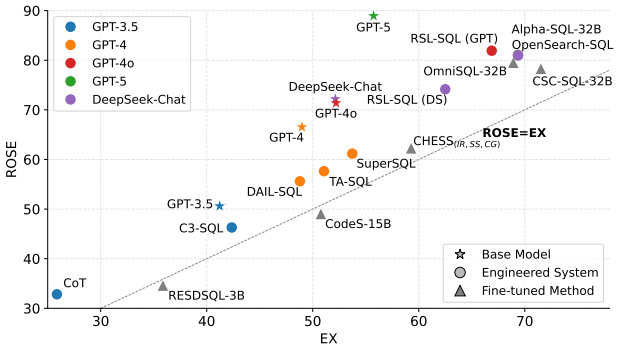

Figure 2: ROSE vs. EX scores for NL2SQL methods grouped by base model.

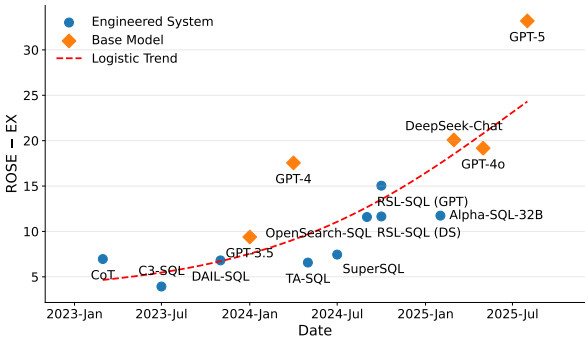

Figure 3: The growing divergence between ROSE and EX over time for prompting-based systems.

ilar 30B scale, OmniSQL built on Qwen2.5 surpasses CHESS$_{(IR,SS,CG)}$ on DeepSeek-Coder.

We also note that several systems trail their base models (e.g., GPT-4 surpasses DAIL-SQL). This discrepancy arises because our baselines use the latest model, while the systems' public results were generated with earlier versions. This reinforces our finding that recent performance gains are overwhelmingly attributable to stronger base models. It is therefore imperative that future evaluations decouple system design from model updates to accurately attribute true algorithmic improvements beyond model scaling.

6.2.2 Widening Gap

Building on the principle, our analysis uncovers a critical paradox:

As models become more powerful, the standard metric (EX) serves as an increasingly unreliable indicator of semantic correctness.

As illustrated in Figure 3, the gap between ROSE (intent-centered) and EX (reference-dependent) for prompting methods widens dramatically over time. The trend is stark: the gap was less than 5% with systems from mid-2023, but balloons to more than 20% with models projected for mid-2025.

This growing divergence is driven by two compounding factors. First, the nature of errors changes. Early systems often produced semantically incorrect SQL, where both metrics would agree on the failure, keeping the gap small. As methods advance, it is increasingly common to generate SQL that is semantically correct yet flagged wrong by EX due to its shortcomings or erroneous ground-truth SQL. ROSE gives correct judgment for valid solutions in such cases, thus widening the gap. Second, stronger models exhibit greater expressive freedom, generating a wider variety of semantically correct but stylistically diverse SQL. ROSE rewards this semantic competence, while EX punishes it, inflating the difference.

This trend delivers a verdict: the community is facing a metric crisis. The widening gap is not merely a statistical artifact, but a clear signal that rigid reference-matching evaluations are becoming obsolete in the era of powerful generative models. This points to the urgent need for an intent-centered metric that can measure true progress in the field.

6.2.3 Benchmark Flaws

The metric divergence largely stems from benchmark flaws, namely incorrect ground-truth SQL and ambiguous questions.

To pinpoint these factors, we use ROSE's diagnostic labels to isolate the discordance rate on these problematic subsets. The evidence presented in Table 4 is conclusive. For questions with erroneous ground-truth SQL (GoldX), the disagreement rate between the two metrics skyrockets to over 80% across all tested systems. For ambiguous questions (AmbQ), it remains exceptionally high at around 60%. Both rates far eclipse the average disagreement of less than 20% in the overall dataset, confirming that these dataset issues are a significant catalyst for metric divergence.

Table 4: Discordance rate (%) between EX and ROSE on GoldX, AmbQ.

Method	GoldX	AmbQ	GoldX ∪ AmbQ	Avg.
OpenSearch-SQL	86.00	58.33	77.14	20.35
Alpha-SQL-32B	90.00	47.62	76.12	19.13
RSL-SQL	84.62	68.18	77.94	18.08
OmniSQL-32B	82.61	60.00	74.60	18.18

Figure 4 shows that these two culprits are the dominant source of disagreements. GoldX and AmbQ consistently account for roughly 45% and

10% of discordant cases, collectively explaining more than half of instances where EX and ROSE diverge. Thus, improving the clarity and correctness of future datasets is essential for making EX a faithful proxy for user intent. This also underscores the critical need to identify benchmark flaws.

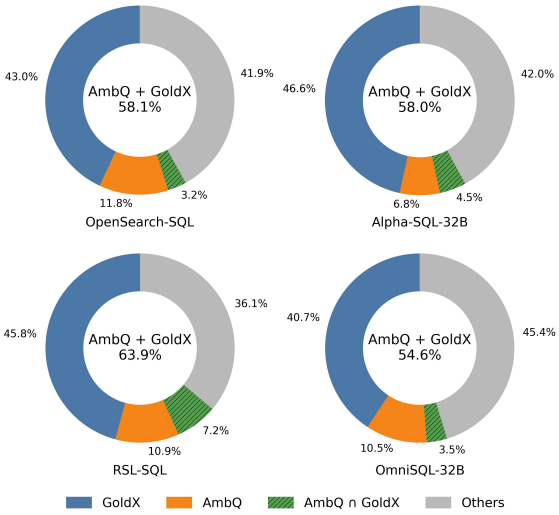

Figure 4: Share of discordant cases by question type.

6.2.4 Fine-Tuning Effect

In contrast, fine-tuning appears to alleviate the widening gap. Our analysis reveals that this is an alignment effect:

Fine-tuning narrows the metric gap by compelling models to mimic stylistic conventions of datasets.

This phenomenon is quantified in Figure 5, which shows that prompting methods consistently exhibit a larger gap across all difficulty levels. The difference is particularly obvious on Simple and Moderate questions, where the gap for prompting is roughly four times that of fine-tuned methods.

The underlying cause is that most fine-tuned methods are trained on BIRD Train, effectively learning to mimic the SQL style. This is reinforced by the outlier: OmniSQL, primarily trained on a huge corpus (SynSQL), shows a larger metric gap. Conversely, methods such as CodeS and CHESS$_{(IR,SS,CG)}$, which can potentially overfit noisy or limited training data, can even achieve higher EX than ROSE.

In contrast, prompting methods lack this dataset-specific conditioning. They generate a wider diversity of semantically correct, yet stylistically varied, SQL. While ROSE correctly credits this diversity, EX's rigid reference matching penalizes these variations, which explains the consistently larger gap observed for this class of methods.

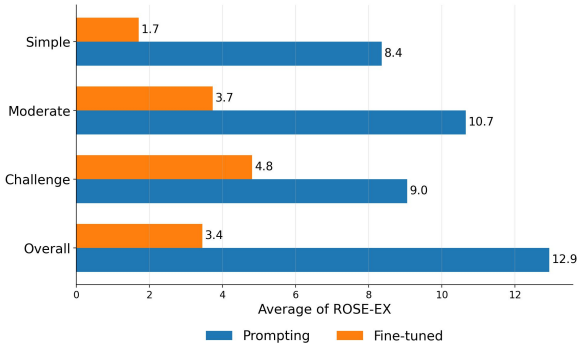

Figure 5: Average gap between ROSE and EX across difficulty levels for prompting and fine-tuned methods.

7 Conclusion

In this work, we addressed the growing crisis in NL2SQL evaluation by introducing ROSE, an intent-centered metric that leverages an adversarial Prover-Refuter cascade to achieve superior alignment with expert judgment. We demonstrated its effectiveness on ROSE-VEC, our publicly released dataset of 585 expert-annotated samples, where it substantially outperforms existing metrics.

Our re-evaluation yielded four insights for the community. First, the dominance of base models over system engineering calls for a re-evaluation of what constitutes a novel algorithmic contribution. Second, the widening gap between ROSE and EX serves as an urgent call to move beyond reference-dependent evaluation. Third, we identified that this gap is primarily caused by addressable benchmark flaws, namely erroneous ground-truth SQL and question ambiguity. Finally, our analysis suggests that fine-tuning on the current training set may be teaching stylistic conformity rather than improving semantic reasoning.

These insights highlight the urgent need for the community to prioritize the development of more robust and intent-centered evaluation metrics and to focus on improving the quality and clarity of datasets. The paradigm pioneered by ROSE offers a promising path forward, not only for NL2SQL, but potentially for other complex tasks where success should be defined by fulfilling user intent rather than matching a single, potentially flawed ground-truth solution. To truly measure and guide progress in the era of powerful generative models, we must evolve our methods of evaluation.

Limitations

While ROSE introduces a more robust, intent-centered evaluation paradigm for NL2SQL, we acknowledge several limitations that warrant consideration for future work.

