CLARITY: A Framework and Benchmark for Conversational Language Ambiguity and Unanswerability in Interactive NL2SQL Systems

Tabinda Sarwar, Farhad Moghimifar, Cong Duy Vu Hoang*, Xiaoxiao Ma, Shawn Chang Xu, Fahimeh Saleh, Poorya Zaremoodi, Avirup Sil, Katrin Kirchhoff
Oracle Corporation

Abstract

NL2SQL systems deployed in industry settings often encounter ambiguous or unanswerable queries, particularly in interactive scenarios with incomplete user clarification. Existing benchmarks typically assume a single source of ambiguity and rely on user interaction for resolution, overlooking realistic failure modes.

We introduce CLARITY, a framework for automatically generating an NL2SQL benchmark with multi-faceted ambiguities and diverse user behaviors across both single- and multi-turn settings. Using a constraint-driven pipeline, CLARITY transforms executable SQL into ambiguous queries, augmented with grounded conversational continuations and schema-level metadata.

Empirical evaluation on Spider and BIRD shows that leading NL2SQL systems, including those based on strong LLMs, suffer significant performance degradation under multi-faceted ambiguity. While these systems often detect ambiguity, they struggle to accurately localize and resolve the underlying schema-level sources. Our results highlight the need for more robust ambiguity detection and resolution in industry-grade NL2SQL systems.

1 Introduction

Natural language to SQL (NL2SQL) systems are increasingly being deployed as production interfaces for non-expert users to query structured data (Shi et al., 2025; Hong et al., 2025; Liu et al., 2025). Despite substantial progress driven by large language models (LLMs), real-world usage continues to present a fundamental challenge: user queries are frequently ambiguous or unanswerable, and these failure modes are amplified in interactive settings where user clarifications may be incomplete, vague, or even irrelevant (Huang et al., 2025; Pourreza and Rafiei, 2023; Talaei et al., 2024; Lee et al., 2025; Chen et al., 2025). In practice, a single query often contains multiple interacting sources of ambiguity, covering both schema columns and values, yet most existing benchmarks and evaluation protocols assume a single ambiguity per query and cooperative users who provide clean clarifications (Ding et al., 2025; Zhao et al., 2024; Vaidya et al., 2025; Saparina and Lapata, 2025).

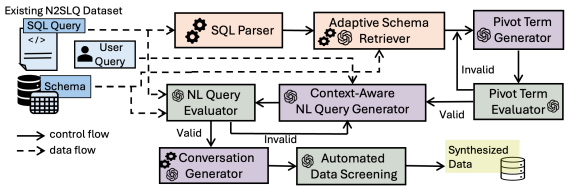

Figure 1: **Overview of the CLARITY framework**. The symbols and denote rule-based and LLM-based components, respectively.

Previous work has addressed ambiguity and unanswerability in NL2SQL, mainly in single-turn or simplified interactive settings as shown in Table 1. Benchmarks such as AMBROSIA (Saparina and Lapata, 2024), AmbiQT (Bhaskar et al., 2023), NoisySP (Wang et al., 2023) and SQUAB (Papicchio et al., 2025) target isolated ambiguity phenomena, while PRACTIQ (Dong et al., 2025), MMSQL (Guo et al., 2024), and BIRD-INTERACT (Huo et al., 2025) extend the evaluation to multi-turn scenarios. However, these benchmarks rely largely on instance-level labels (e.g., ambiguous vs. unanswerable) and do not assess whether systems correctly localize the schema elements responsible for uncertainty. Consequently, high ambiguity-detection accuracy may mask failures in schema-level identification and resolution. We argue that fine-grained, schema-grounded evaluation enables more diagnostic assessment of NL2SQL robustness, particularly under multi-faceted ambiguity and realistic interaction settings.

To address this limitation, we introduce

*Corresponding author: vu.hoang@oracle.com

Schema Entity	Use Case	AMBROSIA	AmbiQT	NoisySP	SQUAB	PRACTIQ	MMSQL	BIRD-INTERACT	CLARITY (OURS)
Type of Data	-	Single turn	Single turn	Single turn	Single turn	Multi turn	Multi turn	Multi turn	Single and multi turn
Column	Unifacet Amb	✓	✓	✓	✓	✓	✓	✓	✓
	Unifacet Unans	✓	✓	✓	✓	✓	✓	✓	✓
	Multifacet Amb	✗	✗	✗	✗	✗	✗	✗	✓
	Multifacet Unans	✗	✗	✗	✗	✗	✗	✗	✓
Value	Unifacet Amb	✗	✗	✓	✗	✓	✓	✗	✓
	Unifacet Unans	✗	✗	✓	✗	✓	✗	✗	✓
	Multifacet Amb	✗	✗	✗	✗	✗	✗	✗	✓
	Multifacet Unans	✗	✗	✗	✗	✗	✗	✗	✓

Table 1: Comparison with related works on ambiguity and unanswerable questions in interactive NL2SQL systems.

Conversational **L**anguage **A**mbiguity and unanswe**R**ability in **I**nterac**T**ive NL2SQL s**Y**stems (CLARITY), a novel unified benchmark for evaluating ambiguous and unanswerable NL2SQL queries under realistic interactions. CLARITY automatically synthesizes NL2SQL instances with multiple interacting ambiguity and unanswerability sources, supports both single- and multi-turn interactions, and models diverse user clarification behaviors, including partially helpful and unhelpful responses that commonly arise in production systems. Crucially, CLARITY augments each instance with fine-grained, schema-grounded metadata, including span-level pivot terms and candidate schema target groups, enabling more precise evaluation of NL2SQL systems and pinpointing areas most in need of improvement, a capability not supported by existing benchmarks. This is essential for building robust production NL2SQL systems, where users depend on real-time analysis, identifying the precise causes of failures is therefore essential.

CLARITY generates data without human annotation, enabling low-effort in-house creation of ambiguous and unanswerable variants from enterprise datasets. Instantiating CLARITY on Spider (Yu et al., 2018) and BIRD (Li et al., 2023) shows that state-of-the-art LLMs degrade under multi-facet ambiguity and that high ambiguity-detection accuracy does not guarantee correct schema localization or resolution.

We make the following contributions:

- **CLARITY Benchmark:** We introduce a diagnostic benchmark that systematically captures *multi-facet* ambiguity and unanswerability in both single-turn and multi-turn NL2SQL (Section 2.2). Although our approach leverages LLMs, its main contribution is a scalable, schema-aware framework for controlled generation of ambiguous and unanswerable NL2SQL data.
- **Schema-Grounded, Fine-Grained Annotations.** Instances are augmented with span-level pivot terms and corresponding schema target groups, enabling schema-aware evaluation beyond coarse instance-level labels.
- **Empirical Diagnosis of LLM Limitations.** Extensive experiments show that state-of-the-art LLMs frequently detect ambiguity without correctly localizing the underlying schema elements, with performance degrading sharply under multi-facet ambiguity, failure modes not exposed by existing benchmarks.

2 Benchmark Construction Methodology

2.1 Problem Statement

Let $\mathcal{D} = \{(u_i, q_i, \mathcal{S}_i)\}_{i=1}^{N}$ denote an NL2SQL dataset, where u_i is a natural language query, q_i is its corresponding SQL query, and $\mathcal{S}_i$ is the associated database schema.

Our goal is to construct a synthetic dataset $\mathcal{D}^{A/U}$ consisting of single-turn and multi-turn NL2SQL instances in which the initial user query may contain one or more sources of *ambiguity* or *unanswerability* (A/U) with respect to the schema. Each instance is associated with:
(i) a set of A/U modes

$$M \subseteq \{\texttt{col\_amb}, \texttt{val\_amb}, \texttt{col\_unans}, \texttt{val\_unans}\},$$

(ii) a conversation type

$$R \in \{\texttt{helpful}, \texttt{partial}, \texttt{unhelpful}\},$$

(iii) a set of *pivot terms* $\{p_j\}$, and (iv) a corresponding set of schema-grounded column or value groups $\{G_j\}$.

The pivot terms and target groups provide fine-grained supervision for ambiguity detection and localization.

Given an instance $(u, q, \mathcal{S}, M, R, \{p_j\}, \{G_j\})$, the framework generates a multi-turn interaction

$$C = \{(r_1, t_1), \ldots, (r_T, t_T)\},$$

where r_t and t_t denote user and agent utterances at turn t, respectively, and $r_1 = u$ is the initial (potentially ambiguous or unanswerable) query. For single-turn settings, C reduces to $\{(r_1, t_1)\}$, and r_1 alone constitutes the generated A/U query.

Algorithm 1 The CLARITY Framework

Require: schema S, NL query u, SQL query q, mode m, response type R, number of cases K
Ensure: conversation C, validation flag $v \in \{\text{TRUE}, \text{FALSE}\}$

$t \leftarrow \text{SQLPARSER}(q, m, K)$ // $t[1..K]$: targets
for $i = 1$ to K **do**
  $G[i] \leftarrow \text{ADAPTIVESCHEMARETRIEVER}(t[i], S, m)$
  $f_p \leftarrow \emptyset$
  repeat
    $p[i] \leftarrow \text{PIVOTGENERATOR}(t[i], G[i], m, f_p)$
    $(f_p, ok) \leftarrow \text{PIVOTEVALUATOR}(p[i], G[i], m)$
  until ok
end for
$f_q \leftarrow \emptyset$
repeat
  $r \leftarrow \text{QUERYGENERATOR}(u, q, p, t, f_q)$ // rewritten query
  $(f_q, ok) \leftarrow \text{QUERYEVALUATOR}(r, S, m, q, p, t)$
until ok
$C \leftarrow \text{CONVERSATIONGENERATOR}(r, R)$
$v \leftarrow \text{AUTOMATEDDATASCREENING}(C, p, t, S, m)$
return C, v

Concept	Definition
Ambiguity	A term can be linked to two or more schema entities (column names or values). Subcategories include: *Lexical* (shared tokens or surface forms) and *Semantic* (shared or closely related meanings).
Unanswerable	A term cannot be linked to any schema entity (column names or their values).
Column Use Case	The A/U case is associated with a schema column.
Value Use Case	The A/U case is associated with a column value.
Facet-based Use Case	*Uni-facet:* The query contains a single A/U case. *Multi-facet:* The query contains two or more A/U cases.
Single-turn Instance	An ambiguous NL query that maps to multiple valid SQL queries, or to none if unanswerable.
Multi-turn Instance	A user–agent dialogue in which turns aim to resolve ambiguity or unanswerability in the NL query.

Table 2: Definitions of ambiguity and unanswerable (A/U) use cases. Examples are provided in Table 8.

2.2 Framework Overview

CLARITY is a modular, end-to-end framework, illustrated in Figure 1 and Algorithm 1. We outline the main procedure here, with detailed descriptions of each module provided in Appendix B. For CLARITY, we omit instance indices and describe the procedure for a single input (u, q).

SQL Parser Similar to (Dong et al., 2025), we developed an SQL Parser which analyzes the input query q to extract (i) the set of referenced columns $\mathcal{C}$ and (ii) the set of column–value pairs $(\mathcal{C}, \mathcal{V})$ induced by the predicate structure. It then samples a set of K A/U targets

$$\mathcal{T} = \{t_j\}_{j=1}^{K},$$

where each target t_j corresponds either to a column (m_j, c_j) or to a column–value pair (m_j, c_j, v_j) with mode $m_j \in M$. The value of K controls whether the uni- or multi-facet A/U cases. Sampling is performed without replacement to avoid degenerate overlaps.

Adaptive Schema Retriever To handle complex datasets (Kannan et al., 2025), we introduce an adaptive schema retriever. For each target $t_j \in \mathcal{T}$, it constructs a target space $\mathcal{X}_j$ of relevant schema entities—columns for column-based modes and column values for value-based modes.

For ambiguity modes, it retrieves a candidate group $G_j \subset \mathcal{X}_j$ of lexically or semantically similar elements using LLM-based, dense, or hybrid retrieval, with the strategy and group size adapted to the schema scale and mode. Lexical similarity is computed via token overlap, while semantic similarity is measured using cosine similarity between sentence embeddings; hybrid retrieval combines both signals.

For unanswerable modes, no candidate group is formed. Instead, the full target space serves as a negative reference, ensuring that the generated pivot term does not match any valid schema element.

