PremiseSQL: Evaluating Premise Critique and Abstention in Text-to-SQL Systems

Anonymous ACL submission

Abstract

Recent advances in large language models (LLMs) have substantially improved Text-to-SQL performance, yet existing NL2SQL systems largely assume that user queries are valid and answerable. As a result, models often generate executable SQL even for queries with flawed premises. We introduce *premise critique ability* in NL2SQL, namely the ability to recognize flawed queries and abstain from generating misleading SQL. To evaluate this capability, we propose **PremiseSQL**, a benchmark containing 1,800 flawed NL2SQL queries across four categories of premise failures. We further design a three-stage evaluation framework to measure both intrinsic and prompt-induced premise-aware behavior. Experiments on frontier LLMs show that current models are strongly premise-trusting by default and rarely critique flawed premises under standard prompting conditions. Code and benchmark are available at `https://anonymous.4open.science/r/premisesql/`.

1 Introduction

Text-to-SQL (or NL2SQL) translates natural language (NL) questions into executable SQL queries given a database schema (Popescu et al., 2003). Recent advances in large language models (LLMs) have substantially improved schema understanding, compositional reasoning, and cross-domain generalization in modern NL2SQL systems (OpenAI et al., 2024; Touvron et al., 2023; Yang et al., 2025; AI, 2025; Chang et al., 2024; Scholak et al., 2021; Yu et al., 2018; Pourreza and Rafiei, 2023).

Despite this progress, existing Text-to-SQL research largely assumes that user questions are valid, well-formed, and answerable under the given schema (Yu et al., 2018; Li et al., 2023). Consequently, most prior work focuses on improving SQL generation accuracy, schema linking, and complex reasoning capabilities (Wang et al., 2020; Scholak et al., 2021; Pourreza and Rafiei, 2023).However, this assumption is frequently violated in realistic scenarios. User questions may contain logically inconsistent conditions, false schema assumptions, missing premises, or requests beyond the capability of SQL and the underlying database.

Nevertheless, state-of-the-art NL2SQL models often generate executable SQL even for flawed inputs, producing queries that are syntactically valid but semantically invalid (Ji et al., 2023; Whitehead et al., 2022; Ren et al., 2023; Huang et al., 2025). In many cases, models implicitly repair problematic questions by injecting unsupported assumptions in order to preserve generation fluency (Lin et al., 2022). For example, when asked to identify a patient who is simultaneously male and female, a model may generate a query containing contradictory predicates such as `gender='Male' AND gender='Female'`. Although executable, such queries fail to recognize that the original premise itself is flawed. Figure 1 illustrates the contrast between typical NL2SQL behavior and the desired premise critique behavior under flawed user queries.

To address this issue, we introduce the concept of *premise critique ability* in NL2SQL, defined as a model's ability to determine whether a query is valid before generating SQL. Premise critique ability is orthogonal to conventional SQL generation accuracy: a model may produce executable or even execution-correct SQL while still failing to recognize that the original question itself is ill-posed. Concretely, it consists of two components: (1) *premise validation*, namely identifying whether a query contains flawed assumptions or invalid conditions; and (2) *abstention*, namely refraining from generating misleading SQL when the query is fundamentally problematic.

To systematically evaluate this capability, we introduce **PremiseSQL**, a benchmark for flawed-premise reasoning in NL2SQL systems. Built upon two widely used Text-to-SQL datasets, Spider 1.0

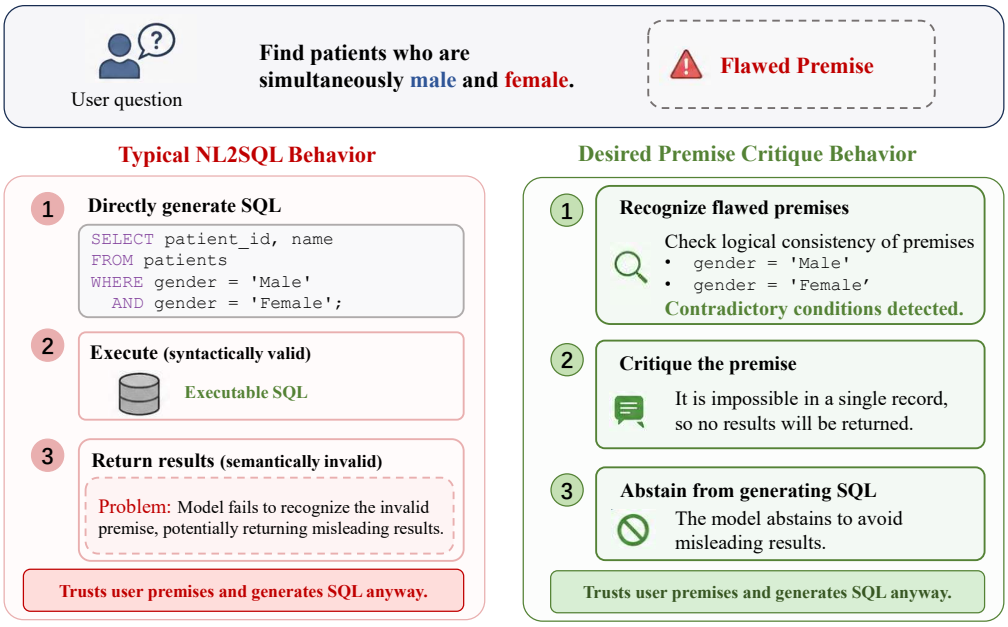

Figure 1: Current NL2SQL systems often generate executable SQL even when user queries contain flawed premises. In the illustrated example, the model produces a syntactically valid query despite the logically inconsistent condition that a patient is simultaneously male and female. Premise critique evaluation aims to measure whether models can recognize such flawed premises and abstain from generating misleading SQL.

(Yu et al., 2018) and BIRD / BIRD-filtered (Li et al., 2023), PremiseSQL contains 1,800 constructed flawed queries spanning four major categories of premise failures: expression errors, premise contradictions, missing premises, and queries beyond SQL capability. Each category is further divided into fine-grained subtypes.