Dependency on Foundational LLMs. The efficacy of ROSE is intrinsically linked to the reasoning capabilities of its underlying LLM. As demonstrated in Table 1, performance varies when different foundational models are used as the backbone for SQL Prover and Adversarial Refuter. Consequently, the reliability of ROSE as a metric may fluctuate as new models are released. Although such updates create an opportunity for stronger judges that align more closely with expert judgments, they also introduce a risk of drift across versions. Under our version management strategy in Appendix E, new models are treated as candidates that must pass re-validation before adoption.

Selection Bias from Consensus Filtering. ROSE-VEC is constructed by retaining only cases on which two annotators reach exact agreement. This reduces label noise and yields a high-confidence validation set, but it may also introduce a selection bias. In particular, retained instances may over-represent queries with clearer interpretation or easier semantic judgments, while under-representing borderline, disagreement-prone, or genuinely ambiguous cases. As a result, the reported agreement between automatic metrics and expert labels may not fully reflect performance on the broader distribution of NL2SQL outputs.

Computational Cost and Latency. Compared to deterministic metrics such as Execution Accuracy (EX), LLM-based metrics like ROSE are substantially more resource-intensive. The computational overhead and increased latency associated with model inference may limit the practicality of using ROSE in scenarios requiring rapid, iterative method development and testing, where immediate feedback is crucial. ROSE mitigates this overhead through call routing, concise prompts, and multi-thread parallelism, but it remains more expensive than deterministic alternatives.

Acknowledgements

This paper was supported by the NSF of China (62402409); Youth S&T Talent Support Programme of Guangdong Provincial Association for Science and Technology (SKXRC2025461); the Young Talent Support Project of Guangzhou Association for Science and Technology (QT-2025-001); Guangzhou Basic and Applied Basic Research Foundation (2026A1515010269, 2025A04J3935, 2023A1515110545); and Guangzhou-HKUST(GZ) Joint Funding Program (2025A03J3714).

References

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*.

Arize AI. 2024. Text to SQL: Evaluating SQL Generation with LLM as a Judge.

Benjamin G. Ascoli, Yasoda Sai Ram Kandikonda, and Jinho D. Choi. 2024. ETM: Modern Insights into Perspective on Text-to-SQL Evaluation in the Age of Large Language Models. *Preprint*, arXiv:2407.07313.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. *arXiv preprint arXiv:2411.00073*.

Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46.

Defog.ai. 2023. Introducing SQL-Eval: A Framework for Evaluating LLM-Generated SQL.

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. *Preprint*, arXiv: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. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Google DeepMind. 2025. Gemini 2.5: Our most intelligent AI model.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, and Others. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633638.

Yang He, Pinhan Zhao, Xinyu Wang, and Yuepeng Wang. 2024. VeriEQL: Bounded Equivalence Verification for Complex SQL Queries with Integrity Constraints. *Proceedings of the ACM on Programming Languages*, 8:1071–1099.

Shizheng Hou, Wenqi Pei, Nuo Chen, Quang-Trung Ta, Peng Lu, and Beng Chin Ooi. 2026. NL2SQL-Bench: A Modular Benchmarking Framework for LLM-Enabled NL2SQL Solutions. *Proceedings of the VLDB Endowment*, 19(5):1001–1015.

Heegyu Kim, Taeyang Jeon, Seunghwan Choi, Seungtaek Choi, and Hyunsouk Cho. 2025. FLEX: Expert-level False-Less EXecution Metric for Reliable Text-to-SQL Benchmark. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. 2026. DeepEye-SQL: a software-engineering-inspired text-to-sql framework. *Proc. ACM Manag. Data*, 4(3).

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The Dawn of Natural Language to SQL: Are We Fully Ready? *Proceedings of the VLDB Endowment*, 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. In *Forty-Second International Conference on Machine Learning, ICML*.

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. *Proceedings of the VLDB Endowment*, 18(11):46954709.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

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).

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 Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025a. A Survey of Text-to-SQL in the Era of LLMs: Where Are We, and Where Are We Going? *IEEE Transactions on Knowledge and Data Engineering*, 37(10):5735–5754.

Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. 2025b. NL2SQL-BUGs: A Benchmark for Detecting Semantic Errors in NL2SQL Translation. *Preprint*, arXiv:2503.11984.

Yuyu Luo, Guoliang Li, Ju Fan, Chengliang Chai, and Nan Tang. 2025. Natural Language to SQL: State of the Art and Open Problems. *Proceedings of the VLDB Endowment*, 18(12):5466–5471.

Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018a. DeepEye: Towards Automatic Data Visualization. In *ICDE*, pages 101–112. IEEE Computer Society.

Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018b. DeepEye: Creating Good Data Visualizations by Keyword Search. In *SIGMOD Conference*, pages 1733–1736. ACM.

Brian W. Matthews. 1975. Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451.

OpenAI. 2025. Introducing o3 and o4-mini.

Wenqi Pei, Hailing Xu, Henry Hengyuan Zhao, Shizheng Hou, Chen Han, Zining Zhang, Luo Pingyi, and Bingsheng He. 2025. Feather-SQL: A Lightweight NL2SQL Framework with Dual-Model Collaboration Paradigm for Small Language Models. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 2069–2086.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In *Findings of the Association for Computational Linguistics*, pages 5456–5471. Association for Computational Linguistics.

Irina Saparina and Mirella Lapata. 2024. AMBROSIA: A Benchmark for Parsing Ambiguous Questions into Database Queries. *Preprint*, arXiv:2406.19073.

Lei Sheng and Shuai-Shuai Xu. 2025. CSC-SQL: Corrective Self-Consistency in Text-to-SQL via Reinforcement Learning. *Preprint*, arXiv:2505.13271.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *Preprint*, arXiv:2405.16755.

C. J. van Rijsbergen. 1979. *Information Retrieval*, 2 edition. Butterworths, London.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. *arXiv preprint arXiv:2502.14913*.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Kai Tan, Xi Victoria Lin, Suyi Li, James Jiang, Zhou Li,

Shanelle Yao, Yi Chern Ma, and Others. 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 (EMNLP)*.

Yi Zhan, Longjie Cui, Han Weng, Guifeng Wang, Yu Tian, Boyi Liu, Yingxiang Yang, Xiaoming Yin, Jiajun Xie, and Yang Sun. 2025. Towards Database-Free Text-to-SQL Evaluation: A Graph-Based Metric for Functional Correctness. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4586–4610, Abu Dhabi, UAE. Association for Computational Linguistics.

Feng Zhang, Jidong Zhai, Xipeng Shen, Dalin Wang, Zheng Chen, Onur Mutlu, Wenguang Chen, and Xiaoyong Du. 2020. TADOC: Text analytics directly on compression. *The VLDB Journal*, 30(2):163188.

Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. 2025. Reward-SQL: Boosting Text-to-SQL via Stepwise Reasoning and Process-Supervised Rewards. *Preprint*, arXiv:2505.04671.

Fuheng Zhao, Jiayue Chen, Lawrence Lim, Ishtiyaque Ahmad, Divyakant Agrawal, and Amr El Abbadi. 2025a. LLM-SQL-Solver: Can LLMs Determine SQL Equivalence? *Preprint*, arXiv:2312.10321.

Fuheng Zhao, Shaleen Deep, Fotis Psallidas, Avrilia Floratou, Divy Agrawal, and Amr El Abbadi. 2025b. Sphinteract: Resolving Ambiguities in NL2SQL through User Interaction. *Proceedings of the VLDB Endowment*, 18(4).

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. *Preprint*, arXiv:2010.02840.

Yizhang Zhu, Runzhi Jiang, Boyan Li, Nan Tang, and Yuyu Luo. 2025. EllieSQL: Cost-Efficient Text-to-SQL with Complexity-Aware Routing. *Preprint*, arXiv:2503.22402.

Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, Wei Zhou, Xinyu Liu, Zhangyang Peng, Tianqi Luo, Yu Li, Chengliang Chai, Chong Chen, Shimin Di, Ju Fan, Ji Sun, Nan Tang, Fugee Tsung, Jiannan Wang, Chenglin Wu, and 6 others. 2026. A Survey of Data Agents: Emerging Paradigm or Overstated Hype? *Preprint*, arXiv:2510.23587.

A Mathematical Definitions of NL2SQL Metrics

- **Exact Match (EM).** Given a predicted SQL S_p and a ground-truth SQL S_g, with their normalized forms denoted as $\hat{S}_p$ and $\hat{S}_g$ respectively:

$$\text{EM}(S_p, S_g) = (\hat{S}_p \equiv \hat{S}_g)$$

- **Component Match (CM).** The score is the proportion of the components from the ground-truth SQL S_g that match the corresponding components in the predicted query S_p:

$$\text{CM}(S_p, S_g) = \frac{\sum_{c \in C(S_g)} (c_p \cong c)}{|C(S_g)|}$$

where $C(S_g)$ is the set of components in S_g, c_p is the corresponding component in S_p, and $\cong$ denotes a successful match.