To distinguish lexical from semantic ambiguity, we apply a two-stage filtering procedure: lexical candidates are first identified via token-level overlap, and semantic candidates are then retrieved using embedding similarity under a lexical-overlap constraint. This separation yields ambiguity cases driven primarily by surface-form versus conceptual similarity.

Pivot Term Generator The Pivot Term Generator synthesizes a set of pivot terms $\{p_j\}_{j=1}^{K}$, one per A/U target using LLM (Nadăș et al., 2025). For each target t_j, it conditions on the retrieval context $(\mathcal{X}_j, G_j)$ and a global exclusion set of previously generated pivot terms. If m_j is an ambiguity mode, the LLM is instructed to generate a term p_j that is compatible with multiple elements in G_j. If m_j is an unanswerable mode, it generates a term that is dissimilar to all elements in $\mathcal{X}_j$. Prompts explicitly prohibit generating terms that already exist in the schema, or previously generated pivot terms, preventing leakage and collisions. $\{p_j\}$ and $\{G_j\}$ serve as ***metadata*** for A/U detection and resolution.

Pivot Term Evaluator Each pivot term is independently validated by multiple LLM judges (Verga et al., 2024) against a set of boolean constraints: (i) conformity to the intended A/U mode, (ii) compatibility with the corresponding target group G_j (for ambiguity modes), and (iii) absence from the schema $\mathcal{S}$. Any violation must be accompanied by a short justification, which is fed back to the Pivot Term Generator for correction. A pivot term proceeds only if all judges unanimously pass all checks.

Context-Aware NL Query Generator This module integrates pivot terms into u to produce an A/U query r_1, enforcing: (i) correct semantic placement of each pivot term based on q; (ii) semantic consistency with q under the intended resolution; and (iii) preservation of the syntactic structure, intent, and writing style of u. This controlled transformation avoids fully synthetic generation, reducing unrealistic phrasing and mitigating potential LLM biases. A context-suppression constraint prevents unintended disambiguation, and failed generations are iteratively refined using evaluator feedback.

NL Query Evaluator The generated query r_1 is evaluated by multiple LLM judges to ensure that: (i) all pivot terms appear exactly once, (ii) the query satisfies the formal A/U definition under $\mathcal{S}$, and (iii) resolving the pivot terms according to $\mathcal{T}$ yields a query consistent with q. Only queries that unanimously pass all criteria are retained.

Conversation Generator This module extends r_1 into a multi-turn interaction. It first produces an agent clarification utterance targeting p_j, which is then diversified by an LLM. Conditioned on the sampled R, the LLM generates alternating user and agent turns, producing a conversation

$$C = \{(r_t, t_t)\}_{t=1}^{T}.$$

For ambiguity cases, user responses are integrated into q to produce a final SQL query; for unanswerable cases, the agent explicitly states that the query cannot be executed.

Automated Data Screening Inspired by (Talaei et al., 2024), we developed an Automated Data Screening module to evaluate each conversation against test cases designed to comply with the A/U specification under $\mathcal{S}$. Using a mixture of LLM judges, each conversation is validated against these test cases, and instances failing majority voting are discarded.

Dataset	Use Case	Uni-Facet	Multi-Facet	Multi-Turn	Acc Human Ann
Spider	Lexical Amb	490	39	2,645	100
	Semantic Amb	612	57	3,345	93.7
	Value Amb	297	9	1,233	93.7
	Unans Column	1,859	1,360	3,219	100
	Unans Value	1,456	200	1,656	100
Bird	Lexical Amb	145	13	790	100
	Semantic Amb	260	32	1,460	100
	Value Amb	421	50	2,355	81.2
	Unans Column	1,071	925	1,996	100
	Unans Value	477	93	570	100

Table 3: Data Statistics. "*Acc Human Ann*" represents the accuracy of human validation on the sampled dataset.

3 Data Statistics

We construct A/U datasets on the development and test splits of Spider (Yu et al., 2018) and BIRD (Li et al., 2023), two large-scale, human-curated NL2SQL benchmarks that pair natural language queries with executable SQL across diverse schemas and domains. Using the CLARITY framework, we generate data without additional human annotation or schema augmentation. GPT-5 is used for generation, while GPT-4o, LLaMA-3.3, and Grok-3 Mini Fast serve as evaluators. We target multi-facet settings by introducing up to two A/U instances per query.

In total, we generate 6,392 (Spider) and 3,487 (BIRD) single-turn instances, and 12,098 (Spider) and 7,171 (BIRD) multi-turn instances across four conversation types. Dataset statistics are summarized in Section 3. The resulting distribution reflects both the structural properties of the source datasets and the pipeline's strict validation criteria. Multi-facet samples are fewer due to the limited prevalence of multi-column SQL queries and stringent pivot- and query-level validation (see Table 10 for examples). Despite their lower frequency, multi-facet instances exhibit higher semantic validity and diagnostic value.

To prevent unbounded regeneration, the evaluator phase is capped at five iterations. Example outputs are provided in Table 12.

To assess whether the generated dataset preserves key linguistic characteristics, we compare it against the original Spider and BIRD datasets. For BIRD, generated queries are longer on average (17.47 vs. 14.93 tokens) and exhibit broader lexical coverage (3,720 vs. 2,301 unique types). Similarly, for Spider, generated queries are slightly longer (13.69 vs. 12.37 tokens) and show increased lexical diversity (4,664 vs. 893 types).

Across both datasets, the distribution of query intents is largely preserved, with retrieval and counting intents accounting for approximately 92% of queries. At the same time, the generated data introduces stronger signals of constraint and aggregation; for instance, the frequency of comparative markers in BIRD increases from 7.2% to 19.9%, reflecting the intended increase in ambiguity and reasoning complexity.

Overall, these results indicate that CLARITY preserves realistic query characteristics while systematically enriching linguistic cues associated with ambiguity and unanswerability.

4 Evaluation Tasks and Metrics

Task. We define evaluation tasks for both single- and multi-turn instances. For value-level A/U, NL2SQL evaluation would require exposing full database contents to the model, which is impractical; context-offloading approaches that could mitigate this are beyond the scope of this work. We therefore exclude value-based A/U categories from this evaluation. Evaluation prompts are provided in Appendix C.

Metrics. For Spider, we report Execution Accuracy (EA), Strict Exact Match (SEM), and Lenient Exact Match (LEM) (Li et al., 2024) to evaluate structural correctness. For BIRD, which contains more complex SQL constructs, we report only EA. For unanswerable cases, the only valid output is `Null`.

4.1 Human Validation

Although the CLARITY benchmark is generated fully autonomously, we conduct a human validation study to verify adherence to the formal A/U definitions in Section 2.2. Three expert annotators with substantial experience in databases and NL2SQL independently evaluated a stratified sample, making binary judgments on whether each instance satisfies its assigned A/U definition, guided by examples distinguishing different A/U types. We sample 8 instances per A/U category from both Spider and BIRD, yielding a balanced set of 448 instances covering all use cases. Validation accuracy is computed via majority vote across annotators.

4.2 SQL Prediction Under Ambiguity

Ambiguous NL queries may admit multiple valid SQL interpretations, whereas unanswerable queries admit none. Accordingly, we design distinct evaluation protocols for single-turn and multi-turn settings.

Single-Turn Prediction. Models are prompted to enumerate all valid SQL interpretations of a given ambiguous query. Performance is evaluated using: (i) Strict Exact Match (SEM), which requires the predicted set of SQL queries to exactly match the ground-truth set; and (ii) Lenient Exact Match (LEM), which requires at least one correct SQL query to be generated.

SEM is particularly informative in multi-faceted settings, as it measures whether the model recovers the full set of valid interpretations.

Multi-Turn Prediction. Models are provided with the full conversation (excluding the final SQL) and must produce a single resolved SQL query. Since ambiguity is resolved through interaction, we evaluate using Execution Accuracy (EA), which measures whether executing the predicted SQL yields the same result as the ground-truth query.

This setup evaluates two complementary capabilities: (1) enumerating alternative interpretations in single-turn scenarios, and (2) resolving ambiguity through dialogue.

Following prior work (Li et al., 2024), we report SEM and LEM on Spider to assess structural correctness, and EA on the more complex BIRD dataset. For unanswerable queries, the only valid output is `null`. We report macro-averaged accuracy for cross-category comparison.

4.3 Ambiguity Detection

We evaluate ambiguity detection on single-turn instances using AmbiSQL (Ding et al., 2025). Predicted ambiguities are compared against the ground-truth metadata from CLARITY. We report *Detection Accuracy (DA)*, which measures whether ambiguity is correctly identified, and *Match Accuracy (MA)*, which assesses whether the predicted ambiguity aligns with the underlying schema elements. We further use our generated instances as few-shot exemplars to evaluate their impact on AmbiSQL's performance.

5 Results and Discussion

Section 3 summarizes the human annotation results. A 10–20% accuracy drop is observed only for `multi val_amb` cases; all other A/U categories show consistently high agreement. One additional

Setting	Uni		Multi	
	LEM	SEM	LEM	SEM
Zero-shot	19.2	15.0	15.2	5.2
No meta (uni)	56.9	42.9	55.3	13.4
No meta (uni+multi)	**62.7**	50.3	61.4	23.4
Meta (uni)	60.5	53.9	67.0	**60.2**
Meta (uni+multi)	61.3	**55.8**	**70.1**	60.1

Table 4: Few-shot single-turn SQL prediction for column ambiguity using GPT-5 (%).

error stems from a syntax issue in an underlying SQL query for `sem_amb`, reflecting dataset-specific artifacts of the data-driven generation process. During generation, multiple evaluators are used to filter erroneous instances (see Table 10), ensuring adherence to the A/U definitions and contributing to the overall high annotation accuracy.

We benchmark five strong LLMs: one open-source model (LLaMA-3.3) and four closed-source models (GPT-4o, GPT-4.1, GPT-5, and Grok-3 Mini Fast). For all models, we set `temperature = 0.0` and `max_tokens = 2048`; for GPT-5, `reasoning_effort` and `verbosity` are additionally set to `medium`. Model parameters are not tuned, and no prompt optimization is performed. Using zero-shot prompting as a baseline, we evaluate performance on the full CLARITY benchmark. Prompts from (Saparina and Lapata, 2024) and (Ding et al., 2025) are used without task-specific adaptation.

Under zero-shot prompting, LLMs struggle to enumerate multiple SQL interpretations for column-level ambiguity (Table 4, Table 15). Low SEM indicates missed valid interpretations, while higher LEM reflects partial coverage (i.e., multiple but incomplete outputs).

We evaluate two few-shot strategies by randomly sampling two exemplars per CLARITY ambiguity case (excluded from the test set). Each exemplar consists of an ambiguous NL query paired with multiple SQL queries. We consider: (i) *without metadata*, and (ii) *with metadata*, where metadata includes pivot-term and target-group annotations. Metadata is used only in-context to aid ambiguity detection and is not available at test time.

For both settings, we compare exemplars containing only uni-facet cases with those including both uni- and multi-facet cases. GPT-5 results are presented in Table 4, with full A/U case-wise results reported in Appendix D, along with complementary statistical analyses.

NL2SQL exemplars substantially improve both LEM and SEM over zero-shot prompting. Incorporating both uni- and multi-facet exemplars further outperforms uni-only settings, with particularly strong gains on multi-facet cases. Metadata-enhanced exemplars provide additional improvements, especially in SEM for multi-facet scenarios. As SEM reflects recovery of the complete set of valid SQL interpretations for an A/U query, it serves as a more diagnostic metric than LEM. Overall, the inclusion of structured metadata significantly enhances performance on multi-facet A/U cases. Further gains may be achievable through optimized exemplar selection.

Table 5 and Table 20 report zero-shot performance on the multi-turn SQL prediction task. Unanswerable cases are generally easier than ambiguous ones, as pivot terms do not correspond to any schema element and correctly yield a `null` output, resulting in higher EA. In contrast, multi-facet ambiguities are consistently more challenging than uni-facet cases, as reflected by lower EA. Performance differences across LLMs are relatively modest, with no single model dominating across all A/U categories, suggesting that successful ambiguity resolution depends more on interaction structure than on model-specific capabilities. EA also varies by conversation type: *verbose* interactions achieve higher scores by providing stronger contextual clarification signals.