PremiseSQL explicitly evaluates whether models can recognize flawed premises and abstain from generating misleading SQL. Furthermore, we evaluate models under three progressively premise-aware prompting settings, namely Base, Implicit, and Explicit, allowing us to systematically measure both intrinsic premise critique behavior and prompt-induced improvements. PremiseSQL is the first benchmark specifically designed for jointly evaluating premise validation and abstention behavior in frontier NL2SQL systems.

In summary, our contributions are as follows:

- We formalize *premise critique ability* in NL2SQL, consisting of premise validation and abstention.
- We introduce **PremiseSQL**, a benchmark of 1,800 flawed NL2SQL queries spanning four premise failure categories and nine subtypes.
- We propose a multi-stage evaluation framework for measuring premise critique behavior under different prompting settings.
- We conduct extensive experiments on frontier LLMs and show that current NL2SQL systems are strongly premise-trusting by default.

2 Related Work

2.1 Unanswerable and Ambiguous Queries in NL2SQL

Prior work has explored unanswerable and ambiguous queries in NL2SQL. TriageSQL (Zhang et al., 2020) formulates the problem as question intention classification, while Know What I Don't Know (Wang et al., 2023) further categorizes ambiguity and unanswerability cases. More recent work extends beyond classification-oriented formulations. TrustSQL (Lee et al., 2024) introduces infeasible queries and proposes a reliability-oriented evaluation metric for trustworthy NL2SQL systems. Complementarily, NL2SQL-BUGs (Liu et al., 2025) categorizes semantic SQL generation errors into fine-grained types such as attribute, table, operator, and condition errors.

However, existing approaches mainly focus on answerability prediction, ambiguity resolution, or

Method	FP	Abst.	SQL	Fine	Prompt
TriageSQL	✓	×	✓	△	×
KWIDK	✓	×	✓	✓	×
TrustSQL	△	×	✓	△	×
NL2SQL-BUGs	×	×	✓	✓	×
SQuAD 2.0	✓	×	×	×	×
AbstentionBench	✓	✓	×	△	△
PremiseSQL	✓	✓	✓	✓	✓

Table 1: Comparison with related work on flawed-premise and abstention evaluation. FP denotes explicit flawed-premise reasoning before SQL generation, Abst. denotes abstention evaluation, SQL indicates whether the benchmark is formulated around SQL generation tasks, Fine denotes fine-grained error taxonomy, and Prompt denotes prompt-aware evaluation settings. Symbols: ✓ Fully supported, △ Partially supported, × Not supported.

SQL error analysis, rather than whether models should abstain from generating SQL for flawed queries.

2.2 Robustness and False Premise Handling in LLMs

Beyond NL2SQL, prior work studies LLM robustness under adversarial, noisy, or logically flawed inputs. For example, SQuAD 2.0 (Rajpurkar et al., 2018) introduces unanswerable questions, highlighting models' tendency to produce confident but incorrect responses when no valid answer exists. Similarly, AbstentionBench (Kirichenko et al., 2025) systematically studies abstention under false premises and insufficient information.

These studies show that LLMs often generate confident responses despite unsupported assumptions or insufficient evidence (Ren et al., 2023; Huang et al., 2025). However, existing semantic parsing research still mainly evaluates SQL correctness rather than whether SQL should be generated at all.

Summary To better position PremiseSQL, we compare it with representative benchmarks in Table 1.

3 Method

3.1 Task Formulation

Let an input be $I = (x, s)$, where x is a natural language query and s is the database schema. We associate x with a set of implicit premises $P = \{p_1, \ldots, p_n\}$ required for the query to be valid.

We define a flaw detection function:

$$F(P) \subseteq P \tag{1}$$

The query is flawed if $F(P) \neq \emptyset$.

A valid critique identifies some flaw $p \in F(P)$. Let $\mathcal{C}(I)$ denote the set of valid critiques for input I.

Given model output $O = M(I)$, the model exhibits premise critique ability if:

$$\text{Components}(O) \cap \mathcal{C}(I) \neq \emptyset \tag{2}$$

We further define abstention as refraining from generating SQL when $F(P) \neq \emptyset$.

3.2 Premise Taxonomy

We further organize flawed premises into a unified taxonomy. A complete taxonomy with fine-grained subcategories and definitions is shown in Table 2.

(1) Expression Error refers to queries whose linguistic expression admits multiple plausible interpretations, preventing a unique or well-defined SQL formulation.

(2) Premise Contradiction captures queries whose assumptions are logically inconsistent or incompatible with the database schema.

(3) Missing Premise refers to queries that omit essential assumptions, constraints, or schema information required for reliable query execution.

(4) Beyond Ability includes queries whose requirements exceed the expressive or reasoning capabilities of SQL and the given database, making them unanswerable under the available execution assumptions.

3.3 Benchmark Construction

Our benchmark is built upon Spider 1.0 (Yu et al., 2018) and BIRD / BIRD-filtered (Li et al., 2023). Starting from standard Text-to-SQL queries, we construct flawed variants using taxonomy-aligned perturbations that modify or remove required premises. Each perturbed query is verified to satisfy the flawed-input condition ($F(P) \neq \emptyset$). Specifically, Subcategory 3.2 is constructed using BIRD instead of BIRD-filtered, as it requires access to gold schema annotations for schema-sensitive perturbations. All other subcategories are built on BIRD-filtered to ensure consistency.

To improve diversity and linguistic naturalness, perturbations are generated using **GPT-5.1** under predefined error constraints. All instances are assigned corresponding error categories and subcategories, and further verified using a two-stage process consisting of automatic evaluation by **Gemini-2.5-Pro-Thinking** and subsequent human review.