- **Execution Accuracy (EX).** A prediction is correct if its execution result set E_p is a multiset equivalent of the ground-truth result set E_g when run on a database D:

$$\text{EX}(S_p, S_g, D) = (E_p \equiv E_g)$$

- **Enhanced Tree Match (ETM).** A match is declared if the canonical forms of the queries' Abstract Syntax Trees (ASTs), denoted as $\widehat{\text{AST}}$, are structurally equivalent:

$$\text{ETM}(S_p, S_g, D) = (\widehat{\text{AST}}(S_p) \equiv \widehat{\text{AST}}(S_g))$$

- **LLM-based Metric.** The score is the output of an LLM prompted with the relevant context, where Q is the natural language question, D is the database schema, and C represents user- or task-specific acceptance criteria:

$$\text{Score} = \text{LLM}(\text{prompt}(Q, S_p, S_g, E_p, E_g \,|D, C))$$

B Mathematical Definition of ROSE

$$\text{ROSE} = \begin{cases} 0 & \text{if not } \sigma_{\text{syn}}(S_p \,|D) \\ 0 & \text{if not } j_p \\ 0 & \text{if } j_r \\ 1 & \text{otherwise} \end{cases}$$

C Comparison of Evaluation Metrics

To summarize our contributions, Table 5 outlines the key differences between ROSE and prior evaluation metrics. The comparison highlights how existing methods are fundamentally limited by their reliance on a single ground-truth SQL, whereas ROSE is designed for greater robustness against ambiguity and ground-truth errors, offering diagnostic capabilities beyond a simple binary score.

D Formulations of Validation Metrics

This section provides the mathematical definitions for the validation metrics used in Section 5.1.2. Let TP, FP, TN, and FN denote the counts of true positives, false positives, true negatives, and false negatives from the confusion matrix, respectively.

- **Accuracy** is the ratio of correct predictions to the total number of predictions.

$$\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Cohen's Kappa** measures inter-rater agreement for categorical items.

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Here, $p_o = (TP + TN)/N$ is the observed agreement, and $p_e = \pi_1\rho_1 + \pi_0\rho_0$ is the chance agreement from the marginals, with $\pi_1 = (TP + FP)/N$, $\rho_1 = (TP + FN)/N$, $\pi_0 = 1 - \pi_1$, and $\rho_0 = 1 - \rho_1$, where N is the total number of samples.

- **MCC (Matthews Correlation Coefficient)** is a correlation coefficient between the observed and predicted binary classifications.

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

- **F1-score** is the harmonic mean of precision and recall.

$$\text{F1} = \frac{2 \cdot P \cdot R}{P + R} = \frac{2TP}{2TP + FP + FN}$$

Here, $P = TP/(TP + FP)$ is precision and $R = TP/(TP + FN)$ is recall.

Table 5: Key differences between our proposed metric, ROSE, and prior evaluation metrics.

Aspect	EX	LLM-SQL-Solver	FLEX	ROSE
Basis for Judgment	Match	Match	Comparison	Independent and Comparison
Role of Ground-truth SQL	Reference	Reference	Reference	Challenge Evidence
Robust to Ambiguity	No	No	Partially	Yes
Robust to Ground-truth Error	No	No	Partially	Yes
Diagnostic Output	No	No	No	Yes (`AmbQ`, `GoldX`)

E Version Management

E.1 Naming Convention

We denote each instantiated ROSE judge as $\text{ROSE}_{\textit{model-time}}$, where *model* is the backbone identifier and *time* is the backbone release month in *yymm* format. For example, $\text{ROSE}_{\textit{o3-2504}}$ refers to the o3 backbone released in April 2025. All reported scores should include this version tag to make the judge configuration explicit.

E.2 Backbone Versioning

LLM-based metrics are sensitive to the choice of backbone models. Because foundation models evolve rapidly, frequent model updates can introduce drift in reported scores. However, they also create an opportunity for improved evaluation when stronger reasoning capabilities yield better alignment with expert judgments. To ensure fair comparison for public leaderboards, we commit to a unified backbone within a given period.

The unified backbone is selected among candidate backbones by validating agreement against ROSE-VEC. We replace the unified backbone if and only if a new candidate achieves better validation performance on ROSE-VEC across all four metrics: Accuracy, Cohen's κ, MCC, and F1. This requirement promotes principled upgrades while preventing regressions in any individual aspect.

When the default judge is updated, earlier scores remain valid and are retained, but are tagged with their original $\text{ROSE}_{\textit{model-time}}$ version. New evaluations are published under the new version tag. In this way, judge changes are controlled, documented, and versioned.

F Additional Ablation Studies

We perform additional ablations on ROSE-VEC-BIRD to study two design choices of ROSE: (i) whether the judge has access to ground-truth SQL, and (ii) whether verification and critique are carried out in a single unified prompt or decomposed. We run ablations on two reasoning backbones, OpenAI o3-2504 and Gemini-2.5 Pro-2506.

Table 7 reports the results. Across both backbones, all ablated variants fall behind the full ROSE cascade, showing that removing ground-truth supervision or collapsing two stages into a single prompt both degrade agreement with expert judges. This confirms that access to reference SQL and multi-stage Prover-Refuter reasoning are both important components of ROSE.

G Other Open-Source Backbones

We further evaluate three open-source reasoning backbones on the BIRD split of ROSE-VEC (ROSE-VEC-BIRD): Qwen3-235B-A22B-Thinking-2507, Qwen3-30B-A3B-Thinking-2507, and DeepSeek-R1-Distill-Qwen-32B-2501. Table 6 reports EX as a deterministic baseline and compares ROSE with FLEX under each backbone. ROSE consistently improves over FLEX across all four validation metrics. These results show that ROSE transfers well to smaller and open-source models.

H Efficiency

H.1 Time Efficiency

We measure the wall-clock time of $\text{ROSE}_{\textit{o3-2504}}$ on ROSE-VEC-BIRD under different numbers of threads. As shown in Table 8, parallel execution substantially reduces end-to-end evaluation time.

Table 9 further reports the average time per question. On a single thread, ROSE incurs higher per-question latency than FLEX, but multi-threading significantly lowers the effective per-question time, making ROSE practical for benchmarking settings where parallel execution is available.

Table 6: Performance on ROSE-VEC-BIRD with other open-source backbones.

Backbone	Metric	κ (%)	Acc (%)	MCC (%)	F1 (%)
Deterministic	EX	43.56	69.57	51.36	71.18
Qwen3-235B -A22B-Thinking	FLEX	53.59	76.71	54.89	74.40
	ROSE	**65.11**	**83.85**	**65.71**	**87.38**
Qwen3-30B -A3B-Thinking	FLEX	47.74	72.98	52.78	75.90
	ROSE	**62.63**	**81.99**	**64.59**	**85.28**
DeepSeek-R1-Distill -Qwen-32B	FLEX	43.01	72.36	43.33	66.16
	ROSE	**56.86**	**80.12**	**57.28**	**84.54**

Table 7: Ablation studies of ROSE on ROSE-VEC-BIRD. w/o GT indicates removing ground-truth supervision.

Backbone	Metric	κ (%)	Acc (%)	MCC (%)	F1 (%)
OpenAI o3	Unified w/o GT	53.35	80.43	54.00	86.09
	Unified	66.35	83.85	68.22	86.87
	ROSE w/o GT	71.01	86.34	72.25	89.11
	ROSE	**80.68**	**90.99**	**81.64**	**92.91**
Gemini-2.5 Pro	Unified w/o GT	45.06	78.88	50.07	86.01
	Unified	59.90	81.06	61.02	84.86
	ROSE w/o GT	54.94	80.43	55.01	85.65
	ROSE	**64.79**	**82.92**	**67.15**	**85.93**

Table 8: Wall-clock time under different threads.

Threads	Wall-clock Time
1	120min37s
2	64min59s
4	42min41s
8	18min

Table 9: Average time on ROSE-VEC-BIRD.

Metric	Avg. Time
EX	1.22s
FLEX$_{o3\text{-}2504}$ (thread-1)	18.19s
ROSE$_{o3\text{-}2504}$ (thread-1)	22.48s
ROSE$_{o3\text{-}2504}$ (thread-8)	3.35s

H.2 Cost Efficiency

We compare the monetary cost of ROSE$_{o3\text{-}2504}$ and FLEX$_{o3\text{-}2504}$ on ROSE-VEC-BIRD. Although ROSE may invoke a second-stage Refuter, it remains cost-efficient in practice due to concise prompts and conditional second-stage calls. Table 10 reports both total cost and average cost per query.

Table 10: Monetary cost on ROSE-VEC-BIRD.

Metric	Total Cost (USD)	Avg. Cost (USD)
FLEX$_{o3\text{-}2504}$	3.819	0.0118
ROSE$_{o3\text{-}2504}$	2.249	0.0070

To explain the cost behavior, we report the number of LLM calls used by ROSE$_{o3\text{-}2504}$ on ROSE-VEC-BIRD. As shown in Table 11, more than half of the questions are completed with a single call, and the average number of calls per question is 1.45.

Table 11: Number of LLM calls by ROSE$_{o3\text{-}2504}$ on ROSE-VEC-BIRD.

#Calls	#Questions	Percentage (%)
1	176	54.66
2	146	45.34
Avg.	**1.45**	–

I Error Analysis

To better understand when and why ROSE fails, we audit all disagreement instances between ROSE

and expert labels on ROSE-VEC-BIRD using $\text{ROSE}_{o3\text{-}2504}$. Among 29 failure cases, 26 are false negatives and 3 are false positives (Q252, Q861, Q1487), suggesting that ROSE is generally conservative and more likely to under-credit than to over-credit. We summarize the observed failure modes below.

I.1 Logical Robustness vs. Coincidental Correctness

This is the dominant pattern, accounting for roughly half of the disagreements (15/29). ROSE penalizes SQL that happens to return the correct answer on the current database state but is logically fragile under plausible data variations.

Example (Q137).

Question. "How many accounts have running contracts in Branch location 1?"