Tables 6 and 7[1] summarize AmbiSQL's detection performance on the CLARITY benchmark. While detection accuracy (DA) is near-perfect, match accuracy (MA) remains substantially lower, indicating weaknesses in schema-level localization. Errors primarily arise from misidentifying ambiguous terms and conflating lexical with semantic ambiguity. Because MA jointly evaluates detection and alignment with ground-truth metadata, models often correctly flag ambiguity but produce misaligned clarification outputs. Incorporating CLARITY exemplars improves MA for uni-facet cases but yields limited gains for multi-facet settings, underscoring the importance of fine-grained, metadata-aware evaluation beyond binary detection.

We conduct paired t-tests to assess gains from CLARITY exemplars. NL2SQL models augmented with CLARITY-based few-shot examples achieve significant improvements over both zero-shot and standard few-shot prompting without metadata

[1]Additional results are provided in Appendix D.

Model	A/U Use Cases								Conversation Types			
	U-Lex Amb	M-Lex Amb	U-Sem Amb	M-Sem Amb	U-Col Unans	M-Col Unans	U-Val Amb	M-Val Amb	Concise	Verbose	Partial	Not
GPT-4o	74.2	73.2	74.2	67.2	99.0	100.0	68.3	60.6	73.1	80.5	74.6	62.6
GPT-4.1	75.2	73.6	81.0	73.2	95.0	99.0	75.2	66.7	68.3	82.2	78.1	71.7
GPT-5	73.0	71.6	77.6	69.4	83.0	93.0	78.3	70.8	62.9	79.3	77.2	70.0
Grok-3 Mini Fast	78.2	75.8	82.4	77.8	86.0	94.0	76.8	65.9	64.8	81.7	79.3	75.8
LLaMA-3.3	76.8	76.8	80.6	69.8	44.0	72.0	72.6	64.9	42.3	81.3	76.5	72.8

Table 5: Multi-turn SQL prediction performance on Spider, measured by execution accuracy (EA, %). U = uni-facet; M = multi-facet; Lex = lexical; Sem = semantic; Col = column; Val = value; Unans = unanswerable.

Dataset	Method	U Col Amb		M Col Amb	
		DA	MA	DA	MA
Spider	AmbiSQL	99.6	57.8	**100.0**	**20.4**
	AmbiSQL-CT	**99.8**	**62.2**	**100.0**	19.6
BIRD	AmbiSQL	98.9	50.1	**100.0**	5.4
	AmbiSQL-CT	**99.3**	**57.7**	**100.0**	**7.0**

Table 6: Column ambiguity detection (%).

Dataset	Method	U Val Amb		M Val Amb	
		DA	MA	DA	MA
Spider	AmbiSQL	98.0	**22.0**	96.8	35.1
	AmbiSQL-CT	**100.0**	21.8	**100.0**	**40.8**
BIRD	AmbiSQL	**98.8**	37.7	98.0	26.0
	AmbiSQL-CT	**98.8**	**41.3**	**98.0**	**32.0**

Table 7: Value ambiguity detection (%).

($p < 0.005$). Similarly, AmbiSQL shows significant gains in column ambiguity detection with CLARITY exemplars ($p < 0.005$).

6 Related Work

Single-Turn Ambiguity and Unanswerability. Most benchmarks use rule-based or semi-automated pipelines combining LLMs and human annotation. Prior work generates ambiguity via schema augmentation or templates (Saparina and Lapata, 2024; Bhaskar et al., 2023; Wang et al., 2023), or synthesizes ambiguous NL2SQL examples from schema-level patterns (Papicchio et al., 2025), while Luo et al. (2025) study ambiguity in text-to-visualization. However, ambiguity is typically limited to a single failure mode per query with coarse instance-level labels.

Conversational NL2SQL. Dong et al. (2025) define a fixed clarification workflow via schema augmentation. Guo et al. (2024) generate multi-turn QA sequences by prompting over existing datasets, without explicitly modeling ambiguity resolution or unanswerable cases. Huo et al. (2025) transform single-turn NL2SQL instances into dialogues via expert annotation. These datasets assume cooperative users who provide clean, sufficient clarifications and ignore variation in response quality.

NL2SQL Dataset Generation Frameworks. The state-of-the-art frameworks (Duan et al., 2025; Li et al., 2025; Guo et al., 2025; Caferoğlu et al., 2025; Pourreza et al., 2024) typically assume a single well-defined SQL target per query and do not explicitly address multiple ambiguities, unanswerabilities, or interactive resolutions.

7 Conclusion and Future Work

CLARITY provides a controlled framework for evaluating NL2SQL systems under multi-faceted ambiguity, unanswerability, and diverse conversational settings. Rather than replacing existing benchmarks, it complements them by exposing schema-localization and ambiguity-resolution failures not captured by instance-level evaluation. Results show substantial variation across A/U types, indicating that NL2SQL systems should be explicitly optimized for diverse failure modes—an important requirement for enterprise deployment.

Future work includes modeling more realistic user behaviors, such as topic shifts and user uncertainty; handling multiple, heterogeneous forms of ambiguity; and supporting more complex multi-turn interactions beyond the settings considered in MMSQL (Guo et al., 2024) and BIRD-INTERACT (Huo et al., 2025). Extending CLARITY to queries that require external or implicit knowledge, as well as integrating it into downstream systems through specialized fine-tuning and retrieval-augmented in-context learning, are promising directions for improving robustness under ambiguity.

References

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improv-

ing text-to-sql generation under ambiguity. *arXiv preprint arXiv:2310.13659*.

Hasan Alp Caferoğlu, Mehmet Serhat Çelik, and Özgür Ulusoy. 2025. Sing-sql: A synthetic data generation framework for in-domain text-to-sql translation. *arXiv preprint arXiv:2509.25672*.

Kaiwen Chen, Yueting Chen, Nick Koudas, and Xiaohui Yu. 2025. Reliable text-to-sql with adaptive abstention. *Proceedings of the ACM on Management of Data*, 3(1):1–30.

Zhongjun Ding, Yin Lin, and Tianjing Zeng. 2025. Ambisql: Interactive ambiguity detection and resolution for text-to-sql. *arXiv preprint arXiv:2508.15276*.

Mingwen Dong, Nischal Ashok Kumar, Yiqun Hu, Anuj Chauhan, Chung-Wei Hang, Shuaichen Chang, Lin Pan, Wuwei Lan, Henghui Zhu, Jiarong Jiang, and 1 others. 2025. Practiq: A practical conversational text-to-sql dataset with ambiguous and unanswerable queries. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 255–273.

Shaoming Duan, Youxuan Wu, Chuanyi Liu, Yuhao Zhang, Zirui Wang, Peiyi Han, Shengyuan Yu, Liang Yan, and Yingwei Liang. 2025. Dsqg-syn: Synthesizing high-quality data for text-to-sql parsing by domain specific question generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2971–2989.

Yu Guo, Dong Jin, Shenghao Ye, Shuangwu Chen, Jianyang Jianyang, and Xiaobin Tan. 2025. Sqlforge: Synthesizing reliable and diverse data to enhance text-to-sql reasoning in llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8441–8452.

Ziming Guo, Chao Ma, Yinggang Sun, Tiancheng Zhao, Guangyao Wang, and Hai Huang. 2024. Evaluating and enhancing llms for multi-turn text-to-sql with multiple question types. *arXiv preprint arXiv:2412.17867*.

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

Yiming Huang, Jiyu Guo, Wenxin Mao, Cuiyun Gao, Peiyi Han, Chuanyi Liu, and Qing Ling. 2025. Exploring the landscape of text-to-sql with large language models: Progresses, challenges and opportunities. *arXiv preprint arXiv:2505.23838*.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, and 1 others. 2025. Bird-interact: Re-imagining text-to-sql evaluation for large language models via lens of dynamic interactions. *arXiv preprint arXiv:2510.05318*.

Shivasankari Kannan, Yeounoh Chung, Amita Gondi, Tristan Swadell, and Fatma Ozcan. 2025. High-fidelity and complex test data generation for real-world sql code generation services. *arXiv preprint arXiv:2504.17203*.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The dawn of natural language to sql: Are we fully ready? *arXiv preprint arXiv:2406.01265*.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, and 1 others. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240*.

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.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025. 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*.

Tianqi Luo, Chuhan Huang, Leixian Shen, Boyan Li, Shuyu Shen, Wei Zeng, Nan Tang, and Yuyu Luo. 2025. nvbench 2.0: Resolving ambiguity in text-to-visualization through stepwise reasoning. *arXiv preprint arXiv:2503.12880*.

Mihai Nadăș, Laura Dioșan, and Andreea Tomescu. 2025. Synthetic data generation using large language models: Advances in text and code. *IEEE Access*.

Simone Papicchio, Luca Cagliero, and Paolo Papotti. 2025. Squab: Evaluating llm robustness to ambiguous and unanswerable questions in semantic parsing. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17937–17957.

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

Mohammadreza Pourreza, Ruoxi Sun, Hailong Li, Lesly Miculicich, Tomas Pfister, and Sercan O Arik. 2024. Sql-gen: Bridging the dialect gap for text-to-sql via synthetic data and model merging. *arXiv preprint arXiv:2408.12733*.

Irina Saparina and Mirella Lapata. 2024. Ambrosia: A benchmark for parsing ambiguous questions into database queries. *Advances in Neural Information Processing Systems*, 37:90600–90628.

Irina Saparina and Mirella Lapata. 2025. Disambiguate first parse later: Generating interpretations for ambiguity resolution in semantic parsing. *arXiv preprint arXiv:2502.18448*.

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

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

Kapil Vaidya, Abishek Sankararaman, Jialin Ding, Chuan Lei, Xiao Qin, Balakrishnan Narayanaswamy, and Tim Kraska. 2025. Odin: A nl2sql recommender to handle schema ambiguity. *arXiv preprint arXiv:2505.19302*.

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing judges with juries: Evaluating llm generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796*.

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.

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.

Fuheng Zhao, Shaleen Deep, Fotis Psallidas, Avrilia Floratou, Divyakant Agrawal, and Amr El Abbadi. 2024. Sphinteract: Resolving ambiguities in nl2sql through user interaction. *Proceedings of the VLDB Endowment*, 18(4):1145–1158.

A The CLARITY Framework - Appendix

This appendix presents a taxonomy of A/U in CLARITY. We define core A/U concepts (Table 8), characterize user clarification behaviors in multi-turn interactions (Table 9), provide representative conversational examples (Table 12), and analyze model failure patterns across A/U categories (Table 10).

Concept	Definition	Examples
Ambiguity	A term can be linked to **AT LEAST TWO** schema entities (can be column names or column values). **Subcategory:** • *Lexical Ambiguity:* Two terms or columns are lexically similar if they share a subset of tokens and writing style • *Semantic Ambiguity:* Two terms or columns are semantically similar if they share same meaning or similar underlying concept	**Lexical Ambiguity:** The term `date` is lexically ambiguous, as it may refer to either `Production_Date` or `Sale_Date` column in the schema. **Semantic Ambiguity:** The term `performance` is semantically ambiguous, as it may refer to either `sales achieved` or `projects completed` column in the schema.
Unanswerable	A term that **CANNOT** be linked to any schema entity (column names or column values)	Refer to examples mention under **Column** and **Value Use Cases**
Column Use Case	The A/U use case is linked with the column of the schema	**Unanswerable Column:** The marked column does not exist in the schema "how many `tables` did we have?"
Value Use Case	The A/U use case is linked with the value of the schema column	**Unanswerable Value:** The marked value does not exist in the database "show records where location is `Pru`"
Facet-based Use Case	***Uni-facet:*** The natural language query contains single A/U column or value ***Multi-facet:*** The query contains two or more A/U columns or values	The marked term represents ambiguity case ***Uni-facet:*** "show me `name` grouped by city" ***Multi-facet:*** "List the `date` claim where the `amount` is less than 200"
Single Turn Instance	One ambiguous natural language query associated with one or more SQL queries. None if unanswerable and no SQL query exists for unanswerable cases	**User Query:** "List the date and amount" **SQL Queries:** *"SELECT paymentDate, amount FROM Order;"* *"SELECT dispatchDate, amount FROM Order;"*
Multi Turn or Conversational Instance	A series of user and agent natural language queries associated with a SQL query, where the series of conversations represents the resolution of ambiguous or unanswerable use case.	**User Query:** "List the date where amount is less than 100" **Agent Response:** "When you say `date`, do you mean `paymentDate` or `dispatchDate`?" **User Query:** "Use paymentDate" **Agent Response:** *"SELECT paymentDate FROM Order WHERE amount <200;"*

Table 8: Ambiguity and Unanswerable (A/U) use case definitions.