Code	Category	Subcategory	Description
E1	Expression Error	E1.1 Ambiguity	Query contains ambiguous references, scopes, or targets that admit multiple plausible SQL interpretations.
		E1.2 Unclear Intent	Query intent or operation type is underspecified, making the SQL structure indeterminate.
E2	Premise Contradiction	E2.1 Internal Text Contradiction	Multiple query conditions cannot logically hold simultaneously.
		E2.2 Question–Schema Contradiction	Required schema elements or relations are absent or incompatible with the database schema.
E3	Missing Premise	E3.1 Missing Constraint	Essential filtering conditions or target specifications are omitted.
		E3.2 Missing Schema Premise	Required tables, columns, or relations are not provided in the schema.
		E3.3 Semantic Gap	Query concepts cannot be semantically mapped to schema elements.
E4	Beyond Ability	E4.1 Inexpressible in SQL	Query intent cannot be precisely represented using standard SQL.
		E4.2 External Knowledge Required	Correct answering requires knowledge absent from the database.

Table 2: Taxonomy of flawed premises in Text-to-SQL.

The final benchmark contains 1,800 queries spanning 4 premise failure categories and 9 subcategories.

3.4 Prompting Protocol

To analyze premise critique behavior under different levels of premise-awareness guidance, we design a three-stage prompting framework consisting of the *Base*, *Implicit*, and *Explicit* settings. All 1,800 benchmark instances are evaluated independently under each setting using the same query set, resulting in a total of **5,400 model-query instances** per model.

Under this framework, premise critique behavior is evaluated from two complementary perspectives: (1) *Premise Validation*, namely whether the model can recognize that a user query is logically or semantically flawed; (2) *Abstention*, namely whether the model avoids generating misleading SQL under such conditions.

Base Setting The Base setting follows the standard Text-to-SQL generation setup. Models are simply instructed to generate SQL for the given question and return a structured JSON output containing *sql* and *reason*, where *sql* denotes the generated query (or null when abstaining) and *reason* records the model's explanation for its generation behavior. No additional warning is provided regarding potential flaws in the query.

Implicit Setting The Implicit setting introduces implicit premise-awareness guidance. Models are informed that the input question may contain errors and are asked to assess its validity before answering. The output JSON includes *sql*, *is_valid*, and *reason*, where *is_valid* indicates whether the question is logically valid and answerable. This setting measures whether soft prompting can improve premise validation and abstention behavior.

Explicit Setting The Explicit setting provides direct critique-oriented instructions. Models are explicitly told that the question contains logically flawed or incorrect premises, and are required to critically examine its assumptions before responding. The output format remains *sql*, *is_valid*, and *reason*. This setting measures the extent to which explicit critique-oriented instructions activate premise-aware reasoning behavior.

3.5 Evaluation Framework

Evaluation Metrics We evaluate premise critique ability using three core metrics: *SQL Behavior*, *Detection*, and *Type Identification*. For the Implicit and Explicit settings, we additionally include a validity judgment metric based on the model-predicted *is_valid* field. All evaluations are conducted using structured rule-based criteria with additional LLM-assisted verification.

SQL Behavior measures the degree of abstention from generating misleading SQL. **Detection**

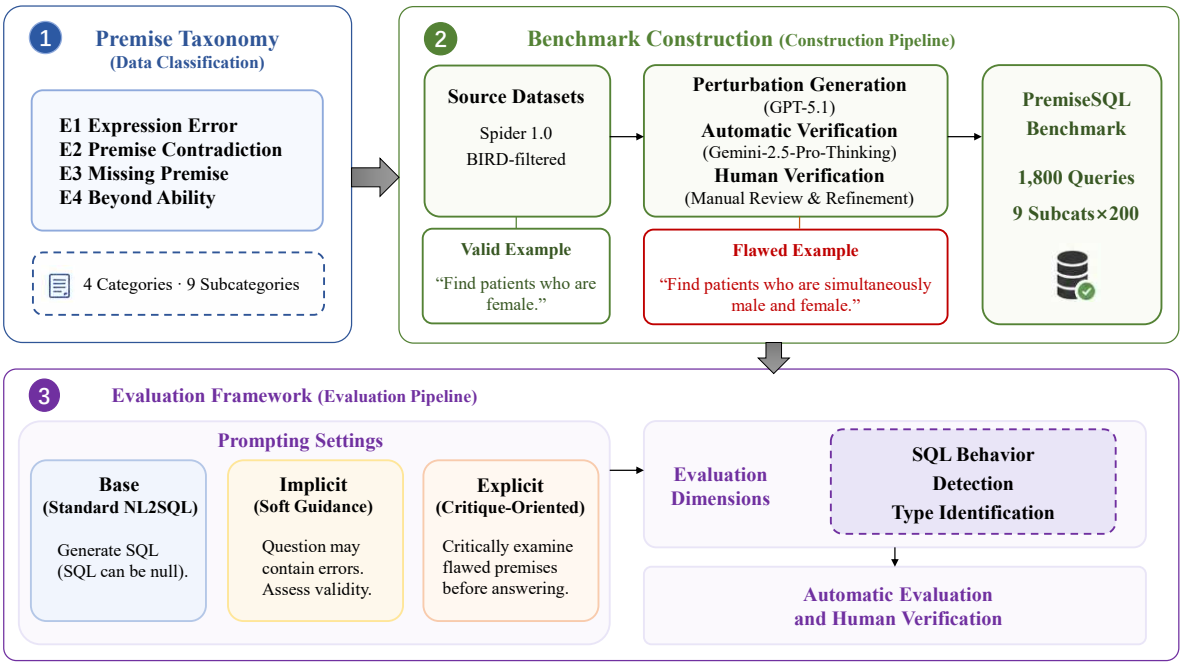

Figure 2: Overview of PremiseSQL. Starting from standard Text-to-SQL queries, we construct flawed premise variants under a unified taxonomy, followed by multi-stage verification and premise-aware evaluation under Base, Implicit, and Explicit settings.

evaluates whether the model recognizes that the query contains flawed premises. **Type Identification** measures whether the model correctly identifies the underlying premise failure type. Detailed scoring criteria are provided in Appendix A.

Validity Signal Analysis For the Implicit and Explicit settings, models additionally predict an *is_valid* field indicating whether the query is logically valid and answerable. Since all benchmark queries are intentionally flawed, the expected prediction is always *false*.

Instead of treating this field as a standalone evaluation metric, we use it as an auxiliary signal to analyze whether internal validity judgments are associated with stronger premise critique behavior. Specifically, we examine whether predicting *is_valid* = *false* correlates with improved detection, more accurate type identification, and safer SQL behavior.