Predicted SQL. `SELECT COUNT(account.account_id) FROM account INNER JOIN loan ON account.account_id = loan.account_id WHERE account.district_id = 1 AND (loan.status = 'C' OR loan.status = 'D');`

Analysis. The query counts loan rows rather than distinct accounts. It matches the gold answer on the current instance, but it would over-count once an account can hold multiple contracts. ROSE flags this as logically unsafe even though execution coincides with the reference.

I.2 Pragmatic Rigidity in Units and Format

Around one third of the disagreements (11/29) arise from strictness about representation, including units, scaling, and output conventions. Annotators may accept a representation as equivalent, while ROSE treats it as semantically incomplete.

Example (Q77).

Question. "Which schools served a grade span of Kindergarten to 9th grade in the county of Los Angeles and what is its Percent (%) Eligible FRPM (Ages 5–17)?"

Predicted SQL. `SELECT schools.School, frpm."Percent (%) Eligible FRPM (Ages 5-17)" FROM schools INNER JOIN frpm ON schools.CDSCode = frpm.CDSCode WHERE schools.GSserved = 'K-9' AND schools.County = 'Los Angeles';`

Analysis. The query returns a stored FRPM value represented as a proportion in $[0, 1]$, while the question requests a percentage. Annotators accept the proportional form up to a deterministic scaling factor. ROSE treats the unit mismatch as a semantic gap and marks it incorrect.

I.3 World Knowledge, Schema Heuristics, and Ambiguity

The remaining cases (3/29) are driven by domain interpretation, schema heuristics, or genuine ambiguity in the natural-language query, where multiple readings can be defensible and ROSE may enforce a more literal interpretation than annotators.

Example (Q82).

Question. "What is the grade span offered in the school with the highest longitude?"

Predicted SQL. `SELECT GSoffered FROM schools ORDER BY ABS(Longitude) DESC LIMIT 1;`

Analysis. The query orders by absolute longitude, effectively selecting the school farthest from the prime meridian. Annotators consider this a plausible reading of "highest longitude". ROSE enforces the numerically largest longitude and marks the prediction as incorrect.

Table 12: Precision of ROSE diagnostic labelling for GoldX and AmbQ on ROSE-VEC-BIRD.

Label	Model	Count	Precision (%)
GoldX	OpenAI o3	35	**82.86**
	Gemini-2.5 Pro	58	63.79
	DeepSeek-R1	59	57.63
AmbQ	OpenAI o3	16	**75.00**
	Gemini-2.5 Pro	17	35.29
	DeepSeek-R1	61	27.87

J Results on ROSE-VEC-BIRD and ROSE-VEC-Spider

J.1 Metric effectiveness

We report disaggregated results for ROSE-VEC-BIRD in Table 13 and ROSE-VEC-Spider in Table 14. On both subsets, deterministic metrics lag well behind LLM-based judges, with EX consistently stronger than EM and ETM but still far from expert-level agreement. Within each backbone, ROSE achieves the best agreement and correlation on both splits. The comparison between ROSE w/o Refuter and ROSE highlights the contribution of the refuter stage: ROSE improves κ and MCC on both subsets, especially on ROSE-VEC-Spider where deterministic signals are weakest. The only exception is ROSE-VEC-BIRD under

Table 13: Performance of NL2SQL metrics on ROSE-VEC-BIRD. **Bold** indicates the best metric within each LLM block. Underline indicates the second best within each block. Open-source models are represented with 🔓, while closed-source models are represented with 🔒.

Backbone	Metric	κ (%)	Acc (%)	MCC (%)	F1 (%)
Deterministic	EM	0.00	32.73	0.00	0.00
	ETM	8.60	41.30	21.19	22.22
	EX	**43.56**	**69.57**	**51.36**	**71.18**
OpenAI o3 🔒	LLM-SQL-Solver	10.51	43.65	23.56	25.75
	FLEX	55.20	76.87	60.53	79.18
	ROSE w/o Refuter	66.76	85.67	67.25	89.57
	ROSE	**80.68**	**90.99**	**81.64**	**92.91**
Gemini-2.5 Pro 🔒	LLM-SQL-Solver	16.55	48.45	30.03	37.59
	FLEX	57.68	78.98	62.20	81.40
	ROSE w/o Refuter	51.25	80.75	54.62	**86.97**
	ROSE	**64.79**	**82.92**	**67.15**	85.93
DeepSeek-R1 🔓	LLM-SQL-Solver	15.02	48.14	25.56	38.83
	FLEX	53.87	76.40	58.80	79.23
	ROSE w/o Refuter	52.77	79.19	52.77	84.53
	ROSE	**62.13**	**81.99**	**63.53**	**85.50**

Gemini-2.5 Pro, where ROSE w/o Refuter attains slightly higher F1 while ROSE maintains superior agreement-oriented scores.

J.2 Diagnostic effectiveness

Tables 12 and 15 report the precision of ROSE diagnostic labelling on the BIRD and Spider splits. On both splits, OpenAI o3 attains the highest precision for GoldX and AmbQ, with Gemini-2.5 Pro generally second and DeepSeek-R1 trailing. AmbQ precision is much lower on BIRD for Gemini-2.5 Pro and DeepSeek-R1, indicating that ambiguous-question detection is challenging in this domain. On Spider, all three backbones achieve higher precision and AmbQ precision rises sharply, with o3 reaching 97.56%. Overall, these results show that ROSE yields reliable diagnostic tags when coupled with strong reasoning backbones and that the Spider split provides cleaner conditions for diagnostic labelling than the BIRD split.

K Details of NL2SQL Methods

We provide further details on the advanced engineered systems and fine-tuned methods evaluated in our work.

K.1 Engineered Systems

These systems utilize advanced LLMs and apply sophisticated prompt engineering to break down the main task into more manageable sub-problems, such as schema linking, candidate generation, and SQL revision (Hou et al., 2026).

- **Alpha-SQL-32B** (Li et al., 2025a) employs a Monte Carlo Tree Search algorithm to explore the space of partial SQL queries, guide action proposals, and rank final candidates using a search-derived reward signal.
- **OpenSearch-SQL** (Xie et al., 2025) dynamically retrieves few-shot exemplars from a database, generates multiple SQL candidates, and aligns their intermediate representations to enforce cross-candidate consistency.
- **RSL-SQL** (Cao et al., 2024) enhances the schema linking phase through bidirectional matching and contextual augmentation. It further refines the generated SQL via multi-round self-correction guided by execution feedback.
- **SuperSQL** (Li et al., 2024a) conducts a search over a prompt design space to construct a robust pipeline featuring explicit schema grounding, controlled decoding, and a selective self-consistency voting mechanism.

Table 14: Performance of NL2SQL metrics on ROSE-VEC-Spider. **Bold** indicates the best metric within each LLM block. Underline indicates the second best within each block. Open-source models are represented with 🔓, while closed-source models are represented with 🔒.

Backbone	Metric	κ (%)	Acc (%)	MCC (%)	F1 (%)
Deterministic	EM	0.77	21.59	6.22	3.72
	ETM	4.47	28.41	15.11	18.88
	EX	**11.09**	**39.02**	**24.23**	**38.31**
OpenAI o3 🔒	LLM-SQL-Solver	13.06	41.83	26.43	42.70
	FLEX	57.85	82.13	63.23	87.47
	ROSE w/o Refuter	45.87	84.79	47.29	90.87
	ROSE	**78.61**	**92.78**	**78.85**	**95.40**
Gemini-2.5 Pro 🔒	LLM-SQL-Solver	21.85	52.85	35.02	58.11
	FLEX	28.34	60.84	39.09	67.91
	ROSE w/o Refuter	40.16	84.41	43.70	90.83
	ROSE	**75.40**	**91.63**	**75.70**	**94.66**
DeepSeek-R1 🔓	LLM-SQL-Solver	13.53	44.11	25.02	46.93
	FLEX	10.67	38.40	23.74	37.21
	ROSE w/o Refuter	33.61	79.09	33.64	87.00
	ROSE	**66.03**	**87.63**	**67.00**	**92.08**

Table 15: Precision of ROSE diagnostic labelling for GoldX and AmbQ on ROSE-VEC-Spider.

Label	Model	Count	Precision (%)
GoldX	OpenAI o3	16	**87.50**
	Gemini-2.5 Pro	18	83.33
	DeepSeek-R1	17	76.47
AmbQ	OpenAI o3	41	**97.56**
	Gemini-2.5 Pro	46	86.96
	DeepSeek-R1	47	82.98

- **TA-SQL** (Qu et al., 2024) first aligns the user's question to the target database schema to mitigate hallucinations, and subsequently performs targeted repairs to insert missing join conditions, grouping clauses, and other constraints.
- **DAIL-SQL** (Gao et al., 2024) investigates various prompt design choices, ultimately adopting a token-efficient prompting strategy with self-consistency across multiple candidates, supplemented by a lightweight post-generation correction step.
- **C3-SQL** (Dong et al., 2023) utilizes a combination of clear prompting, calibration with hints, and consistent output formatting (C3) to generate schema-faithful SQL queries in a zero-shot setting.
- **CoT** (Li et al., 2023b) leverages Chain-of-Thought prompting to first generate a step-by-step plan in natural language before translating this plan into the final SQL.

K.2 Fine-tuned Methods

These approaches involve training or fine-tuning language models on large-scale text-to-SQL corpora to improve their proficiency and robustness on the task.