Type	Definition	Examples
Helpful	User provides a response that helps in resolving the ambiguity found in the NL query. **Subcategories:** *Concise:* User response provides a direct answer. *Verbose:* User response explains additional details.	**User:** show me name grouped by city **Agent:** When you say "name", do you mean "Customer Name" or "Product Name"? **Concise: User:** Product Name **Verbose: User:** I mean product names grouped by cities.
Partially Helpful	The user response only partially matches a schema entity prompted by the agent	**Partially Helpful:** **User:** prod name (or just "prod")
Not Helpful	The user response does not provide direct resolution instructions	**Not Helpful:** **User:** i don't know

Table 9: Different types of multi-turn conversation. The *Not Helpful* case covers user responses that are irrelevant or out of scope, such as gibberish input or references to non-existent or unrelated schema columns.

Model	Unqualified for A/U		Non-existent Target Group		Term Evaluator Failure		Query Evaluator Failure		Data Screening Failure	
	Spider	BIRD	Spider	BIRD	Spider	BIRD	Spider	BIRD	Spider	BIRD
Uni Lex Amb	9.53	2.41	33.84	17.54	22.94	29.53	22.13	37.35	3.77	3.52
Multi Lex Amb	40.1	6.32	32.01	49.48	15.73	32.46	8.7	10.37	0.03	0.26
Uni Sem Amb	9.53	2.41	20.38	13.95	36.47	35.79	24.48	28.68	0.39	0.52
Multi Sem Amb	40.1	6.32	26.09	28.55	25.61	52.61	7.37	10.04	0.01	0.07
Uni Val Amb	73.06	71	0	0	12.2	11.56	5.4	4.17	10.5	11.23
Multi Val Amb	99.8	96.31	0	0	0	0.07	0.04	4.46	0.03	0.45
Uni Col Unans	9.53	2.41	0	0	0.06	2.22	63.7	24.64	0.16	0.72
Multi Col Unans	40.1	6.32	0	0	5.89	17.21	34.59	15.97	0	0
Uni Val Unans	70.46	23.47	0	0	0.06	0.02	5.11	2.42	0	0
Multi Val Unans	94.69	93.83	0	0.07	5.89	0.59	0.94	0.46	0	0

Table 10: Unsuccessful Cases (%) at Various Steps of Framework . Total instances in original dataset Spider=7,000 and BIRD=1,534. Uni: Unifacet; Multi: Multifacet; Lex: Lexical Column; Sem: Semantic Column; Val: Value; Col: Column.

Example	Rejection Category	Details
1	Rejected due to implicit disambiguation	**Ambiguous Query:** What is the average number of horses for farms with total horses greater than 5000? **Pivot Term:** horses **Target Group:** [Working_Horses, Total_Horses] **Reason for rejection:** The query context implicitly resolves the ambiguity (average → Working_Horses, total → Total_Horses), making the query effectively unambiguous
2	Rejected due to weak ambiguity signal	**Ambiguous Query:** What is the average number of horses for farms with total horses greater than 5000? **Pivot Term:** "horses" **Target Group:** [Working_Horses, Total_Horses] **Reason for rejection:** The query context implicitly resolves the ambiguity (average → Working_Horses, total → Total_Horses), making the query effectively unambiguous.

Table 11: Rejected instances identified by the NL Query Evaluator in the CLARITY framework.

A/U Use Case	Example
Uni-facet Lexical Column Ambiguity Verbose Helpful	"USER" : "Find the list of attributes that appear more than 3 times." "AGENT" : "Which attribute do you mean: attribute_data_type, attribute_name, or attribute_id?" "USER" : "Please use attribute_data_type, that's what I meant." "AGENT" : "SELECT attribute_data_type FROM Attribute_Definitions GROUP BY attribute_data_type HAVING count(*) > 3"
Uni-facet Semantic Column Ambiguity Concise Helpful	"USER" : "Which region has the most wines with a score greater than 90?" "AGENT" : "When you say "region," should I use County, Appelation, or state?" "USER" : "County." "AGENT" : "SELECT T1.County FROM APPELLATIONS AS T1 JOIN WINE AS T2 ON T1.Appelation = T2.Appelation WHERE T2.Score > 90 GROUP BY T1.County ORDER BY count(*) DESC LIMIT 1"
Multi-facet Lexical Column Ambiguity Partially Helpful	"USER" : "List the date claim for claims where the amount is less than or equal to the average amount." "AGENT" : "When you say "date claim," should I use "date_claim_settled" or "Date_Claim_Made"? Also, for "amount," do you mean "Amount_Settled" or "amount_claimed"?" "USER" : "made and settled" "AGENT" : "SELECT Date_Claim_Made FROM Claims WHERE Amount_Settled <= (SELECT avg(Amount_Settled) FROM Claims)"
Multi-facet Lexical Column Ambiguity Not Helpful	"USER" : "What is the total lesson time where appellation is Janessa and identity component is Sawayn?" "AGENT" : "When you say "appellation," are you referring to "first_name", "middle_name", or "last_name"? Also, for "identity component," should that be "first_name", "middle_name", or "last_name"?," "USER" : "I don't know." "AGENT" : "The query cannot be generated as "appellation" and "identity component" cannot be mapped to valid column names"
Uni-facet Value Ambiguity Concise Helpful	"USER" : "How many addresses are there in state "U.S. states"?" "AGENT" : "When you say "U.S. states," which value in state_province_county do you mean? For example: Vermont, Georgia, Colorado, NorthCarolina, Arkansas, Pennsylvania, Kentucky, Michigan, Montana, NewYork, Missouri, Louisiana, or NewHampshire." "USER" : "state_province_county = Colorado" "AGENT" : "SELECT count(*) FROM addresses WHERE state_province_county = "Colorado""
Uni-facet Column Unanswerable	"USER" : "Which customers have made at least two orders? Return the churn_risk_score and number of orders." "AGENT" : "Sorry, I can't complete this request because the column churn_risk_score isn't found in the database, and there are no close matches."
Uni-facet Value Unanswerable	"USER" : "Count the number of cards where card type code is Prepaid." "AGENT" : "Sorry, I can't complete that request because the value 'Prepaid' does not exist in the 'card_type_code' column in the database."

Table 12: Example conversations generated by CLARITY.

B Pseudo Codes - Appendix

In this section, we present the algorithms of each component in CLARITY following execution order. We start with the end-to-end framework and then elaborate each sub-module it invokes.

Overview. Algorithm 1 summarizes the full CLARITY pipeline. Given a database schema, an original NL–SQL pair, and a target ambiguity or unanswerability mode, the framework iteratively constructs pivot terms, generates ambiguous or unanswerable NL queries, simulates multi-turn conversations, and finally validates the resulting instances via automated screening.

Target Extraction. The first step identifies schema elements to be perturbed. Algorithm 2 parses the SQL query and samples K target columns or column–value pairs depending on the ambiguity or unanswerability mode.

Algorithm 2 SQLParser

Require: SQL query q; integer K;
mode $m \in$ `col_amb`, `val_amb`, `col_unans`, `val_unans`
Ensure: Parsed representation $sql\_structure$; list $target$ of length K
$sql_structure \leftarrow$ ParseSQL(q)
$Columns \leftarrow$ ExtractColumns($sql_structure$)
$ColValPairs \leftarrow$ ExtractColumnValuePairs($sql_structure$)
$target \leftarrow [\,]$
for $k \leftarrow 1$ to K **do**
  if $m \in$ `col_amb`, `col_unans` **then**
    $item \leftarrow$ RandomChoice($Columns$)
    $target[k] \leftarrow item$
    $Columns \leftarrow Columns \setminus item$
  else if $m \in$ `val_amb`, `val_unans` **then**
    $item \leftarrow$ RandomChoice($ColValPairs$)
    $target[k] \leftarrow item$
    $ColValPairs \leftarrow ColValPairs \setminus item$
  else
    error Unknown mode m
  end if
end for
return $(target, sql\_structure)$

Schema Retrieval. Given a target element, Algorithm 3 retrieves a set of competing columns or values that serve as the basis for ambiguity or unanswerability construction. For ambiguous queries, retrieval is selective; for unanswerable queries, the full schema is used as a negative reference.

Pivot Term Generation and Evaluation. Algorithms 4 and 5 jointly construct and validate synthetic pivot terms. The generator proposes candidate pivot terms, while the evaluator checks validity, dissimilarity from schema elements, and ambiguity consistency. A feedback loop ensures only high-quality pivot terms are retained.

NL Query Generation and Evaluation. Algorithm 6 injects validated pivot terms into the original NL query while preserving its intent and surface form. Algorithm 7 then verifies pivot usage, ambiguity or unanswerability compliance, and semantic consistency with the original SQL query.

Conversation Modeling. Given a validated ambiguous or unanswerable NL query, Algorithm 8 simulates a multi-turn interaction between a user and an agent. Different response styles (helpful, partially helpful, or unhelpful) are supported to model realistic clarification behaviors.

Final Validation. Algorithm 9 performs automated quality control using predefined test cases specific to each ambiguity or unanswerability mode. Only instances that pass all screening checks are retained in the final dataset.

Algorithm 3 AdaptiveSchemaRetriever

Require: target t;
mode $m \in$ `col_amb`, `val_amb`, `col_unans`, `val_unans` ;
schema $\mathcal{S}$; *retrieval_type* $\in$ `llm`, `hybrid`, `combined`
Ensure: list G of similar columns or values for A/U query generation
$sample_k \leftarrow$ HEURISTICTOPK($|\mathcal{S}|$)
if $m \in$ `col_amb`, `val_amb` **then**
  if $m =$ `col_amb` **then**
    $target_space \leftarrow \mathcal{S}$ {schema column space}
  else
    $target_space \leftarrow$ ALLVALUESINCOLUMN($t, , sample_k$)
  end if
  if *retrieval_type* $=$ `llm` **then**
    $G \leftarrow$ LLMRETRIEVER($t, , target_space$)
  else if *retrieval_type* $=$ `hybrid` **then**
    $G \leftarrow$ DENSERETRIEVER($t, , target_space$)
  else if *retrieval_type* $=$ `combined` **then**
    $k \leftarrow$ HEURISTICTOPK($|target_space|$)
    $candidates \leftarrow$ HYBRIDRETRIEVER_TOPK($t, , target_space, , k$)
    $candidates_normalized \leftarrow$ NORMALIZESIMILARITYSCORES($candidates$)
    $G \leftarrow$ LLM_FILTER($t, , candidates_normalized, , m$)
  else
    error Unknown retrieval type
  end if
else if $m \in$ `col_unans`, `val_unans` **then**
  $G \leftarrow \mathcal{S}$ {use full schema as negative reference}
else
  error Unknown mode m
end if
return G

Algorithm 4 PivotTermGenerator

Require: G set of retrieved columns or values;
mode $m \in$ `col_amb`, `val_amb`, `col_unans`, `val_unans`; $generation\_type \in$ `semantic`, `lexical`
Ensure: synthetic pivot term p that induces ambiguity or unanswerability
if CONTAINS(m, `ambiguity`) **then**
  $constraint \leftarrow$ EXTRACTDEFINITION($generation\_type$)
else
  $constraint \leftarrow$ EXTRACTDEFINITION(m)
end if
$p \leftarrow$ LLM_GENERATEPIVOTTERM($G, m, generation\_type, constraint$)
return p

Algorithm 5 PivotTermEvaluator

Require: p pivot term ;
G retrieved set ;
S database schema ;
mode $m \in$ `col_amb`, `val_amb`, `col_unans`, `val_unans`;
$generation\_type \in$ `semantic`, `lexical`;
Ensure: `evaluation_outcome`: dictionary of evaluation outcomes and feedbacks; `all_pass`: boolean
`evaluation_outcome` $\leftarrow$ empty dictionary
`evaluation_outcome`[“`pivot_validity`”] $\leftarrow$ `LLM`($p, m,$ `generation_type`) {Check p satisfies the definition(s) in m}
`evaluation_outcome`[“`pivot_dissimilarity_all`”] $\leftarrow$ `LLM`($p, S, m,$ `generation_type`) {Verify p is dissimilar to schema S}
if CONTAINS(m, `ambiguity`) **then**
  `evaluation_outcome`[`pivot_term_for_target_group`] $\leftarrow$ `LLM`($p, G, m,$ `generation_type`) {Check whether p is truly ambiguous and conflicts with multiple schema columns or values in G}
end if
`all_pass` $\leftarrow$ ALLTRUE(VALUES(`evaluation_outcome`))
return `evaluation_outcome`, `all_pass`