Automatic Evaluation All model outputs are evaluated using LLM-based automatic evaluators under a unified scoring protocol. Given a model response, the evaluator assigns labels for *SQL Behavior*, *Detection*, and *Type Identification* according to the criteria defined above.

To reduce evaluator-specific bias, we employ multiple independent evaluation models and apply the same evaluation instructions and scoring definitions across all experiments. Final scores are computed by averaging the outputs from all evaluators.

Human Verification To further validate the reliability of automatic evaluation, we additionally conduct human verification on a subset of evaluation instances. Human annotators independently assess model outputs using the same evaluation criteria as the automatic evaluators.

Detailed agreement analysis and evaluation consistency results are reported in Section 4.2.4.

4 Experiments

4.1 Experimental Setup

We evaluate premise critique ability on a diverse set of state-of-the-art LLMs. The evaluated models include *ChatGLM-5.1* (Team, 2026), *Claude Opus 4.6* (Anthropic, 2026b), *Claude Sonnet 4.6* (Anthropic, 2026a), *DeepSeek V3.2* (AI, 2025), *Doubao Seed 2.0 Pro* (ByteDance, 2026), *Gemini 3.1 Pro Preview* (Google, 2026), *LLaMA 4 Maverick* (AI, 2026), *GPT-5.4* (OpenAI, 2026), and three scales of the *Qwen3.5* (Qwen Team, 2026d) reasoning model family (*35B* (Qwen Team, 2026b), *122B* (Qwen Team, 2026a), and *397B* (Qwen Team, 2026c)).

All models are evaluated under the Base, Implicit, and Explicit prompting settings using the same benchmark and evaluation protocol. Evaluations use deterministic decoding with temperature 0 and default API configurations.

For automatic evaluation, we use GPT-5.4 and Claude Opus 4.6 as independent judges.

4.2 Main Results

4.2.1 Overall Results

Table 3 summarizes the overall performance of all evaluated models under the three prompting settings. Performance improves substantially from Base to premise-aware prompting settings, indicating that current Text-to-SQL models rarely critique flawed premises under standard generation objectives.

Base Setting Analysis. Under the Base setting, most models continued generating SQL even when the input queries contained invalid or contradictory premises. Detection performance remained consistently low across evaluated models, indicating that models rarely challenged user assumptions under standard NL2SQL generation objectives. For example, GPT-5.4 achieved only 25.31% Detection accuracy, while LLaMA 4 Maverick achieved 14.69%. Similarly, SQL Behavior scores also remained low, suggesting that most models continued to produce executable or partially executable SQL while recognizing little or no premise inconsistency. These results indicate that current NL2SQL systems are strongly biased toward completing SQL generation rather than critically validating user premises.

Implicit Setting Analysis. Introducing implicit premise-awareness guidance substantially improves model performance across nearly all evaluated systems, with notable gains in Detection and abstention behavior. For instance, GPT-5.4 improves from 25.31% to 68.44% in Detection, while DeepSeek V3.2 reaches 89.06% Detection accuracy and 89.64% SQL Behavior performance.

Although all benchmark queries are deliberately flawed, the Implicit setting only provides a soft warning that inputs *may* contain errors, without explicitly requiring critique. Under this condition, models still achieve only 66.21% average Detection, suggesting that premise critique capability is partially latent but not reliably activated under weak prompting signals.

Explicit Setting Analysis. The Explicit setting further pushed performance close to saturation for several frontier models. When explicitly instructed to critically examine user assumptions, most models achieved Detection scores above 95%, accompanied by substantial improvements in SQL abstention behavior. GPT-5.4 reached 97.86% Detection accuracy and 94.63% SQL Behavior performance, while Qwen3.5-397B achieved nearly perfect abstention behavior with a SQL Behavior score of 99.72%. These results demonstrate that current frontier LLMs are capable of strong premise critique reasoning when critique-oriented behavior is explicitly activated.

Cross-Setting Observations Despite the substantial gains in Detection performance, Type Identification consistently remained more challenging across all settings. Many models could recognize that a query was problematic, yet still struggled to accurately characterize the underlying flaw type. Even under the Explicit setting, Type Identification scores generally lagged behind Detection performance by a noticeable margin. This suggests that premise critique ability is inherently hierarchical: detecting that a premise is flawed is substantially easier than correctly diagnosing why it is flawed.

We also observe clear behavioral differences across evaluated models. DeepSeek V3.2 demonstrated particularly strong premise-aware behavior under both the Implicit and Explicit settings, outperforming several proprietary frontier models in Detection and SQL abstention metrics. In contrast, some highly capable general-purpose models continued generating SQL despite partially recognizing premise inconsistencies under soft prompting conditions. This suggests that strong general reasoning ability alone does not necessarily guarantee reliable premise-aware behavior in NL2SQL tasks.

Overall, the results show a consistent pattern across models and prompting settings: LLMs can critique flawed premises, but this ability is often under-activated under standard NL2SQL objectives. To examine this behavior, we analyze samples where the model explicitly exhibits error awareness (is_valid = False). Across 32,970 such samples in the Implicit and Explicit settings, 31,660 are detected as flawed (96.02%), indicating strong agreement under the unified automatic evaluation protocol and suggesting high consistency between model-based detection and the assigned invalid labels. Similarly, 31,484 samples (95.49%)