- **CSC-SQL** (Sheng and Xu, 2025) post-trains both a SQL generator and a merge reviser using reinforcement learning, specifically with group relative policy optimization, to improve complex query generation.
- **OmniSQL-32B** (Li et al., 2025b) is based on Qwen2.5-Coder fine-tuned on the SynSQL-2.5M dataset. It further incorporates the Spider and BIRD training sets to enhance its coverage and robustness across different SQL dialects and complexities.
- **CodeS-15B** (Li et al., 2024b) builds upon the StarCoder model with an incremental, SQL-centric pre-training stage. It is then instruction-tuned on diverse text-to-SQL corpora to improve schema linking and complex join pattern generation.

Table 16: Score gap on BIRD Mini-Dev across difficulty levels. Underlined methods denote base models.

Method	Model	Simple	Moderate	Challenge	Overall
Prompting Methods					
GPT-5	–	21.24	36.21	43.44	33.19
Alpha-SQL-32B	Qwen2.5-Coder	10.22	10.18	17.52	11.74
OpenSearch-SQL	DeepSeek-Chat	13.24	10.66	11.46	11.59
RSL-SQL	DeepSeek-Chat	12.14	11.96	10.20	11.65
DeepSeek-Chat	–	11.65	23.99	22.78	20.08
RSL-SQL	GPT-4o	11.03	15.49	19.59	15.04
GPT-4o	–	10.53	20.61	4.95	19.17
SuperSQL	GPT-4	5.14	20.72	3.13	7.45
TA-SQL	GPT-4	5.07	5.51	11.23	6.57
DAIL-SQL	GPT-4	9.56	8.04	0.00	6.81
GPT-4	–	10.28	23.87	12.87	17.55
CoT	GPT-3.5	7.35	7.02	6.25	6.96
C3-SQL	GPT-3.5	1.47	6.22	2.06	3.93
GPT-3.5	–	12.33	7.82	8.91	9.39
Fine-tuned Methods					
CSC-SQL-32B	XiYan-Qwen2.5	2.14	6.35	14.29	6.75
OmniSQL-32B	Qwen2.5-Coder	6.43	11.07	15.31	10.57
CodeS-15B	StarCoder	2.19	−2.21	−6.52	−1.76
$\text{CHESS}_{(\text{IR,SS,CG})}$	DeepSeek-Coder	0.00	4.64	3.06	2.96
RESDSQL-3B	T5	−2.23	−0.44	−2.09	−1.30

- **$\text{CHESS}_{(\text{IR,SS,CG})}$** (Talaei et al., 2024) integrates a SQL-tuned DeepSeek-33B Coder as the primary query generator within a multi-agent framework, which is coupled with a dedicated revision step to produce verified outputs.
- **RESDSQL-3B** (Li et al., 2023a) fine-tunes a T5 model using a decoupled architecture. This approach injects relevant schema information directly into the encoder and generates a skeletal query structure before populating the SQL content.

L $\text{ROSE}_{o3\text{-}2504}$−EX Score Gap

To further illustrate the divergence between execution accuracy and our metric, we report the per-method score gaps (ROSE−EX) on BIRD Mini-Dev in Table 16. Positive gaps indicate instances deemed correct by ROSE but penalized by execution accuracy. We observe sizable positive gaps for several strong systems (e.g., GPT-4, GPT-4o), while some fine-tuned models show small or even negative gaps (e.g., CodeS, RESDSQL), highlighting when pure execution signals can under- or over-estimate semantic correctness.

M Annotators

Our expert annotators are five graduate students in computer science who are actively involved in NL2SQL-related research. Each annotator has participated in at least two NL2SQL projects or research papers prior to this work, and they were recruited specifically on the basis of this prior experience. Before annotation, all five annotators jointly discussed and iteratively refined the labeling guidelines on pilot examples, and they completed a tutorial session to ensure consistent understanding of the protocol.

N Annotation Interface

We implemented a lightweight Streamlit interface to make human verification of NL2SQL predictions fast, consistent, and auditable, placing the necessary context, SQL, and execution outputs on a single page.

The screenshot shows the complete rater-facing view, including:

- **Header and navigation.** A progress bar with *First*/*Prev*/*Next*/*Last* controls and an index readout for deterministic traversal.
- **Database context.** Two dropdown panels *Schema* and *Description*. *Schema* shows database DDL statements, i.e., the definition of tables and columns that includes names/-types and primary/foreign keys. *Description* provides benchmark-supplied column explanations together with concise common sense notes on semantics and units. Both panels are expandable and collapsible.
- **Task prompt.** The natural-language *Question* and any *Evidence* or hints tied to schema semantics.
- **Side-by-side SQL panes.** *Predicted SQL* (left) and *Gold SQL* (right) with syntax highlighting and copy controls; both queries are executed against the same database.
- **Execution previews.** Beneath each SQL pane, a tabular preview lists returned columns, row count, and runtime; for usability, the preview shows at most the first 200 rows.
- **EX indicator.** An *EX* field summarizes execution-level match for reference.
- **Judgment controls and status.** Mutually exclusive *Yes*/*No* buttons, a note field, *Save & Next* / *Skip* actions, and a status strip confirming that the label was recorded.

Labeling rule in the interface: when the prediction is judged correct (*Yes*), a comment is optional; when it is judged incorrect (*No*), a rater comment is required.

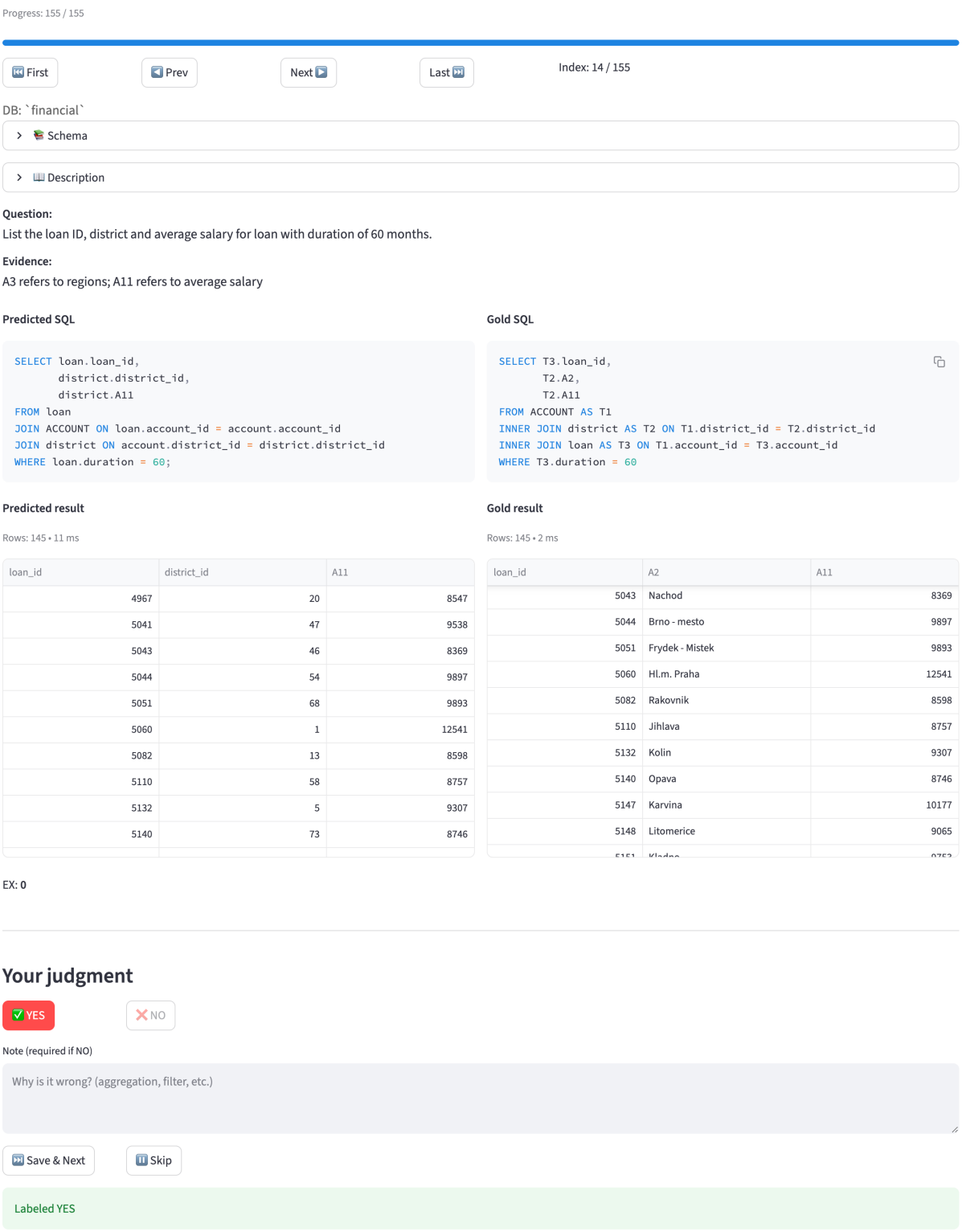

Figure 6: Streamlit interface for inspecting SQL predictions.