Algorithm 6 NLQueryGenerator

Require: original NL query u; SQL query q; set of pivot terms p; set of targets t
Ensure: A/U NL query r_1 containing all pivot terms
$\mathcal{C} \leftarrow$ constraint set defined as:
(i) Use pivot terms in p to refer to targets in t (do not use specific target names).
(ii) Ensure r_1 semantically corresponds to the SQL query q.
(iii) Preserve the sentence structure, intent, tone, and writing style of u.
(iv) Suppress contextual cues that would disambiguate targets in t from pivot terms in p.
$r_1 \leftarrow \text{LLM}\big(\textsc{Prompt}(u, q, p, t, \mathcal{C})\big)$
return r_1

Algorithm 7 NLQueryEvaluator

Require: r_1 generated A/U NL query;
q SQL query;
p pivot terms;
t targets;
S schema;
mode $m \in$ `col_amb`, `val_amb`, `col_unans`, `val_unans`;
generation_type $\in$ `semantic`, `lexical`
Ensure: evaluation_outcome: dictionary mapping criteria to (pass, feedback); all_pass: boolean
evaluation_outcome $\leftarrow$ empty dictionary
evaluation_outcome["pivot_in_utterance"] $\leftarrow \text{LLM}(r_1, p)$ {Verify all pivot terms in p appear in r_1}
evaluation_outcome["validity"] $\leftarrow \text{LLM}(r_1, m,$ *generation_type*$, S)$ {Check A/U compliance for m and *generation_type*; ensure no contextual disambiguation}
evaluation_outcome["consistency"] $\leftarrow \text{LLM}(r_1, p, t, q)$ {Check semantic consistency with q after resolving p via t}
`all_pass` $\leftarrow$ ALLTRUE(VALUES(`evaluation_outcome`))
return (all_pass, evaluation_outcome)

Algorithm 8 ConversationGenerator

Require: r_1 initial NL query;
q original SQL query;
p set of pivot terms;
t set of targets;
$mmode \in$ `col_amb`, `val_amb`, `col_unans`, `val_unans`;
$Rresponse\_type \in$ `concise helpful`, `verbose helpful`, `partial`, `unhelpful`
Ensure: conversation log $C = (r_t, t_t)_{t=1}^{T}$ (list of user/agent response tuples)
$t_1 \leftarrow$ GENERATETEMPLATERESPONSE(r_1, p, t, m) {Template indicating A/U use case}
$C \leftarrow$ GENERATECONVERSATION(t_1, R, q) {Generate multi-turn user/agent responses using LLM}
return C

Algorithm 9 AutomatedDataScreening

Require: C Multi–turn instance; p set of pivot terms; t set of targets; S schema; mode $m \in$ "`col_amb`", "`val_amb`", "`col_unans`", "`value_unans`"
Ensure: validation_flag $\in$ True, False
$TC \leftarrow$ EXTRACTPREDEFINEDTESTCASE(m)
evaluation $\leftarrow$ [] {initialize empty evaluation}
for $i \leftarrow 1$ to $|TC|$ **do**
evaluation[i] $\leftarrow \text{LLM}(TC[i], C, p, t, S)$
end for
return True if all items in evaluation pass, else False

B.1 LLM Prompts

The prompts utilised in CLARITY are presented below.

A/U Term Generator Prompt - Lexical Column Ambiguity Case

You are a SQL expert specializing in natural language (NL)-to-SQL translation and data generation. You are given a column of interest and list of similar columns. Your task is to analyze the list of similar columns and identify a subset of columns that similar to the given column of interest, which might cause ambiguity if referenced without enough detail in a query. Ambiguity arises when multiple columns have similar names or overlapping meanings, making it unclear which one should be used.

Instructions
The following instructions must be strictly followed:
1. **Group Selection**: Identify groups of columns that are lexically similar to the given column of interest
- Identify the columns that have similar names and writing style or structure to the column of interest and group them.
- The column of interest and selected columns must share common tokens, words or segments.
- If there are many column names which are similar to the column of interest, then select the ones that share most tokens, words or segments.
- The group should contain one or more columns from the given list.

2. **Term Generation**: Generate an ambiguous term for the column of interest that can likely be asked by a human database manager and cause ambiguity with the selected columns group using the following rules:
- The ambiguous term should be lexically similar to column of interest in terms of wording, length, and style.
- The ambiguous term should NOT be an exact match with column of interest and any column from the selected column list i.e., the term should be slightly different to all the columns in the list and should not exactly match with any of the columns.
- The ambiguous term should NOT contain token, word, or segment from column(s) of the selected group which does not appear in the column of interest e.g., for "employee name" column of interest, "manager employee" is an invalid as token 'manager' token exists in selected column groups, ["Manager Employee Name", "Manager Employee ID", "Manager Employee Contact"] but not in the column of interest.
- The ambiguous term should NOT contain keyword or token that could be linked with a unique column e.g., for column of interest "Customer Number" and selected columns ["Customer Returning ID", "Customer Name"], "customer identifier" is an invalid ambiguous term as it can be associated with "customer Returning ID" instead of the column of interest. Moreover, it is also not associated with the column of interest.
- The ambiguous term should NOT be a synonym or paraphrased word e.g., if we have two columns "Employee Title" and "Employee Designation" then "worker" term is invalid as it is a synonym of "Employee".
- Do not use vague generic terms. For example, "count" for 'Number of People', 'Number of Cities' or 'Number of Countries' columns is an invalid ambiguous term.
- Ignore the case (uppercase or lowercase), singular plural and minor variations when generating an ambiguous term for the selected group. For example:
-'Customer Count', 'customer count', 'CUSTOMER COUNT', 'customer_count' and 'customer counts' represent the same column.
- 'no.', 'number' or '#' represent the same word or token.
- 'Customer Count' and 'Count Customer' represent the same column.

- If an ambiguous term cannot be generated when multiple lexically similar columns do not exist, then use 'Null' for ambiguous term.
3. **Constraints**:
- The ambiguous term and selected group must be specific to the column of interest, i.e., the ambiguous term is generated for the column of interest which causes ambiguity with the selected columns group.
- For the ambiguous term, DO NOT use tokens, words or segments from columns of the selected group that do not appear in the column of interest.
- The token, words or segments in ambiguous term must appear in both column of interest and the columns of the selected group.
- Take into consideration the history, if provided, when generating the response.
- The history contain errors linked with the term generated in the previous attempt(s).
- Strictly use the history to generate a valid ambiguous term.
- Strictly generate the response considering the column of interest, similar columns list and given constraints. Do not introduce any assumptions or external knowledge beyond the given columns.
- Refrain from providing elaborative reasoning and only provide a short summarised reason, following the provided examples and output template.
4. **Format**: Strictly generate the response using the given output format. DO NOT generate any other content or text.

Examples

Example 1:
Column of Interest: "Monthly Orders"
Given List of Similar Columns: ["Total Orders", "Yearly Orders", "Order Dispatch Date" , "Order Invoice ID"]
History: None
Selected Group: ["Total Orders", "Yearly Orders"]
Ambiguous Term: "orders"
Reason: "The column of interest 'Monthly Orders' share common token 'orders' with all the columns in the selected list and 'orders' itself is not a column name in the given list."

Example 2:
Column of Interest: "Customer_ID"
Given List of Similar Columns: ["Employee_ID", "Employee_Name", "Department_ID", "Department_Name", "Organisation_ID"]
History: None
Selected Group: ["Employee_ID", "Department_ID", "Organisation_ID"]
Ambiguous Term: "ID"
Reason: "The column of interest 'Customer_ID' has 'ID' which also appears in the three selected columns and 'ID' itself is not a column name in the given list."

Example 3:
Column of Interest: "Order Generated Transfer Number"
Given List of Similar Columns: ["Order Generated Bar Number", "Parcel Generated Bar Number", "Order Generated Dispatch Number"]
History: None
Selected Group: ["Order Generated Bar Number", "Order Generated Dispatch Number"]
Ambiguous Term: "order generated number"
Reason: "'Order Generated' and 'number' in the column of interest are common across the three selected columns. So 'order generated number' ambiguous term could be used as this is an exact column in the list"

Example 4:
Column of Interest: "Cost"
Given List of Similar Columns: ["Annual Cost", "Order Date", "Order Cost", "Annual Profit", "Cost per Item"]
History: "'cost' itself is a column name in the given list. So this is not a valid ambiguous term"
Selected Group: ["Annual Cost", "Order Cost"]
Ambiguous Term: "cost info"
Reason: "cost info is ambiguous enough to cover column of interest and all the columns in the selected group list, avoiding exact match with any one of them"

Example 5:
Column of Interest: "Invoice Allocated Serial Number"
Given List of Similar Columns: ["Inventory Allocated Batch Number", "Order Dispatch Batch Number", "Inventory Allocated Serial Number", "Invoice Allocated Batch Number"]
History: None
Selected Group: ["Inventory Allocated Batch Number", "Invoice Allocated Batch Number"]
Ambiguous Term: "allocated batch number"
Reason: "'allocated number' generalises to column of interest and selected columns, causing ambiguity in the selection of exact column name"

Example 6:
Column of Interest: "Manager_First_Name"
Given List of Similar Columns: ["Manager_Contact", "Manager_Email", "Manager_Designation", "Department"]
History: None
Selected Group: ["Manager_Contact", "Manager_Email", "Manager_Designation"]
Ambiguous Term: "Name"
Reason: "The column of interest 'Manager_First_Name' contains 'Manager' which also appears in the three selected columns and 'Manager' itself is not a column name in the given list."

Example 7:
Column of Interest: "Number of AC DC services"
Given List of Similar Columns: ["Number of AC services", "AC services requested", "Price per AC service", "Number of paid services", "Number of unpaid services"]
History: "'no. of AC services' is an invalid ambiguous term as it represents the actual column "Number of AC services" in the given list"
Selected Group: ["Number of AC services", AC services requested", "Price per AC service"]
Ambiguous Term: "AC services"
Reason: "'AC services' term can cause ambiguity for the column of interest, making it unclear which column is being referred to and generalise to all the columns in the given list"

Example 8:
Column of Interest: "Department"
Given List of Similar Columns: ["Department", "Section", "Category"]
History: None
Selected Group: []
Ambiguous Term: 'Null'
Reason: "There is no column lexically similar to Department, hence an ambiguous term cannot be created"

Example 9:
Column of Interest: "managerName"
Given List of Similar Columns: ["employeeName", "departmentName", "contactNo", "head"]
History: None
Selected Group: ["employeeName", "departmentName"]
Ambiguous Term: "Name"
Reason: "The column of interest 'managerName' contains 'Name' which also appears in the two selected columns and 'Name' itself is not a column name in the given list."

Example 10:
Column of Interest: "transaction_number"
Given List of Similar Columns: ["transaction_id", "customer_id", "customer_contact", "credit_card_transaction", "amount_transaction"]
History: None
Selected Group: ["transaction_id", "credit_card_transaction", "amount_transaction"]
Ambiguous Term: "transaction"
Reason: "The column of interest 'transaction_number' contains 'transaction' which also appears in the three selected columns and 'transaction' itself is not a column name in the given list."

Inputs

Column of Interest:
{coi}

List of Similar Columns:
{subset_columns}

History:
{history_term}

Outputs

Your response must follow these instructions:
{format_instructions}

A/U Term Evaluator Prompt - Lexical Column Ambiguity Case

You are a database analyst specializing in extracting relevant columns from a database. You are given a term and a list of columns. Your task is to analyze the list of columns and the given term to determine whether the term is lexically similar to the columns and represents an ambiguous term, without being an exact match or a direct representation of any of them.

Definition

Ambiguity: Ambiguity arises between a term and list of columns if it is not clear which column name is being referred by the term in the given list. A valid ambiguous term can be linked with more than one column in the given list.
* The following conditions must be met for a valid ambiguous term:
* The ambiguous term should be lexically similar to two or more columns in the list in terms of wording, length, and style.