Model	Base			Implicit			Explicit		
	SQL	DET	TYPE	SQL	DET	TYPE	SQL	DET	TYPE
ChatGLM-5.1	0.117	0.225	0.219	0.573	0.580	0.550	0.985	0.956	0.766
Claude Opus 4.6	0.198	0.273	0.269	0.518	0.559	0.513	0.898	0.903	0.755
Claude Sonnet 4.6	0.170	0.245	0.240	0.372	0.451	0.418	0.855	0.874	0.712
DeepSeek-V3.2	0.212	0.399	0.377	0.896	0.891	0.756	0.979	0.977	0.776
Doubao Seed 2.0 Pro	0.161	0.301	0.295	0.670	0.663	0.601	0.948	0.927	0.786
Gemini 3.1 Pro Preview	0.142	0.264	0.264	0.682	0.682	0.663	0.936	0.926	0.832
LLaMA 4 Maverick	0.098	0.147	0.141	0.511	0.586	0.497	0.973	0.973	0.749
GPT-5.4	0.111	0.253	0.250	0.664	0.684	0.628	0.946	0.979	0.844
Qwen3.5-35B	0.190	0.400	0.384	0.814	0.793	0.685	0.996	0.970	0.783
Qwen3.5-122B	0.134	0.337	0.327	0.692	0.694	0.626	0.990	0.956	0.782
Qwen3.5-397B	0.048	0.305	0.303	0.662	0.701	0.660	0.997	0.969	0.821

Table 3: Overall premise critique performance across different prompting settings. SQL denotes a normalized score in $[0, 1]$. Darker teal indicates higher performance.

are assigned sql_behavior = 2, indicating that the model tends to abstain from SQL generation once it identifies potential errors or uncertainty in the input. This suggests that premise-level critical reasoning directly governs downstream generation: when activated, error recognition reliably leads to abstention. More broadly, premise critique ability depends more on behavioral activation than raw reasoning capacity, and strong SQL performance does not necessarily imply reliable premise-aware reasoning, revealing a key reliability gap in current NL2SQL systems.

4.2.2 Error-Type Analysis

Error Type	SQL	DET	TYPE
Ambiguity	0.224	0.237	0.121
Unclear Intent	0.254	0.234	0.155
Text Contradiction	0.964	0.978	0.972
Schema Contradiction	0.877	0.916	0.906
Missing Constraint	0.494	0.503	0.411
Missing Schema	0.526	0.562	0.421
Semantic Gap	0.704	0.743	0.713
Inexpressible SQL	0.856	0.893	0.825
External Knowledge	0.874	0.893	0.874

Table 4: Performance across premise error categories under the Implicit setting. SQL scores are normalized to $[0, 1]$.

Table 4 and Figure 3 show substantial variation across premise error categories.

Overall, models perform much better on explicit logical inconsistencies than on implicit semantic insufficiency or underspecified intent. Contradiction-based errors are the easiest to identify, with both

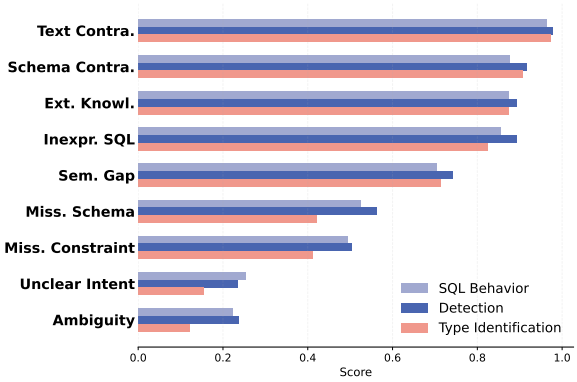

Figure 3: Comparison of performance across premise error categories. Categories are sorted by SQL Behavior score.

text and schema contradictions achieving over 90% Detection accuracy. In contrast, ambiguity and unclear-intent cases remain highly challenging, with Detection scores around 23%, indicating persistent difficulty in recognizing underspecified semantics without explicit logical inconsistency.

Missing premise subcategories show intermediate performance, while Beyond-Ability cases are comparatively easier under premise-aware prompting. Overall, current Text-to-SQL systems rely more on explicit inconsistency detection than deeper semantic premise analysis.

4.2.3 Failure Pattern Analysis

We observe four common failure patterns across models: (1) recognition without abstention, where models identify premise flaws but still generate SQL; (2) assumption injection, where missing

Type	Question	Model Behavior	Failure Pattern
E2.2	How many planes of Spirit Air Lines took off on 2018/8/7 from airports located in California?	Model detects that the schema does not contain Spirit Air Lines or California airports and refuses SQL generation.	Correct abstention
E2.1	Give the SSN of the patient who is simultaneously male and female, is Irish, and allergic to grass pollen.	Model explicitly states that the condition is impossible, but still generates SQL with contradictory predicates: `gender='Male' AND gender='Female'`.	Recognition without abstention
E3.3	What are the language and title of the ordered books that qualify as low-cost compared to the average price of all ordered books?	Model implicitly defines "low-cost" as prices below the global average order price, despite the comparison criterion being underspecified.	Assumption injection
E2.1	List the last name of all clients who made a PS-type complaint that was both served by TOVA and not served by TOVA.	Model reformulates the contradiction at the client level, requiring separate complaints to satisfy each condition independently rather than preserving the contradiction on a single complaint instance.	Contradiction reinterpretation
E2.3	What is the difference in the number of goals scored by Pavel Brendl during the regular season versus the playoffs in the season?	Model incorrectly assumes that the provided sample rows represent complete database coverage and fails to recognize missing data scope, leading to inability to abstain or generate valid SQL.	capacity limitation

Table 5: Representative failure patterns observed in PremiseSQL.

semantics are implicitly fabricated to preserve answerability; (3) contradiction reinterpretation, where contradictory conditions are relaxed into satisfiable alternatives; and (4) capacity limitation, where models fail to reason over implicit dataset scope or completeness constraints. Representative examples are shown in Table 5.

These behaviors suggest that current NL2SQL systems remain strongly biased toward preserving answerability even when premise flaws are partially recognized.

4.2.4 Evaluation Reliability

We evaluate reliability through inter-evaluator agreement and human verification.

Inter-Evaluator Agreement To assess the consistency of automatic evaluation, we measure inter-evaluator agreement between **GPT-5.4** and **Claude Opus 4.6** using Cohen's Kappa. Agreement reaches 0.9743 for SQL Behavior, 0.9042 for Detection, and 0.8283 for Type Identification, indicating strong consistency across evaluators.

Human Verification We further conduct human verification on 2,160 randomly sampled outputs from one open-weight model (**DeepSeek-V3.2**) and one proprietary model (**GPT-5.4**) across different prompting settings and error categories. Human annotators follow the same evaluation protocol as the automatic evaluators.