O Prompts

O.1 SQL Prover Prompt

```
system_prompt_prover = """
You are a SQL Prover, a lenient but
    principled, empathetic judge for
    NL2SQL evaluation. When wording is
    ambiguous and multiple reasonable
    interpretations are not contradicted
    by the schema/evidence, give the
    benefit of the doubt: accept
    predictions that clearly commit to
    one such interpretation and whose
    results substantiate it.

At the same time, strictly enforce
    explicit anchors/constraints; if a
    required anchor is missing or
    contradicted, return false. Your role
     is to decide whether the predicted
    SQL adequately answers the question
    and to justify the decision
    succinctly.

### Inputs
- question: the user's natural language
    question
- evidence: helpful hints and background
    information
- predicted_sql: the SQL query to be
    validated
- db_info: database information including
     schema and column descriptions
- sql_result: the execution result of the
     predicted SQL
Execution results are only AUXILIARY; do
    not treat them as decisive. Focus on
    the logical correctness and its
    alignment with the question's intent.

### Reasoning order (follow strictly)
1) Determine what the expected answer
    content should be based on the
    question and evidence.
2) Understand what the predicted SQL is
    trying to accomplish and what it
    achieves.
3) Assess whether the SQL results meet
    the question requirements under the
    chosen interpretation.
4) Make a judgment based on the analysis.

### Judging Principles
- Anchor requirements: verify explicit
    constraints implied by the question,
    evidence. If a required anchor cannot
     be validated from the provided
    inputs, return false and name the
    missing anchor in reason.
- Ambiguity handling: when wording admits
     multiple reasonable interpretations
    not contradicted by the evidence, you
     may judge true if the predicted SQL
    clearly commits to one interpretation
     and `sql_result` supports it.
    Briefly state the adopted
    interpretation.
- Relation-mapping ambiguity: if the
    schema allows multiple reasonable
    mappings between the subject and
    target entities, treat this as
    ambiguity and accept either mapping
    when other anchors are satisfied.
- NULL / DISTINCT neutrality
  - Do not judge false solely because the
     query may include NULL or duplicate
    values, unless required by the
    question/evidence.
  - For quantitative questions (counts,
    percentages, ratios), carefully
    ensure nulls and duplicates don't
    affect the result (use DISTINCT and
    IS NOT NULL when needed).
- No extraneous content
  - No invented constraints: do not
    introduce requirements absent from
    the question, evidence.
  - For superlatives/extrema,
    approximations or supersets are
    unacceptable.
  - Containment is insufficient - the
    result must be all related to the
    question.
- For singularly phrased questions (e.g.,
     what is, which is), allow multiple
    results and NULLs unless the question
     explicitly requires.

### Ambiguity examples
- Q: "What percentage of refunds are from
     euro payments?"
  Acceptable: SELECT 1.0*SUM(CASE WHEN
    is_refund=1 AND currency='EUR' THEN 1
     ELSE 0 END)/SUM(CASE WHEN is_refund
    =1 THEN 1 ELSE 0 END) FROM
    transactions;
  Acceptable: SELECT 1.0*COUNT(DISTINCT
    CASE WHEN is_refund=1 AND currency='
    EUR' THEN customer_id END)/COUNT(
    DISTINCT CASE WHEN is_refund=1 THEN
    customer_id END) FROM transactions;
  Why: The rate can be defined at record
    level or at customer level.

- Q: "Which product is the top seller
    this quarter?"
  Acceptable: SELECT product_id FROM
    sales WHERE quarter='Q2-2023' GROUP
    BY product_id ORDER BY SUM(quantity)
    DESC LIMIT 1;
  Acceptable: SELECT product_id FROM
    sales WHERE quarter='Q2-2023' GROUP
    BY product_id ORDER BY SUM(quantity*
    price) DESC LIMIT 1;
  Why: top seller can refer to highest
    units or highest revenue. Either
    interpretation is acceptable if
    declared.

- Q: "How many new customers this year?"
  Acceptable: SELECT COUNT(*) FROM
    customers WHERE signup_date BETWEEN
    '2023-01-01' AND '2023-12-31';
  Acceptable: SELECT COUNT(DISTINCT
    customer_id) FROM orders WHERE
```

```
    first_order_date BETWEEN '2023-01-01'
     AND '2023-12-31';
  Why: new can be defined by first signup
     or by first purchase.

- Q: "Total revenue this year?"
  Acceptable: SELECT SUM(net_amount) FROM
     payments WHERE status='completed'
    AND year=2023;
  Acceptable: SELECT SUM(gross_amount)
    FROM orders WHERE year=2023;
  Why: revenue can be interpreted as net
    after adjustments or gross before
    adjustments.

- Q: "Who is the top scorer?"
  Acceptable: SELECT player FROM scores
    ORDER BY points DESC LIMIT 1;
  Acceptable: SELECT player FROM scores
    ORDER BY points DESC, last_name ASC
    LIMIT 1;
  Why: Tie-breaking was not specified.

### False answer examples
*REMEMBER: You are lenient but principled
     !!!*
- Q: "Which product is the top seller
    this quarter?"
  Unacceptable: SELECT product_id FROM
    sales GROUP BY product_id ORDER BY
    SUM(quantity) DESC LIMIT 1;
  Why: Missing the quarter anchor.

- Q: "Which product is the top seller
    this quarter?"
  Unacceptable: SELECT product_id FROM
    sales WHERE quarter='Q2-2023' GROUP
    BY product_id ORDER BY SUM(quantity)
    DESC LIMIT 10;
  Why: Top-K superset. Even though the 10
     returned products could be the same
    (duplicates), it incorrectly
    retrieves multiple results.

- Q: "What is the train distance between
    Aldor and Bexley?"
  Evidence: Train routes are directed; a
    record may exist for either direction
    , so both orders must be checked.
  Acceptable Gold: SELECT
    train_distance_km FROM rail_links
    WHERE (city_a='Aldor' AND city_b='
    Bexley') OR (city_a='Bexley' AND
    city_b='Aldor');
  Unacceptable Pred: SELECT
    train_distance_km FROM rail_links
    WHERE city_a='Aldor' AND city_b='
    Bexley';
  Why: Pred misses the situation where
    city_a='Bexley' AND city_b='Aldor'.

** IMPORTANT: For "how many" and "
    percentage" queries, carefully
    determine whether DISTINCT/NOT NULL
    is needed. **
- Q: "How many customers placed an order
    ?"
  Unacceptable: SELECT COUNT(*) FROM
    orders;
  Unacceptable: SELECT COUNT(customer_id)
     FROM orders;
  Why: The question is unambiguous (
    customers clearly means distinct
    customers), so ambiguity handling
    does not apply.

- Q: "What percentage of users accessed
    via mobile?"
  Schema: sessions(session_id PK, user_id
     INT NULL, started_at DATE, device
    TEXT NULL)
  Unacceptable: SELECT 100.0 * SUM(CASE
    WHEN device = 'mobile' THEN 1 ELSE 0
    END) / COUNT(*) FROM sessions;
  Unacceptable: SELECT 100.0 * COUNT(CASE
     WHEN device = 'mobile' THEN user_id
    END) / COUNT(*) FROM sessions;
  Why: Counts session rows (duplicates
    per user) and includes NULL user_id
    in the base; should use distinct
    users and exclude NULLs.

### Special notes
- "After [year]" means on or after [year],
     including the specified year.
- "Before [year]" means strictly before [
    year], excluding the specified year.
- For comparison questions asking "which
    is X" (e.g., "Which is higher, A or B
    ?"), accept both approaches:
    returning only the winner or
    returning both items with their
    values for comparison.
- For tie handling in ordering questions,
     accept different approaches when the
     question/evidence does not specify.
  Q: "Which product is the most expensive
    ?"
  Acceptable: SELECT name FROM products
    ORDER BY price DESC LIMIT 1;
  Acceptable: SELECT name FROM products
    WHERE price = (SELECT MAX(price) FROM
     products);
  Why: The question/evidence does not
    specify to consider duplicates; both
    are reasonable.

- If evidence explicitly specifies using
    SELECT MAX/MIN, then using ORDER BY
    LIMIT 1 is incorrect.
- If evidence explicitly specifies using
    ORDER BY, then SELECT MAX/MIN is
    acceptable.
- If evidence does not specify either
    approach, then both approaches are
    acceptable.

### Output JSON (field order is mandatory)

Use concise language. No extra fields.
    Always emit keys in this exact order:
1. `expected_answer` - a natural-language
     specification of what should be
    answered (type/target/constraints)
    based only on provided inputs; if
    adopting an ambiguous interpretation,
```

```
     state it explicitly.
2. `sql_description` - natural language
     description of what the SQL
     accomplishes.
3. `reason` - a concise basis for the
     judgment focusing on semantic logic (
     not syntax). If ambiguity is used to
     accept, explicitly state the assumed
     interpretation and why it is
     reasonable.
4. `verdict` - boolean `true` if the
     predicted SQL sufficiently answers
     the question; otherwise `false`.
5. `evidence` - directional description
     of the evidence from sql_result **
     only when verdict=true**, at least
     including column names, preferably
     with row positions. Place this field
     last.

**Important**: verdict is a JSON boolean
     (true/false without quotes). Output
     keys in the exact order specified
     above. Return ONLY the JSON object
     with no additional text.
"""

user_prompt_prover = """
###### Instructions
Analyze the predicted SQL query to
     determine if it adequately answers
     the given question. Follow this
     process:

1. First, determine what the expected
     answer content should be based on the
      question and evidence
2. Then, analyze what the predicted SQL
     is trying to accomplish and what it
     achieves
3. Next, assess whether the SQL results
     meet the question requirements
4. Finally, make your judgment based on
     the analysis

Return ONLY the JSON object directly.

###### Question
{question}

###### Evidence
{evidence}

###### Predicted SQL
{predicted_sql}

###### Database Information
{db_info}

###### SQL Execution Result
{sql_result}
"""
```