* Lexical similarity involves sharing same tokens, words or segments.
* If a term can be linked to two or more columns of the given list then it qualifies for valid ambiguity.
* The entire ambiguous term should be linked with two or more columns. If the entire term cannot be linked with the given columns, then it is invalid. For example:
* 'primary contact', the entire term, can be linked to "primary contact name" and "primary contact address" raising ambiguity in column selection.
* 'secondary email' cannot be linked to "contact email" and "secondary address" as the entire term cannot be associated with the mentioned two columns. Such cases do not qualify for valid ambiguity.
* The ambiguous term must not exactly match the name of any column in the provided list.
* An exact match is only allowed if it refers to a single, specific column in the list. This represents invalid ambiguity.
* If the term can be linked with more than one columns, then it holds valid ambiguity.
* The ambiguous term should not be a generic synonym or paraphrased word e.g., if the list contains columns "Employee Title" and "Employee Designation" then "person" term is invalid as it is a synonym of "Employee" and there is no column that could be lexically linked with "person" term.
* **Constraints**
* Ignore the case (uppercase or lowercase), singular plural and minor variations during the comparison and analysis For example:
*'Customer Count', 'customer count', 'CUSTOMER COUNT', 'customer_count' and 'customer counts' represent the same column.
* 'no.', 'number' or '#' represent the same word or token.
* 'Customer Count' and 'Count Customer' represent the same column.
* **Examples fo Valid Ambiguity:**
* "student" is a valid ambiguous term for ["Student ID", "Program Start Date", "Course ID" , "Program End Date", "Student First Name", "Student Last Name"] as it can be linked to multiple columns "Student ID", "Student First Name", and "Student Last Name" columns, causing ambiguity in column selection.
* "Sales - Contact Number" is a valid ambiguous term for ["Sales - Customer Home Number", "Sales - Customer Office Number", "Sales - Customer ID", "Purchases - Employee ID"] as it causes ambiguity in selecting column either "Sales - Customer Home Number" or "Sales - Customer Office Number".
* "employee" is a valid ambiguous term for ["name_employee", "ID_employee", "contact_employee"] as "employee" segment is applicable to all the columns in the list and does not have an exact match with any of the columns.
* "dept" is a valid ambiguous term for ["DeptID", "HeadID", "DeptName","HeadName", "HeadEmail", "DepartEmail"] as it can be maped to columns "DeptID", "DeptName", and "DepartEmail", not clarifying which one is being referred to.
* "cost info" is a valid ambiguous term for ["Cost", "Annual Cost", "Monthly Cost"] as it can be associated with all the columns, as 'info' could be linked with any of the columns.
* "student_name" is a valid ambiguous term for ["student_id", "student_first_name", "student_last_name", "course_enrolled", "program_enrolled"] as the entire term can be linked to "student_first_name" and "student_last_name" columns.
* "given name" is a valid ambiguous term for ["customer id", "customer given name", "customer last name", "employee id", "employee given name", "employee last name" ,"employee contact"] as the term can be linked with "employee given name" and "customer given name" columns.
* **Examples of Invalid Ambiguity:**
* "purchase date" is an invalid ambiguous term for ["Purchase Date", "Purchase Date Approved", "Purchase Date Initiated"] as it has an exact match with "Purchase Date" in the given list.
* "manager employee" is an invalid ambiguous term for ["Manager Employee Name", "Manager Employee StartDate", "Employee Manager", "Manager Employee Contact"] as it can be mapped to "Employee Manager" column.
* "finance dept" is an invalid ambiguous term for ["Finance Department", "Corporate Finance

Department", "Finance Planning Department", "HR Department"] as "finance dept" is another variant of "Finance Department" in the list and does not represent ambiguity.
* "code" is an invalid ambiguous term for ["zipcode", "street number", "unit", "state"] as "code" can directly map to "zipcode". To qualify for ambiguity criterion, the term must be lexically similar to at least two columns.
* "employee salaries" is an invalid ambiguous term for ["Employee ID", "Employee Name", "Employee Salary", "Employee Salary Level", "Employee Designation"] as it represents a variant of "Employee Salary" column. Therefore, it does not qualify for ambiguity criterion.
* "customer_number" is an invalid ambiguous term for ["customer_id", "customer_registration", "customer_contact", "employee_contact", "employee_number"] as the entire term cannot be linked with any of the columns in the list . Therefore, it does not qualify for ambiguity criterion.

Instructions

1. **Analyze**:
* Compare the ‘Given Term‘ with each column of the ‘Given Column List‘ using the given ‘Ambiguity‘ definition.
* Focus on the conditions of the ambiguity criterion to identify valid ambiguous term.
2. **Decide**: Outcome is ‘"valid"‘ if the term is ambiguous. Otherwise, its ‘"invalid"‘.
3. **Explain**: Provide a brief reason for your decision.
4. **Constraints**:
* Base your reason strictly on the provided definitions.
* The reason should be specific to the given term and list of columns.
* Do not use any external knowledge or make assumptions.
5. **Format**: Structure your entire response according to the ‘Output Format‘ instructions

Inputs

Given Term:
{term}

Given List of Columns:
{dataset_columns}

Outputs

Your response must follow these instructions:
{format_instructions}

NL Query Generator Prompt

You are a SQL expert specializing in natural language (NL)-to-SQL translation and data generation.
Your task is to generate a natural language query from the given SQL query. This query is then used to generate its ambiguous version.
The ambiguous version represents a natural language query that cannot be converted into SQL statement as the specification of column is not clear

Instructions

1. **Natural Language Query Generation**:
- Using the given SQL query, write a natural language query that would generate the given SQL query
- Strictly use the column names mentioned in the SQL query and its context to generate the natural language query. Do not make any assumptions
- All the column names mentioned in the SQL query must be explicitly mentioned in the natural language query.
- A reference natural language query is provided. Use similar writing style and tone to generate the natural language query.
2. **Ambiguous Query Generation**:
- You are given a list of column names and corresponding ambiguous terms. Use these to generate an ambiguous query where the given column is replaced with the ambiguous term.
- strictly use the given column name to replace it with the corresponding ambiguous term for the ambiguous query.
- Use the generated natural language query and the given list of ambiguous candidates to generate its ambiguous variant as a user request.
- Ensure the replacements create reasonable uncertainty about column references while still preserving interpretability.
- Retain key contextual elements so that the ambiguous question still guides toward the SQL query but introduces a genuine need for clarification.
- Ensure that the ambiguous version is genuinely unclear about which column is being referenced. Despite the ambiguity, the query should still make logical sense in the context of the SQL statement.
- Strictly base the modifications on the given details and constraints. Do not introduce any assumptions or external knowledge beyond the schema.
3. **Constraints**:
- The generated natural language query and its ambiguous variant should correspond to the same underlying SQL query. The aim of the queries should not change.
- The generated natural language query should be written in a human-like style and should not use the exact column names format found in the SQL query. Use the provided examples as a guide.
- Use natural language variant of the column name e.g., for column name "customer_id", use "customer id"
- If possible, avoid converting a singular ambiguous term into its plural form when generating the natural language query.
- Write concise queries e.g, for where condition "amount>=10", the natural language query could be "for amount greater than or equal to".
- Ensure the natural language query accurately reflects the SQL query and corresponds to each distinct command it contains.
- Do not convert "joins" in the SQL query to natural query instruction.
- Do not mention the database or table name in the generated natural language query.
- Consider the history if available when generating the response.
- The history contains issues with the generated query that mismatches with the given SQL query. Use this history to rewrite the queries to fix the error(s).
- Examples are provided as guide to generate accurate response and avoid any mistakes.
4. **Format**:
- Strictly generate response requested in the output instructions.
- DO NOT GENERATE any other supplementary explanation or description. Strictly generate output as mentioned in the output format

Examples

Example 1: Given SQL Query: 'SELECT count("Monthly Orders"), "Employee ID" FROM my_data GROUP BY "Employee ID" ORDER BY Employee Name ASC'

Given Column Names: ["Monthly Orders"]
Given Ambiguous Terms: ["orders"]
History: None
Reference Natural Language Query: "What is the total number of yearly orders by client ID sorted by client name?"
Generated NL Query: "What are the number of monthly orders by employee id sorted on employee name"?
Generated NL Ambiguous Query: "What are the number of orders by employee id sorted on employee name"?

Example 2: Given SQL Query: 'SELECT "Order Detail", "Start Date" FROM my_data WHERE "Start Date" = "01-01-2010" AND "Employee ID" = 454 ORDER BY Department ASC'
Given Column Names: ["Start Date", "Employee ID"]
Given Ambiguous Terms: ["date", "employee"]
History: None
Reference Natural Language Query: "Display the client information for client id 178 with a start date of 11-11-2000, sorted by country."
Generated NL Question: "display order detail when start date is 01-01-2010 and employee id is 454 sorted on department"
Generated NL Ambiguous Query: "display order detail when date is 01-01-2010 and employee is 454 sorted on department"

Example 3: Given SQL Query: "SELECT Host_Name FROM farm_competition WHERE Theme != 'Fantasy' and Total_Participants<=50"
Given Column Names: ["Host_Name"]
Given Ambiguous Terms: ["host"]
History: None
Reference Natural Language Query: "Return the names of clients for internet services where the company is not 'Newtelco' and number of client are less than or equal to 1000."
Generated NL Question: 'Return the host name of competitions when theme is not "Fantasy" and total participants are less than or equal to 50?'
Generated NL Ambiguous Query: 'Return the host of competitions when theme is not "Fantasy" and total participants are less than or equal to 50?'

Example 4: Given SQL Query: "SELECT T1.product_name, T2.sales_price FROM Products AS T1 JOIN Product_Pricing AS T2 ON T1.product_id = T2.product_id"
Given Column Names: ["product_name", "sales_price"]
Given Ambiguous Terms: ["product", "revenue"]
History: None
Reference Natural Language Query: "Can you provide the sales prices for each product name?"
Generated NL Question: "Can you show me the sales prices for all the product names?"
Generated NL Ambiguous Query: "Can you show me the revenue for all the products?"

Example 5: Given SQL Query: 'SELECT P.ProjectName FROM Project AS P JOIN WorksOn AS W ON P.ProjectID = W.ProjectID GROUP BY P.ProjectID ORDER BY COUNT(*) ASC LIMIT 1'
Given Column Names: ["ProjectName"]
Given Ambiguous Terms: ["name"]
History: None
Reference Natural Language Query: "What project has the lowest number of employees?"
Generated NL Question: "What is the project name with fewest employees?"

Generated NL Ambiguous Query: "What is the project name with fewest employees?"

Example 6: Given SQL Query: "SELECT T1.customer_name, T2.billing_address, T2.order_amount FROM Customers AS T1 JOIN Orders AS T2 ON T1.customer_id = T2.customer_id"
Given Column Name: ["order_amount", "billing_address"]
Given Ambiguous Term: ["expense", "address"]
History: None
Reference Natural Language Query: "What are the names of customers, their billing addresses, and the amounts of their orders?"
Generated NL Question: "What are the customer names, billing addresses, and order amounts?"
Generated NL Ambiguous Query: "What are the customer names, addresses, and expenses?"

Example 7: Given SQL Query: "SELECT C.customer_number, C.residential_address FROM Customers AS C JOIN Service_Requests AS S ON C.customer_number = S.customer_id GROUP BY C.customer_number ORDER BY COUNT(*) DESC LIMIT 1"
Given Column Names: ["residential_address"]
Given Ambiguous Terms: ["personal information"]
History: None
Reference Natural Language Query: "Which customer submitted the highest number of service requests? Display their customer ID and home address."
Generated NL Question: "Which customer has submitted the most service requests? Return the customer number and residential addresses."
Generated NL Ambiguous Query: "Which customer has submitted the most service requests? Return the customer number and personal information."

Example 8: Given SQL Query: 'SELECT "Item Code", "Production Date" FROM my_dataset GROUP BY "Item Code" ORDER BY count(*) DESC LIMIT 1'
Given Column Names: ["Item Code", "Production Date"]
Given Ambiguous Terms: ["item", "date"]
History: None
Reference Natural Language Query: "What item code appears most frequently, and what is its production date?"
Generated NL Question: "What is the most common item code and its production date?"
Generated NL Ambiguous Query: "What is the most common item and its date?"