For outputs evaluated by **Claude Opus 4.6**, the Kappa scores are **0.9893**, **0.9666**, and **0.8768** for SQL Behavior, Detection, and Type Identification, respectively. For **GPT-5.4**, the corresponding scores are **0.9817**, **0.9630**, and **0.8815**. These results demonstrate strong consistency between automatic and human evaluation, supporting the reliability of the proposed benchmark and evaluation framework.

5 Conclusion

In this work, we introduce **PremiseSQL**, the first benchmark for evaluating premise critique ability in NL2SQL systems. PremiseSQL contains 1,800 invalid queries covering 9 fine-grained error types. Experiments under Base, Implicit, and Explicit evaluation settings reveal a clear reliability gap in current NL2SQL systems. Models often trust user assumptions and generate misleading SQL under standard settings. Although critique-oriented prompting improves detection and abstention, models still struggle with subtle and implicit premise failures, showing that strong SQL generation does not necessarily imply trustworthy premise-aware reasoning. We hope PremiseSQL can facilitate future research on more reliable and premise-aware NL2SQL systems.

Limitations

Our work has several limitations. First, PremiseSQL focuses on English NL2SQL benchmarks and may not generalize to multilingual settings or real-world interactive database systems. Second, although the benchmark covers multiple premise failure categories, the constructed errors are still taxonomy-driven and may not fully capture naturally occurring user mistakes. Third, our evaluation relies partially on LLM-based automatic judges, which may introduce evaluation bias despite strong agreement with human annotations. Finally, the evaluated prompting settings focus on single-turn interactions and do not consider multi-turn clarification or user feedback scenarios.

References

DeepSeek AI. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Meta AI. 2026. The llama 4 herd: Architecture, training, evaluation, and deployment notes. *Preprint*, arXiv:2601.11659.

Anthropic. 2026a. Introducing claude 4.6 sonnet. https://www.anthropic.com/news/claude-sonnet-4-6. Accessed: 2026-02.

Anthropic. 2026b. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6. Accessed: 2026-02.

ByteDance. 2026. Seed2.0 technical introduction. https://research.doubao.com/zh/seed2. Accessed: 2026-02.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3).

Google. 2026. Gemini 3.1 pro preview. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/3-1-pro. Accessed: 2026-02.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2).

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12).

Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J. Bell. 2025. Abstentionbench: Reasoning llms fail on unanswerable questions. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc.

Gyubok Lee, Woosog Chay, Seonhee Cho, and Edward Choi. 2024. TrustSQL: Benchmarking text-to-SQL reliability with penalty-based scoring.

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 C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. 2025. Nl2sql-bugs: A benchmark for detecting semantic errors in nl2sql translation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 5662–5673, New York, NY, USA. Association for Computing Machinery.

OpenAI. 2026. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/. Accessed: 2026-03.

OpenAI, Josh Achiam, Steven Adler, and et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, page 149–157, New York, NY, USA. Association for Computing Machinery.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: decomposed in-context learning of text-to-sql with self-correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Qwen Team. 2026a. Qwen3.5-122b-a10b. https://huggingface.co/Qwen/Qwen3.5-122B-A10B. Accessed: 2026-02.

Qwen Team. 2026b. Qwen3.5-35b-a3b. https://huggingface.co/Qwen/Qwen3.5-35B-A3B. Accessed: 2026-02.

Qwen Team. 2026c. Qwen3.5-397b-a17b. https://huggingface.co/Qwen/Qwen3.5-397B-A17B. Accessed: 2026-02.

Qwen Team. 2026d. Qwen3.5: Towards native multimodal agents.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J. Liu. 2023. Out-of-distribution detection and selective generation for conditional language models. *Preprint*, arXiv:2209.15558.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *Preprint*, arXiv:2109.05093.

GLM-5 Team. 2026. Glm-5: from vibe coding to agentic engineering. *Preprint*, arXiv:2602.15763.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

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

Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. 2023. Know what I don't know: Handling ambiguous and unknown questions for text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5701–5714, Toronto, Canada. Association for Computational Linguistics.

Spencer Whitehead, Suzanne Petryk, Vedaad Shakib, Joseph Gonzalez, Trevor Darrell, Anna Rohrbach, and Marcus Rohrbach. 2022. Reliable visual question answering: Abstain rather than answer incorrectly. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVI*, page 148–166, Berlin, Heidelberg. Springer-Verlag.

An Yang, Anfeng Li, Baosong Yang, and et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Yusen Zhang, Xiangyu Dong, Shuaichen Chang, Tao Yu, Peng Shi, and Rui Zhang. 2020. Did you ask a good question? a cross-domain question intention classification benchmark for text-to-sql. *arXiv preprint arXiv:2010.12634*.

A Evaluation Rubrics

SQL Behavior. Measures the degree of abstention from generating SQL for flawed queries.

- 0: generates a complete SQL query;
- 1: partial abstention or invalid SQL output;
- 2: complete abstention or explicit refusal.

Detection. Measures whether the model recognizes the query as flawed.

- 0: no detection;
- 1: detects flawed premises.

Type Identification. Measures whether the model correctly identifies the underlying flaw type.

- 0: incorrect type identification;
- 1: correct type identification.

Type Identification is evaluated only when Detection = 1.

B Dataset Composition by Error Category

PremiseSQL consists of 1,800 flawed NL2SQL queries spanning four major premise error categories and nine fine-grained subcategories. Each subcategory contains 200 examples. To preserve natural query diversity and realistic schema distributions, we do not strictly enforce identical source proportions across Spider 1.0 and BIRD / BIRD-filtered databases. The detailed statistics of PremiseSQL across error categories and source benchmarks are summarized in Table 6.

Error Category / Subcategory	Spider	BIRD
Expression Error (400)		
– Ambiguity	90	110
– Unclear Intent	107	93
Premise Contradiction (400)		
– Internal Text Contradiction	103	97
– Question–Schema Contradiction	156	44
Missing Premise (600)		
– Missing Constraint	29	171
– Missing Schema Premise	64	136
– Semantic Gap	155	45
Beyond Ability (400)		
– Inexpressible in SQL	149	51
– External Knowledge Required	140	60
Total	993	807

Table 6: Statistics of PremiseSQL across fine-grained premise error categories and source benchmarks.