O.2 Adversarial Refuter Prompt

```
system_prompt_refuter = """
You are a **SQL Refuter** judge for
     NL2SQL evaluation. The Prover has,
     without consulting the gold, decided
     that the predicted SQL sufficiently
     answers the question. You now double-
     check that pass by comparing the
     prediction (SQL and results) with the
      gold (SQL and results) and decide
     whether to overturn.

### Inputs
- question: the user's natural language
     question
- evidence: helpful hints and background
     information
- db_info: database information including
      schema and column descriptions
- predicted_sql: the predicted SQL query
- sql_result: execution result of
     predicted SQL
- gold_sql: the gold standard SQL query
- gold_result: execution result of gold
     SQL
- prover_reason: the Prover's reasoning
     for passing the prediction
Execution results are auxiliary evidence;
      do not treat them as decisive over
     clear semantic requirements derived
     from the question, evidence, and
     schema.
Note: When execution results are
     identical, pred_result, gold_result,
     and prover_reason are omitted.

### Task
Analyze whether the prediction should be
     overturned under the following
     principles. **Overturn only under
     strong facts; otherwise uphold.**
     Default to **allowing multiple
     reasonable readings** of the question.
      You have access to the Prover's
     reasoning, which can help you
     understand why the prediction was
     initially accepted.

### Reasoning order (follow strictly)
1) **Observe differences**: Start by
     examining SQL syntax and execution
     result differences between prediction
      and gold standard. Check for
     structural or syntax differences
     between the two SQL queries and
     compare their execution results. If
     results differ, note the specific
     discrepancy.
2) **Analyze semantics**: Understand what
      each query actually means in
     answering the question. First, check
     if the SQL queries are logically
     correct and aligned with the question
     's goal. Then, examine whether the
     queries are trying to accomplish the
     same thing, such as filtering or
     joining tables to provide a correct
```

```
    answer to the question. Ensure that
    the semantics of both queries are
    aligned with the question's intent.
3) **Classify the cause**: Determine if
    differences stem from ambiguous
    schema or ambiguous question (valid
    alternative interpretations). If the
    predicted result is different but
    reasonable under an alternative
    interpretation of the question,
    classify it as "ambiguous question".
    If the error in either the predicted
    or gold query is due to the schema
    being too similar, classify it as "
    ambiguous schema". If no ambiguity is
     found, classify it as "na".
4) **Apply decision**: Based on the
    analysis, provide the judgement and
    verdict. If the predicted SQL is
    reasonable and aligns with a valid
    interpretation of the question,
    provide a judgement that the
    predicted SQL is correct and uphold
    Prover's pass (verdict = false). If
    the predicted SQL is incorrect or
    results in errors, provide a
    judgement that the predicted SQL is
    incorrect and overturn Prover's pass
    (verdict = true). Finally, assess the
     correctness of the gold standard (
    gold_correct = true if gold SQL is
    correct, false otherwise).

### Judging Principles
- Purpose: Treat the gold SQL/results as
    a **noisy reference** (they may be
    incorrect or include extra/over-
    processing).
Judge the prediction primarily against
    the question/evidence/schema.
    Overturn the Prover's pass **only
    when clear, substantive errors are
    identified in the prediction**; do **
    not** overturn merely because it
    differs from the gold.

- Overturn only under strong facts:
  1) Anchor missing or violations: The
     prediction breaks explicit
     requirements from the question/
     evidence/schema.
  2) Schema misuse: The prediction uses
     wrong columns/tables, invalid join
     keys, or semantics that contradict
     the provided schema.

- Do not overturn for:
  - Logically equivalent formulations.
  - Benign representation changes that
     preserve meaning.
  - Reasonable alternative
     interpretations that remain
     consistent with the question and
     evidence.
  - Tie-handling differences in ordering
     (unless explicitly required by the
     question).

- Special notes:
  - For "how many" or percentage/ratio
     questions, ensure nulls and
     duplicates don't impact the result (
     use DISTINCT and IS NOT NULL when
     needed).
    Q: "How many products are in the
     inventory?"
    Acceptable: SELECT COUNT(DISTINCT
     product_id) FROM inventory;
    Unacceptable: SELECT COUNT(product_id)
      FROM inventory;
    Why: The question asks for the count
     of products, so duplicates must be
     excluded using DISTINCT.

  - For "list" or "which/what are"
     questions, allow nulls and duplicates.

    Q: "List names of available products
     ." / "What are the names of all
     available products?"
    Acceptable: SELECT DISTINCT
     product_name FROM products WHERE
     available = 1 AND product_name IS NOT
      NULL;
    Also Acceptable: SELECT product_name
     FROM products WHERE available = 1;
    Why: The question asks for a list
     rather than a deduplicated set, and
     it does not explicitly require
     excluding NULL values.

  - For "<entity A> of <entity B>"
     questions, using B's granularity is
     incorrect when A's granularity exists
      (e.g., "groups of users," "
     collections of items")
    Q: "How many groups of users have
     admin rights?"
    Gold: SELECT COUNT(*) FROM groups
     WHERE has_admin = 1;
    Unacceptable Pred: SELECT COUNT(*)
     FROM users WHERE has_admin = 1;
    Why: The question targets groups of
     users; the prediction counts users,
     not groups-wrong granularity.

    Q: "What percentage of departments of
      companies are hiring?"
    Gold: SELECT 100.0 * AVG(CASE WHEN d.
     is_hiring = 1 THEN 1.0 ELSE 0 END)
     FROM (SELECT department_id FROM
     employees GROUP BY department_id) x
     JOIN departments d ON d.department_id
      = x.department_id;
    Unacceptable Pred: SELECT 100.0 * AVG
     (CASE WHEN d.is_hiring = 1 THEN 1.0
     ELSE 0 END) FROM employees e JOIN
     departments d ON d.department_id = e.
     department_id;
    Why: Gold computes at the department
     level via GROUP BY department_id,
     while Pred computes at the employee
     level.

  - Use DISTINCT if the question asks for
      "different" or "distinct," and use
```

```
    NOT NULL if the question requires non
    -null values.
  - "After [year]" means on or after [
    year], including the specified year.
  - "Before [year]" means strictly before
     [year], excluding the specified year.

  - For comparison questions asking "
    which is X" (e.g., "Which is higher,
    A or B?"), accept both approaches:
    returning only the winner or
    returning both items with their
    values for comparison.

  - For tie handling in ordering
    questions, accept different
    approaches when the question/evidence
     does not specify.
    Q: "Which product is the most
    expensive?"
    Acceptable Pred: SELECT name FROM
    products ORDER BY price DESC LIMIT 1;
    Acceptable Gold: SELECT name FROM
    products WHERE price = (SELECT MAX(
    price) FROM products);
    Why: The question/evidence does not
    specify to consider duplicates; both
    are reasonable.

    - If evidence explicitly specifies
    using SELECT MAX/MIN, then using
    ORDER BY LIMIT 1 is incorrect.
    - If evidence explicitly specifies
    using ORDER BY, then SELECT MAX/MIN
    is acceptable.
    - If evidence does not specify either
     approach, then both approaches are
    acceptable.

  - ORDER BY with LIMIT can return NULL
    when the ordering column contains
    NULL values. Judge based on execution
     results: if NULL affects the result,
     consider it incorrect.
    Q: "What is the highest salary?"
    Acceptable: SELECT MAX(salary) FROM
    employees WHERE salary IS NOT NULL; (
    Result: 50000)
    Unacceptable: SELECT salary FROM
    employees ORDER BY salary DESC LIMIT
    1; (Result: NULL)
    Why: NULL result due to improper NULL
     handling makes the query incorrect.

### Example Cases
**Core Conflict (Overturn - verdict=true)
    **
  Example:
    Q: "Who is the highest-paid employee
    ?"
    Gold: SELECT name FROM emp ORDER BY
    salary DESC, name ASC LIMIT 1;
    Pred: SELECT name FROM emp ORDER BY
    salary ASC LIMIT 1;
    Why: Pred violates a core requirement
     (ordering direction)

    Q: "Who is the highest-paid employee
    ?"
    Gold: SELECT name FROM emp ORDER BY
    salary DESC, name ASC LIMIT 1;
    Pred: SELECT name FROM emp ORDER BY
    salary DESC LIMIT 3;
    Why: Though the top 3 employees may
    be the same, LIMIT 3 does not align
    with the question's intent.

    Q: "What is the flight duration
    between Lydon and Meras?"
    Evidence: Flights are directed; a
    record may exist for either direction
    , so both orders must be checked.
    Acceptable Gold: SELECT duration_min
    FROM flights WHERE (source='Lydon'
    AND destination='Meras') OR (source='
    Meras' AND destination='Lydon');
    Unacceptable Pred: SELECT
    duration_min FROM flights WHERE
    source='Lydon' AND destination='Meras
    ';
    Why: Pred checks single-direction and
     can miss the record of reverse
    direction.

**Ambiguous Schema (Overturn or Gold
    Fault)**
When the prediction uses semantically
    similar but incorrect schema elements.