Example 9: Given SQL Query: 'SELECT "Item Type", avg("Cost") FROM my_dataset GROUP BY "Item ID"'
Given Column Names: ["Item Type"]
Given Ambiguous Terms: ["item"]
History: None
Reference Natural Language Query: "What is the average price for every type of item?"
Generated NL Question: "What is the average cost for each item type?"
Generated NL Ambiguous Query: "What is the average cost for each item?"

Example 10: Given SQL Query: 'SELECT count("Annual Orders"), "Vendor ID" FROM Sales where Year = 2000 GROUP BY "Vendor ID"' '
Given Column Names: ["Annual Orders"]
Given Ambiguous Terms: ["purchases"]
History: None
Reference Natural Language Query: "What is the total number of yearly orders by vendor id for year 2000?"

Generated NL Query: "What are the number of annual orders by vendor id for year 2000"?
Generated NL Ambiguous Query: "What are the number of purchases by vendor id for year 2000"?

Inputs
Given SQL Query:
{expected_query}

Given Column Names:
{cois}

NL Query Evaluator Prompt - Consistency with SQL Query

You are a SQL expert specializing in natural language (NL)-to-SQL translation and data quality refinement.
You are given an ambiguous version of a natural language query. This ambiguity represents the use of generic words/terminology for a column which could be mapped to multiple columns in the database. This confusion can lead to different SQl query translation. Your task is to assess the consistency of the ambiguous version of the given query against the original SQL query.

Instructions

The following instructions must be strictly followed:
1. **Consistency Assessment**: The ambiguous query should be compared with the original SQL query for assessment.
- The ambiguous term(s) and corresponding true column(s) is given that resolves the ambiguity in the given query.
- When the ambiguous term(s) is replaced by the true column(s), the resulting query should be consistent with the underlying SQL query. Specifically, it should meet the following two conditions:
* the query translates to the given SQL query.
* the intention and goal of the given ambiguous query aligns with the SQL query.
2. **Response Generation**: After replacing the ambiguous term(s) with true column(s) in the given ambiguous query, if the resulting query corresponds to underlying SQL query then the outcome is `'valid'`. Otherwise, it is `'invalid'`.
- The valid assessment must address the two consistency condition.
- Provide a reason for the assessment as well.
3. **Constraints**:
- Base your reason strictly on the provided instructions. Response should be specific to given queries, ambiguous term and true column information.
- Do not use any external knowledge or make assumptions.
- Refrain from providing a SQL language-based response and instead, provide a natural language response.
4. **Response Format**: Strictly generate the response using the given output format. DO NOT generate any other content or text.

Examples

Example 1:
Ambiguous Natural Language Query: `'show count of orders by employee sorted on employee name'`
Ambiguous terms: `['orders', 'employee']`

True Columns: ['Monthly Orders', 'Employee ID']
Underlying SQL Query: SELECT count("Monthly Orders"), "Employee ID" FROM my_data GROUP BY "Employee ID" ORDER BY "Employee Name" ASC
Outcome: 'valid'
Reason: 'When "orders" and "employee" in the ambiguous query are replaced by given true columns "Monthly Orders" and "Employee ID", it results in the same expected SQL query. Also, the goal and intention of the query is consistent with the SQL query.'

Example 2:
Ambiguous Natural Language Query: 'display the total detail for employee id 454 and start date of 01-01-2010, sorted by Department."
Ambiguous Terms: ['detail']
True Column: ['Order Detail']
Underlying SQL Query: 'SELECT "Order Detail", "Start Date" FROM my_data WHERE "Start Date" = "01-01-2010" AND "Employee ID" = 454 ORDER BY Department ASC'
Outcome: 'invalid'
Reason: 'When "detail" in the ambiguous query is replace by given true column "Order Details", it resulting query does not matches with the original query and results in a different SQL query to the given one. Specifically, the ambiguous query mentions "total details" that suggests aggregation, which is inconsistent with the underlying SQL query.'

Example 3:
Ambiguous Natural Language Query: 'Can you show me the prices for all the products?"
Ambiguous Terms: ['product', 'price']
True Column: ['product_name', 'billing_price']
Underlying SQL Query: 'SELECT T1.product_name, T2.billing_price FROM Products AS T1 JOIN Product_Pricing AS T2 ON T1.product_id = T2.product_id'
Outcome: 'valid'
Reason: 'When "product" and "price" in the ambiguous query is replace by given true columns "product_name" and "billing_price", it results in the same expected SQL query. Also, the goal and intention of the query is consistent with the underlying SQL query".

Example 4:
Ambiguous Natural Language Query: "Count the number of membership"
Ambiguous Terms: ['membership']
True Column: ['Enrolment ID']
Underlying SQL Query: 'SELECT count(DISTINCT "Enrolment ID") FROM my_dataset'
Outcome: 'invalid'
Reason: 'The SQL query is requesting for distinct "Enrolment ID", whereas the the ambiguous query is not considering the distinct "membership" which is linked with "Enrolment ID". As the ambiguous query is not consistent with the underlying SQL query, so the outcome is invalid".

Inputs

Ambiguous Natural Language Query:
{ac_query}

Ambiguous Terms:
{term}

True Columns:

{cois}

Underlying SQL Query:
{expected_query}

Outputs
Your response must follow these instructions:
{format_instructions}

Data Screening Prompt A

You are a NL2SQL system expert. You are given a natural language query that belongs to one of the four categories:
**Type of Categories*
1. **Lexical Column Ambiguity**: The query includes tokens or terms that refer to a column, but there is no exact match with any column in the given schema. However, the term or token is lexically similar to two or more columns of the schema, making it unclear which column the query is referring to.
- Lexical similarity means that the term shares similar name and writing style or structure with two or more columns of the give schema.
- This ambiguity requires clarification from human for column selection to successfully generate a SQL query. Without this clarification, a SQL query cannot be generated
- Here are a few examples of lexical similarity:
- "orders" term is lexically similar to "Total Orders" and "Yearly Orders" columns
- "ID" term is lexically similar to "Employee_ID", "Department_ID", and "Organisation_ID"
- "allocated batch number" term is lexically similar to "Inventory Allocated Batch Number" and "Invoice Allocated Batch Number"
2. **Semantic Column Ambiguity**: The query includes tokens or terms that refer to a column, but there is no exact match with any column in the schema. However, the term or token is semantically similar to two or more columns of the schema, making it is unclear which column the query is referring to.
- Semantic Similarity means that the term represents higher-level category, concept, synonym, or overlapping ideas in a knowledge space with two or more columns of the schema.
- This ambiguity requires clarification from human for column selection to successfully generate a SQL query. Without this clarification, a SQL query cannot be generated
- Here are a few examples of semantic similarity:
- "issue" is semantically similar to "Security Breach", "Malfunctions in Hardware", and "Hardware Failure" columns
- "expense" is semantically similar to "annual_cost", "order_cost", "registration_fee", and "seller_commission_fee" columns.
- "performance" is semantically similar to "salesAchieved", "projectsCompleted", and "scoreReceived" columns
3. **Column Confusion**: The query contains tokens or terms that cannot be mapped to any columns in the schema. The term or token is neither lexically nor semantically similar to any columns in the schema. The column does not exists in the schema.
4. **Unambiguous**: The query contains tokens or terms referring to a column, with each term being mapped to a single column in the schema. Unambiguous queries can be translated to SQL query without any human intervention

Instructions:
You are given a natural language query that may belong to one of the given categories. Your task is to identify terms or tokens that represents a column name but does not have an exact match with columns of the given schema. Follow the given instructions to achieve this goal.
1. Analyse the natural language query to identify the terms that refers to column in the schema taking into account the given ***Type of Categories**
- Schema can contain multiple tables, compare the terms against each table
- Ignore the case (uppercase or lowercase), singular plural and minor variations during the comparison. For example:
-'Customer Count', 'customer count', 'CUSTOMER COUNT', 'customer_count' and 'customer counts' represent the same entity.
- 'no.', 'number' or '#' represent the same word or token.
- 'Customer Count' and 'Count Customer' represent the same entity.
- 'Identifier', 'ID', or 'identity' represent the same entity.
2. Identify the list of terms that refer to a column but do not exactly match any column names in the schema.
- There could be more than one terms whose exact match does not exist
- If exact match with the schema columns exist for the identified terms then use 'None Found' for response
3. Provide a reason for the assessment. The reason should be specific to the given schema.

Format: Strictly generate the response using the given output format, recording terms and reason seperately. DO NOT generate any other content or text.

Inputs
Datbase Schema:
{db_schema}

Natural Language Question:
{ac_query}

Outputs
{format_instructions}

Data Screening Prompt B

You are a data scientist. You are given two set of list, your task it is assess whether there is an overlap of the elements in the two lists or not.
Instructions:
- Compare the elements of given ‘List 1‘ and ‘List 2‘.
- If there is an overlap of elements between the two lists i.e., the lists share common elements, then the response is 'yes', otherwise, its 'no'.
- The response is 'no' if one of the lists does not contain valid elements for comparison
- Provide a reason for the outcome stating the overlapping or unique elements found during the comparison
- When making the comparison between the two lists, ignore the case (uppercase or lowercase), singular plural and minor variations . For example:
-'Customer Count', 'customer count', 'CUSTOMER COUNT', 'customer_count' and 'customer counts' represent the same entity.
- 'no.', 'number' or '#' represent the same word or token.

- 'customer_count' and 'Count Customer' represent the same entity.
- Strictly generate the response using the given output format. DO NOT generate any other content or text.

List 1: {term}
List 2: {cols_to_compare}

Outputs
{format_instructions}

C Evaluation Tasks - Appendix

No-shot prompt for text-to-SQL queries generation of column ambiguity use case

The task is to write SQL queries based on the provided questions in English. Questions can take the form of an instruction or command and can be ambiguous, meaning they can be interpreted in different ways. In such cases, write all possible SQL queries corresponding to different interpretations and separate each SQL query with ';' and an empty line. Do not include any explanations, and do not select extra columns beyond those requested in the question.

Given the following Database schema:
{schema}
Answer the following:
{question}

No-shot prompt for conversation-to-SQL query generation

You are an NL2SQL expert and you are given a user and agent conversation. This converstaion represent an initial user natural language query that can be ambiguous, which is later resolved by agent and user interaction.

Use this conversation to generate a SQL query. Consider the given database schema to generate a SQL query. If a SQL query cannot be generated then generate "Null" response Do not include any explanations, and do not select extra columns beyond those requested in the question.

Given the following Database schema:
{schema}
Given User-Agent Conversation:
{question}

Prompt for comparing the AmbiSQL results with the ground-truth metadata

Task

You are provided with a natural language question that contains ambiguity, making it impossible to directly convert it into a valid SQL query. A separate model detects this ambiguity and generates a clarification question along with a list of suggested columns that could be used to form a valid SQL query.

Your task is to evaluate the clarification question and suggested columns by comparing them with the ground truth:
- **Underlying Ambiguous Term**: The term in the natural language question that introduces ambiguity.
- **Ground-Truth Columns**: The set of columns that could be used for the underlying ambiguous term to generate a valid SQL query.
- After comparison, make a binary decision:
- True: If the clarification question and suggested columns appropriately address the ambiguity and align with the ground truth.
- False: If they do not properly address the ambiguity or correspond to the correct columns. - Provide a brief explanation justifying your decision

Inputs
Question: {question}

Generated Clarification Question: {clarification}

Suggested Columns: {suggested_columns}

Underlying Ambiguous term: {term}

Ground-Truth Columns: {gt_columns}

D Results - Appendix

Prompting Technique	Unifacet Lexical Amb		Multifacet Lexical Amb		Unifacet Semantic Amb		Multifacet Semantic Amb	
	LEM	SEM	LEM	SEM	LEM	SEM	LEM	SEM
No-shot	18.2	15.3	18	5.1	20.6	14.7	12.3	5.3
without metadata uni-facet	55.9	42.4	61.5	12.82	57.8	43.3	49.1	14.03
without metadata uni-& multi-facet	**60.9**	50.4	66.7	20.5	**64.6**	50.2	56.1	26.3
with metadata uni-facet	60.4	53.9	74.4	61.5	60.3	53.9	59.6	**56.1**
with metadata uni-& multi-facet	58.6	**54.5**	**76.9**	**64.1**	63.9	**57**	**63.2**	**56.1**

Table 13: Few-Shot Single Turn SQL Prediction Task for Column Ambiguity (Amb) Use Cases Using GPT-5 (%). Boldface indicates the comparatively best-performing value.