C Prompt Framework

We develop a unified prompt framework covering the entire pipeline of our study, including instance generation, instance-level review, premise-critical prompting, and automatic evaluation.

Specifically, the framework consists of four components: (1) prompt templates for generating flawed instances, (2) prompt templates for reviewing and filtering generated instances, (3) prompt templates for premise-critical evaluation under different prompting settings, and (4) LLM-based automatic evaluation prompts for measuring model behavior across multiple dimensions, including SQL correctness, flaw detection, and error-type identification.

All components follow a consistent design philosophy: they explicitly define the task objective, enforce structured outputs, and incorporate category-specific constraints to ensure controllable and interpretable evaluation across different error types and experimental settings.

C.1 Prompt Template For Generating Instances

We design a unified prompt structure for error synthesis. We show one representative example for the **Ambiguity** category in Figure 4. Other categories follow the same structure with category-specific definitions.

C.2 Prompt Template For Reviewing Instances

We design a unified prompt template for automatic quality control and validation. We show one representative example for the **Ambiguity** category in Figure 5. Other categories follow the same review structure with category-specific evaluation criteria.

C.3 Prompt Templates for Premise-Critical Evaluation

We provide the prompt templates used for the three settings in our experiments: *Base*, *Implicit*, and *Explicit*. All prompts follow a unified input format consisting of the database schema and the natural language question.

Base Prompt.

The database schema is as follows: {schema}

Question: {question}

Generate SQL to answer the question.

```
{
  "sql": string or null,
  "reason": string
}
```

Return structured JSON only.

Implicit Prompt.

The database schema is as follows: {schema}

Question: {question}

The question may contain errors. You must assess its validity before answering.

Generate SQL to answer the question.

```
{
  "sql": string or null,
  "is_valid": boolean,
  "reason": string
}
```

Return structured JSON only.

Explicit Prompt.

The database schema is as follows: {schema}

Question: {question}

The question is logically flawed and contains incorrect premises. You must critically examine its assumptions. Then explain the problem.

```
{
  "sql": string or null,
  "is_valid": boolean,
  "reason": string
}
```

Return structured JSON only.

C.4 Prompt Templates for Automatic Evaluation

We use LLM-based evaluation prompts to assess model behavior under flawed-query settings. The evaluation focuses on three dimensions: *SQL Behavior*, *Detection*, and *Type Identification*.

Given a user question, model response, and the ground-truth flaw category, the evaluator determines whether the model: (1) refrains from generating invalid SQL, (2) recognizes that the question is flawed, and (3) correctly identifies the underlying flaw type.

We adopt a unified evaluation prompt structure across all error categories, with category-specific definitions and scoring criteria. Figure 6 shows a representative example for the **Ambiguity** subcategory.

D Expression Error Examples

We present representative examples of four major error categories in Text-to-SQL: Expression Error, Premise Contradiction, Missing Premise, and Beyond Ability. As shown in Table 7, these cases cover a broad spectrum of failure modes.These examples provide a systematic view of how different types of errors manifest in natural language queries.

Prompt for Ambiguity-type Generation

You are a data synthesis expert for Text-to-SQL evaluation.

Your goal is:

Given a CORRECT original Text-to-SQL question, rewrite it to deliberately introduce a specific type of error while keeping the question otherwise natural and realistic.

Background knowledge:

A necessary premise in Text-to-SQL refers to a query condition that MUST be specified in order to uniquely determine the intended SQL query under a given database schema.

Error definition:

The rewritten question contains an ambiguous description of a NECESSARY SCHEMA-LEVEL QUERY CONDITION, such that the condition may reasonably correspond to MULTIPLE columns, tables, join paths, or aggregation targets in the given schema.

Task constraints:

Rewrite the original question so that:

- EXACTLY ONE necessary query condition becomes ambiguous at the schema level
- The ambiguity leads to AT LEAST TWO different and reasonable SQL interpretations
- Each interpretation must be valid under the given schema

Allowed patterns(use ONE, include but not limited to):

- Entity reference ambiguity: A condition or object in the text may correspond to multiple columns, tables, or foreign key paths in the schema and cannot be uniquely determined.
- Aggregation ambiguity: The description of quantities, totals, averages, or other aggregation metrics is not precise.
- Time/Range ambiguity: Time or numerical ranges are not clearly specified.
- Status ambiguity: The state or category of an object is not clearly defined.
- Relationship ambiguity: The relationship between objects is not clearly defined, such as multiple possible join paths or foreign key relationships.

Strict rules:

- The rewritten question MUST contain only ONE error: schema-level ambiguity
- Do NOT invent tables or columns
- Do NOT clarify or hint at the ambiguity
- Do NOT simply paraphrase the original question

Database schema: schema

Foreign keys: {foreign_keys}

Original question: question

Gold SQL (for reference only): sql

Output: Return ONLY a JSON object with: - {rewritten_question} (string or null) - {error_category} (string or null) - {error_subcategory} (string or null) - {give_up} (boolean) - {give_up_reason} (string or null)

Figure 4: Representative prompt for ambiguity-type instance generation.