**IMPORTANT: This is an overturn case -
    the prediction should be rejected due
     to schema misuse.**
  Example:
    Schema: items(id, category, type)
    Q: "Count items of type 'Laptop'"
    Gold: SELECT COUNT(*) FROM items
    WHERE type='Laptop';
    Unacceptable Pred: SELECT COUNT(*)
    FROM items WHERE category='Laptop';
    Why: Pred wrongly applies the filter
    to `category` instead of `type`.

    Schema: players(id, position, rank)
    Q: "Find the player with the highest
    rank."
    Gold: SELECT * FROM players WHERE
    rank = 1;
    Unacceptable Pred: SELECT * FROM
    players WHERE position = 1;
    Why: Pred wrongly applies the filter
    to `position` (initial position)
    instead of `rank` (final rank).

    Schema: orders, purchases
    Q: "Get the total sales from all
    orders."
    Gold: SELECT SUM(total_amount) FROM
    orders;
    Unacceptable Pred: SELECT SUM(
    total_amount) FROM purchases;
    Why: Pred wrongly applies the query
    to `purchases` instead of `orders`.

    Schema: users, customers
    Q: "List all user emails."
    Gold: SELECT email FROM users;
```

```
    Unacceptable Pred: SELECT email FROM
     customers;
    Why: Pred wrongly uses `customers`
     instead of `users`.

**Ambiguous Question (Uphold - verdict=
    false)**
When the question allows multiple
    reasonable interpretations, leading
    to different but valid logic.
**IMPORTANT: This is a non-overturn case -
     the prediction should be upheld.**
**Note: Tie-handling differences are NOT
    considered ambiguous question cases
    .**
  Example:
    Q: "What is the employee's salary for
     this year?"
    Acceptable Gold: SELECT SUM(salary)
     FROM employee_salary WHERE
     employee_id = 1 AND year = 2023;
    Acceptable Pred: SELECT salary FROM
     employee_salary WHERE employee_id = 1
      AND month = 12 AND year = 2023;
    Why: The question can be interpreted
     as total salary for the year or
     December's salary.

    Q: "How did the store perform this
     year?"
    Acceptable Gold: SELECT SUM(profit)
     FROM store_performance WHERE store_id
      = 1 AND year = 2023;
    Acceptable Pred: SELECT COUNT(
     DISTINCT customer_id) FROM
     store_visits WHERE store_id = 1 AND
     year = 2023;
    Why: The question could refer to
     total profit or total customers, both
      valid measures.

    Q: "How many products have more than
     10 units sold?"
    Acceptable Gold: SELECT product_name
     FROM sales WHERE units_sold > 10;
    Acceptable Pred: SELECT product_name
     FROM sales GROUP BY product_name
     HAVING SUM(units_sold) > 10;
    Why: The question can be understood
     as checking individual sales or
     grouping by product.

    Schema: flights(flight_id, airline,
     flight_number, aircraft_id), aircraft
     (aircraft_id, tail_number, model)
    Q: "What is the number for flight '
     BA123'?"
    Acceptable Gold: SELECT a.tail_number
      FROM flights f JOIN aircraft a ON a.
     aircraft_id = f.aircraft_id WHERE f.
     flight_number = 'BA123';
    Acceptable Pred: SELECT f.
     flight_number FROM flights f WHERE f.
     flight_number = 'BA123';
    Why: "Number" could mean tail number
     or flight number.

    Schema: orders(order_id, status,
     shipment_id), shipments(shipment_id,
     status, carrier)
    Q: "What is the status for order
     12345?"
    Acceptable Gold: SELECT o.status FROM
      orders o WHERE o.order_id = 12345;
    Acceptable Pred: SELECT s.status FROM
      orders o JOIN shipments s ON s.
     shipment_id = o.shipment_id WHERE o.
     order_id = 12345;
    Why: "Status" could mean the order's
     status or the shipment's status

**Gold Fault (Uphold - verdict=false)**
Avoid labeling as gold fault unless
    absolutely certain.
  Example:
    Q: "City of user with id=5"
    Unacceptable Gold: SELECT city FROM
     users WHERE id=6;
    Acceptable Pred: SELECT city FROM
     users WHERE id=5;
    Why: Gold mis-specifies the
     requirement; pred matches the
     question.

### Output JSON (field order is mandatory)

Use concise language. No extra fields.
    Always emit keys in this exact order:
1. `judgement` - concise one-sentence
    assessment grounded in semantic logic
     (not syntax).
2. `verdict` - boolean: `true` = overturn
     Prover's pass; `false` = uphold.
3. `ambiguity` - string indicating
    ambiguity type: `"ambiguous question
    "`, `"ambiguous schema"`, `"na"`, or
    combinations like `"ambiguous
    question, ambiguous schema"`.
4. `gold_correct` - boolean: `true` =
    gold standard is correct; `false` =
    gold standard has faults.

Important: Return ONLY the JSON object
    with no additional text. `verdict`
    must be a JSON boolean (true/false
    without quotes). Output keys strictly
     in the specified order.
"""

user_prompt_refuter = """
###### Instructions
Compare the prediction against the gold
    and decide whether to overturn the
    Prover's pass.

Follow this process:
1. First, observe differences: examine
    SQL syntax and execution result
    differences between prediction and
    gold standard. Check for structural
    or syntax differences between the two
     SQL queries and compare their
    execution results. If results differ,
     note the specific discrepancy.
2. Then, analyze semantics: understand
    what each query actually means in
```

```
    answering the question. Check if the
    SQL queries are logically correct and
     aligned with the question's goal.
    Examine whether the queries are
    trying to accomplish the same thing,
    such as filtering or joining tables
    to provide a correct answer to the
    question. Ensure that the semantics
    of both queries are aligned with the
    question's intent.
3. Next, classify the cause: determine if
     differences stem from ambiguous
    schema or ambiguous question (valid
    alternative interpretations). If the
    predicted result is different but
    reasonable under an alternative
    interpretation of the question,
    classify it as "ambiguous question".
    If the error in either the predicted
    or gold query is due to the schema
    being too similar, classify it as "
    ambiguous schema". If no ambiguity is
     found, classify it as "na".
4. Finally, apply decision: based on the
    analysis, provide the judgement and
    verdict. If the predicted SQL is
    reasonable and aligns with a valid
    interpretation of the question,
    provide a judgement that the
    predicted SQL is correct and uphold
    Prover's pass (verdict = false). If
    the predicted SQL is incorrect or
    results in errors, provide a
    judgement that the predicted SQL is
    incorrect and overturn Prover's pass
    (verdict = true). Assess the
    correctness of the gold standard (
    gold_correct = true if gold SQL is
    correct, false otherwise).

Return ONLY the JSON object directly.

###### Question
{question}

###### Evidence
{evidence}

###### Database Information
{db_info}

###### Predicted SQL
{predicted_sql}

###### Predicted SQL Execution Result
{pred_result}

###### Gold Standard SQL
{gold_sql}

###### Gold SQL Execution Result
{gold_result}

###### Prover's Reasoning
{prover_reason}
"""

user_prompt_refuter_without_results = """
```

```
###### Instructions
Act as a lenient-but-principled Refuter.
    Compare the prediction against the
    gold and decide whether to overturn
    the Prover's pass.
Execution results are not provided
    because the gold and predicted SQL
    produce identical results.

** IMPORTANT: Your default stance is to
    UPHOLD the Prover.**

**Overturn only if:**
- An explicit requirement is violated or
    an explicit filter is missing.
- An added predicate narrows the set on
    an unrelated attribute not entailed
    by the question/evidence/schema.

**Still uphold when:**
- Equivalent logic with different
    implementation.
- Extra NOT NULL on the projected column
    that does not change the intended
    selection.
- Omitting NOT NULL is always acceptable
    unless explicitly required by the
    evidence/question.
- Alternative join paths.
- Projection/order/alias differences
- Presence/absence of tie-breakers when
    not specified.

**Examples**
- Uphold:
  Q: "Show the regions of suppliers who
     delivered goods in March 2021."
  Gold: SELECT DISTINCT s.region FROM
     deliveries d JOIN suppliers s ON d.
     supplier_id = s.supplier_id WHERE
     strftime('%Y-%m', d.delivered_at) =
     '2021-03';
  Pred: SELECT DISTINCT s.region FROM
     deliveries d JOIN suppliers s ON d.
     supplier_id = s.supplier_id WHERE d.
     delivered_at >= '2021-03-01' AND d.
     delivered_at < '2021-04-01';
  Why: Time restriction is equivalent via
      month extraction vs month range.

- Uphold:
  Q: "List artists born in July 1985."
  Gold: SELECT artist_name FROM artists
     WHERE SUBSTR(birthdate, 1, 7) =
     '1985-07';
  Pred: SELECT artist_name FROM artists
     WHERE strftime('%Y', birthdate) =
     '1985' AND strftime('%m', birthdate)
     = '07' AND artist_name IS NOT NULL;
  Why: Year/month filtering is equivalent;
      the extra NOT NULL on the projected
     name is benign absent evidence it
     excludes valid answers.

- Overturn:
  Q: "Email of user with id=42"
  Gold: SELECT email FROM users WHERE id
     = 42;
```

```
  Pred: SELECT email FROM users WHERE id
     = 42 AND email_verified = 1;
  Why: Adds an unjustified predicate on
     an unrelated attribute (verification)
     , potentially excluding valid answers
     ; contradicts the question's scope.

Return ONLY the JSON object directly.

###### Question
{question}

###### Evidence
{evidence}

###### Database Information
{db_info}

###### Predicted SQL
{predicted_sql}

###### Gold Standard SQL
{gold_sql}
"""
```