Prompting with Exemplars	Unifacet Amb		Multifacet Amb	
	LEM	SEM	LEM	SEM
Zero-shot	(17.3, 21.9)	(13, 17.1)	(8.6, 22.6)	(1.1, 10.7)
without metadata uni-facet	(54.2, 60.2)	(40.1, 46)	(46.2, 66.7)	(7.5, 21.5)
without metadata uni-& multi-facet	(60.9, 66.6)	(47.4, 53.4)	(52.7, 72)	(16.1, 33.3)
with metadata uni-facet	(57.6, 63.4)	(51.2, 57)	(58.1, 77.4)	(50.5, 69.9)
with metadata uni-& multi-facet	(58.7, 64.7)	(53.2, 59)	(61.3, 79.5)	(51.6, 71)

Table 14: Confidence intervals for the exact-match accuracy of the few-shot, single-turn SQL prediction task under column ambiguity (Amb) are reported in 4. Confidence intervals are presented as (lower limit, upper limit) and are computed via bootstrapping with replacement over 10,000 resamples.

Model	Unifacet Lexical Amb		Multifacet Lexical Amb		Unifacet Semantic Amb		Multifacet Semantic Amb	
	LEM	SEM	LEM	SEM	LEM	SEM	LEM	SEM
GPT-4o	13.9	12.5	10.3	2.6	15.7	11.6	14	3.5
GPT-4.1	35.1	24.3	41	2.6	36.1	23.4	35.1	5.3
GPT-5	18.2	15.3	18	5.1	20.6	14.7	12.3	5.3
Grok-3 Mini Fast	33.3	22.9	48.7	5.1	34.5	24.4	35.1	8.8
LLaMA-3.3	31.2	21.8	18	2.6	33.8	25.2	26.3	8.8

Table 15: No-Shot Single Turn SQL Prediction Task for Column Ambiguity (Amb) Use Cases (%)

Prompting Technique	Unifacet Lexical Amb		Multifacet Lexical Amb		Unifacet Semantic Amb		Multifacet Semantic Amb	
	LEM	SEM	LEM	SEM	LEM	SEM	LEM	SEM
Zero-shot	13.9	12.5	10.3	2.6	15.7	11.6	14	3.5
without metadata uni-facet	25.5	19.8	25.6	15.4	33	25	31.6	7
without metadata uni-& multi-facet	26.3	21	23	12.8	33.3	26.8	**40.4**	10.5
with metadata uni-facet	29.2	24.3	17.9	7.7	39.7	30.3	15.8	12.3
with metadata uni-& multi-facet	**38.6**	**35.7**	**35.9**	**33.3**	**47.9**	**44.8**	22.8	**17.5**

Table 16: Few-Shot Single Turn SQL Prediction Task for Column Ambiguity (Amb) Use Cases Using GPT–4o(%). Boldface indicates the comparatively best-performing value

Prompting Technique	Unifacet Lexical Amb		Multifacet Lexical Amb		Unifacet Semantic Amb		Multifacet Semantic Amb	
	LEM	SEM	LEM	SEM	LEM	SEM	LEM	SEM
Zero-shot	35.1	24.3	41	2.6	36.1	23.4	35.1	5.3
without metadata uni-facet	39.2	28.2	41	10.3	49.8	36.8	40.4	14
without metadata uni-& multi-facet	39	29	43.6	10.3	49.8	38.6	40.4	10.5
with metadata uni-facet	57.1	52.9	69.2	53.8	64.2	59	50.9	**45.6**
with metadata uni-& multi-facet	**58.6**	**55.1**	**71.8**	**61.5**	**64.9**	**59.6**	**56.1**	40.4

Table 17: Few-Shot Single Turn SQL Prediction Task for Column Ambiguity (Amb) Use Cases Using GPT–4.1(%). Boldface indicates the comparatively best-performing value

Prompting Technique	Unifacet Lexical Amb		Multifacet Lexical Amb		Unifacet Semantic Amb		Multifacet Semantic Amb	
	LEM	SEM	LEM	SEM	LEM	SEM	LEM	SEM
Zero-shot	33.3	22.9	48.7	5.1	34.5	24.4	35.1	8.8
without metadata uni-facet	47.3	35.5	46.2	10.3	53.4	41.8	43.9	12.3
without metadata uni-& multi-facet	49.4	38.4	41	15.4	58.3	45.9	49.1	19.3
with metadata uni-facet	59.8	**54.7**	**71.8**	59	61.8	57	50.9	47.4
with metadata uni-& multi-facet	**60.2**	**54.7**	**71.8**	**61.5**	**64**	**60.3**	**58**	**54.4**

Table 18: Few-Shot Single Turn SQL Prediction Task for Column Ambiguity (Amb) Use Cases Using Grok-3 Mini Fast (%). Boldface indicates the comparatively best-performing value

Prompting Technique	Unifacet Lexical Amb		Multifacet Lexical Amb		Unifacet Semantic Amb		Multifacet Semantic Amb	
	LEM	SEM	LEM	SEM	LEM	SEM	LEM	SEM
Zero-shot	31.2	21.8	18	2.6	33.8	25.2	26.3	8.8
without metadata uni-facet	52.5	36.5	61.5	10.3	58.7	44.6	52.6	19.3
without metadata uni-& multi-facet	53.9	40.2	59	20.5	62.4	49.5	49.1	17.5
with metadata uni-facet	60.6	56.9	59	46.1	65.8	61.3	50.9	**40.4**
with metadata uni-& multi-facet	**63.3**	**59.4**	**69**	**48.7**	**67.3**	**63.2**	**56.1**	35.1

Table 19: Few-Shot Single Turn SQL Prediction Task for Column Ambiguity (Amb) Use Cases Using LLaMA-3.3 (%). Boldface indicates the comparatively best-performing value

Model	A/U Use Cases								Conversation Types			
	U Lx	M Lx	U Sm	M Sm	U Col Unans	M Col Unans	U Val Amb	M Val Amb	Concise	Verbose	Partially	Not
GPT-4o	62.8	70.8	55.6	53.6	94	99	39.8	43.2	67	47	39.5	57
GPT-4.1	64.8	72.4	62.2	65.6	87	98	42.1	47.6	65.2	47.3	43.4	63.8
GPT-5	71.8	67.6	64	65	60	83	45.1	52	54.9	53.9	51.5	60.4
Grok-3 Mini Fast	66.2	66.2	61.8	66.8	64	82	44.1	53.8	54.3	48.2	44.8	65.1
LLaMA-3.3	66	64.4	65.6	52	19	47	38.9	43	30.6	45.7	42.7	62.5

Table 20: Multi Turn SQL Prediction Task Execution Macro Accuracy (EA-%) for BIRD Dataset; U - Uni-facet; M - Multi-facet; Col - Column; Val - Value; Amb - Ambiguity; Unans - Unanswerable; Lx - Lexical Column Amb; Sm - Semantic Column Amb;

Dataset	**Method**	**U Col Amb**	**M Col Amb**	**U Val Amb**	**M Val Amb**
Spider	AmbiSQL	(44.8, 69.5)	(12.5, 29.2)	(15, 29.2)	(23.8, 51.1)
	AmbiSQL-CT	(59.3, 64.9)	(7.3, 26.8)	(13.1, 27.9)	(34, 48.4)
BIRD	AmbiSQL	(49.7, 59.4)	(0, 9)	(34.6, 40.9)	(16.3, 40.8)
	AmbiSQL-CT	(52.2, 61.9)	(0, 15.5)	(36.5, 46.1)	(20.4, 46.9)

Table 21: Confidence intervals for ambiguity detection task performance (DA%) in Tables 6 and 7. Confidence intervals are presented as (lower limit, upper limit) and are computed via bootstrapping with replacement over 10,000 resamples.

Dataset	**Method**	U Lex Amb		M Lex Amb		U Sem Amb		M Sem Amb		U Val Amb		M Val Amb	
		DA	MA	DA	MA	DA	MA	DA	MA	DA	MA	DA	MA
Spider	AmbiSQL	99.4	58.6	**100**	**15.4**	99.7	56.9	**100**	25.4	98	**22**	96.8	35.1
	AmbiSQL - CLARITY Exemplars	**99.8**	**64.3**	**100**	12.8	**99.8**	**60**	**100**	**26.3**	**100**	21.8	**100**	**40.8**
BIRD	AmbiSQL	98.6	55.2	**100**	**7.7**	**99.2**	45	**100**	3.1	**98.8**	32.1	98	26
	AmbiSQL - CLARITY Exemplars	**99.3**	**60.7**	**100**	**7.7**	**99.2**	**54.6**	**100**	**6.3**	**98.8**	**40.9**	**98**	**32**

Table 22: Ambiguity Detection Task Performance (%) Using GPT-5. Boldface indicates the comparatively best-performing value

Dataset	**Method**	U Lex Amb		M Lex Amb		U Sem Amb		M Sem Amb		U Val Amb		M Val Amb	
		DA	MA	DA	MA	DA	MA	DA	MA	DA	MA	DA	MA
Spider	AmbiSQL	89	49.7	93.6	33.3	94	61.6	99.1	45.6	98	14	97.4	40.7
	AmbiSQL - CLARITY Exemplars	93.1	50.8	94.9	41	96.2	59	100	52.6	98.4	2.5	95.4	38.3
BIRD	AmbiSQL	94.5	40	100	7.7	96.2	33.9	96.9	18.8	96	27.3	96	24
	AmbiSQL - CLARITY Exemplars	94.5	43.5	100	30.8	98.8	44.2	100	28.1	95.7	28.3	98	24

Table 23: Ambiguity Detection Task Performance (%) Using GPT-4o.

Dataset	Method	U Lex Amb		M Lex Amb		U Sem Amb		M Sem Amb		U Val Amb		M Val Amb	
		DA	MA	DA	MA	DA	MA	DA	MA	DA	MA	DA	MA
Spider	AmbiSQL	84.6	46.6	100	38.5	94	61.8	97.4	48.2	98	31	84.6	33.7
	AmbiSQL - CLARITY Exemplars	88.2	53.9	94.9	46.2	95.8	73.2	100	63.2	99.2	31.1	88.8	57.7
BIRD	AmbiSQL	91	49	100	38.5	98.9	46.5	96.9	18.8	93.1	36.1	94	46
	AmbiSQL - CLARITY Exemplars	95.9	57.9	100	46.2	98.5	56.2	100	25	96.4	43	98	62

Table 24: Ambiguity Detection Task Performance (%) Using GPT-4.1.

Dataset	Method	U Lex Amb		M Lx Amb		U Sm Amb		M Sm Amb		U Val Amb		M Val Amb	
		DA	MA	DA	MA	DA	MA	DA	MA	DA	MA	DA	MA
Spider	AmbiSQL	91.3	59.8	94.9	28.2	96.1	72.6	100	24.6	96	21	91.2	51.2
	AmbiSQL - CLARITY Exemplars	89.6	63.3	89.7	15.4	96.2	76	98.3	24.6	97.5	25.4	89.3	66.3
BIRD	AmbiSQL	95.2	63.5	92.3	15.4	98.9	58.5	100	6.3	93.6	40.6	96	36
	AmbiSQL - CLARITY Exemplars	91.7	62.8	100	23.1	98.5	63.8	100	15.6	91	40.4	98	36

Table 25: Ambiguity Detection Task Performance (%) Using Grok-3 Mini Fast.

Dataset	Method	U Lex Amb		M Lex Amb		U Sem Amb		M Sem Amb		U Val Amb		M Val Amb	
		DA	MA	DA	MA	DA	MA	DA	MA	DA	MA	DA	MA
Spider	AmbiSQL	97	60.9	98.7	50	97.5	70	93	47.34	87	21	94	43.9
	AmbiSQL - CLARITY Exemplars	94.5	58	100	53.9	95.9	70.1	96.5	59.6	92.6	18.9	96.4	49
BIRD	AmbiSQL	95.2	61.4	100	21.9	97.3	55.8	100	21.9	92.2	30.6	94	54
	AmbiSQL - CLARITY Exemplars	89.7	57.2	100	15.4	97.3	54.2	96.9	25	95.2	40.6	96	50

Table 26: Ambiguity Detection Task Performance (%) Using LLaMA-3.3.