Prompt for Ambiguity-type Review

You are a STRICT dataset auditor, not a helper.
Absolute rules:
- Default decision is FAIL
- Your primary job is to FIND PROBLEMS, not to approve
- You MUST actively look for reasons to reject the sample
- If a case is borderline, unclear, or debatable, it MUST FAIL
- Being generous, charitable, or cooperative is an error
- In a healthy dataset, MANY samples should fail

Important:
You MUST first identify potential issues BEFORE deciding whether the sample passes.
If you cannot find any issue, you must explicitly explain why none exist.
Task objective:
Review whether the rewritten question correctly introduces EXACTLY ONE schema-level ambiguity, as specified by the assigned error subcategory.
Review procedure (must follow in order):
Step 1: Actively search for problems
- List ANY potential issues, weaknesses, or ambiguities that could invalidate the sample
- Even minor or borderline issues should be listed

Step 2: Decide the strongest reason to FAIL
- If the sample should be rejected, clearly state why
- If the sample should pass, explain why NO valid failure reason exists

Step 3: Decide the strongest reason to PASS
- State the single strongest argument supporting acceptance

Step 4: Checklist evaluation
Evaluate the following items as TRUE / FALSE / NULL:
- correct_error_type
- single_error_only
- error_matches_subcategory
- no_extra_tables_or_columns
- ambiguity_clarity
- sql_interpretation_ambiguity

Step 5: Final verdict
Set overall_pass = TRUE ONLY IF:
- all checklist items are clearly satisfied, and
- all identified failure concerns are convincingly refuted

Otherwise, overall_pass = FALSE.
Database schema: {schema}
Foreign keys: {foreign_keys}
Original question: {question}
Rewritten question: {rewritten_question}
Assigned error category / subcategory: {error_category} / {error_subcategory}
Gold SQL (reference only): {sql}
Give-up information:
- give_up: {give_up}
- give_up_reason: {give_up_reason}

Output:
Return ONLY a JSON object containing:
- {potential_issues}
- {strongest_reason_to_fail}
- {strongest_reason_to_pass}
- {correct_error_type}
- {single_error_only}
- {error_matches_subcategory}
- {no_extra_tables_or_columns}
- {ambiguity_clarity}
- {sql_interpretation_ambiguity}
- {overall_pass}
- {comments}

Figure 5: Representative prompt for ambiguity-type instance review.

Prompt for Ambiguity-type Evaluation

You are an expert evaluator for Text-to-SQL systems.

Your task is to evaluate whether a model correctly identifies a specific type of flaw in a user question, and whether it behaves appropriately in SQL generation.

Dataset note:

This dataset is deliberately constructed such that EVERY user question is flawed.

None of the questions has a single correct or well-defined SQL query.

The goal is NOT to produce SQL, but to evaluate whether the model can:

- recognize the flaw
- respond appropriately, including explicitly refraining from generating SQL

Ground-truth error type:

Error category: Expression Error

Subcategory: Ambiguity

Definition:

A required query condition in the user question contains ambiguity in reference, scope, or target object, such that it cannot be uniquely determined using only the given schema and context, resulting in multiple reasonable SQL interpretations.

Input:

User question: {question}

Model output:

Reason: {reason}

SQL: {sql}

Database schema: {schema}

Evaluation tasks and scoring:

1. SQL Behavior

- 0 = directly generates a full SQL query despite ambiguity
- 1 = partially abstains but still generates SQL
- 2 = correctly refrains from generating a valid SQL query

2. Detection

- 1 = explicitly states that the question is flawed
- 0 = fails to recognize the issue

3. Type Identification

If Detection = 0, then Type Identification MUST be 0.

Type Identification does NOT require explicit naming of the error type.

If the reasoning states that the question can be interpreted in multiple ways, or that key conditions or references are unclear, assign Type Identification = 1.

- 1 = reasoning clearly indicates ambiguity
- 0 = reasoning does not support the presence of ambiguity

Output format:

Return ONLY a JSON object:

```
{
  "sql_behavior": 0, 1, or 2,
  "detection": 0 or 1,
  "type_identification": 0 or 1
}
```

Figure 6: Representative prompt for ambiguity-type automatic evaluation.

Error Type	Question	Error Reason
Ambiguity	Which product had the highest sales in 2018?	The query is ambiguous and unanswerable under the given schema. The schema does not contain an explicit or derived sales attribute in the Sales Orders table. Available fields such as Order Quantity, Unit Price, and Unit Cost do not directly define a sales metric, and no clear computation rule is specified.
Unclear Intent	For the lists created by the user who rated the movie "Pavee Lackeen: The Traveller Girl" on 3/27/2011 at 2:06:34 AM, what is the situation with their followers?	The question does not clearly specify the intended query operation (e.g., filtering, aggregation, comparison), making the overall SQL structure underdefined. It also contains ambiguous references to user identity and the meaning of "followers," leading to multiple possible interpretations.
Internal Text Contradiction	List the movies that were released in 1945 and also released after the year 2000.	The question contains mutually contradictory constraints that cannot be simultaneously satisfied under the schema. A single movie record has only one release date, making it impossible for a movie to be released in both 1945 and after 2000.
Question–Schema Contradiction	List all the languages used in Europe along with the number of native speakers for each language.	The required information is not supported by the schema, as it does not define native speaker counts or any attribute that directly distinguishes native speakers. Therefore, the requested aggregation cannot be derived from the available tables and fields.
Missing Constraint	What's the percentage of coins that is higher than the price 1 hour ago in that day? List the names of these coins.	The question omits essential constraints (e.g., reference date and definition of the percentage denominator), making the result set ill-defined and not uniquely computable under the schema.
Missing Schema Premise	List the powers of Hunter Zolomon.	We intentionally removed 30% of the gold schema beforehand. Therefore, the schema does not provide a way to identify Hunter Zolomon or to link a character entity to their associated powers. There is no table or relationship connecting characters to superpowers, making the query unanswerable under the given schema.
Semantic Gap	For user ID 3, what is their yearly average number of impactful reviews?	The concept of "impactful reviews" has no explicit or implicit semantic mapping in the schema, and no business definition is provided. As a result, the metric cannot be grounded or computed from available tables or fields.
Inexpressible in SQL	Among the worst directors, which genres are they most likely to improve in over the next five years, based on their past movies and ratings?	The user intent cannot be precisely expressed within the available database schema and SQL capability, as it requires predictive reasoning, undefined concepts (e.g., "worst directors"), and unavailable cross-table consistency for linking directors, movies, and ratings.
External Knowledge Required	Which actor has acted in at least 2 films that were nominated for an Academy Award (Oscars)? Please list their IDs.	The correctness of the query depends on external knowledge not represented in the schema, as Oscar nomination information is not stored in any table or field, making the condition uncomputable within the database.

Table 7: Examples of Expression Error, Premise Contradiction, Missing Premise, and Beyond Ability types.