ASKING WHAT MATTERS: DISAGREEMENT-GUIDED AMBIGUITY DETECTION FOR INTERACTIVE TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

Interactive text-to-SQL requires an LLM agent not only to generate executable SQL, but also to decide which uncertainty to clarify within a limited interaction budget. However, current clarification policies have three limitations: (i) they target surface ambiguity without testing its SQL impact; (ii) they miss ambiguities implicit in schema routes or domain knowledge; and (iii) they spend limited turns on unfocused or redundant questions. These limitations expose a critical challenge: identifying SQL-impacting ambiguity before the intended SQL is known. In this paper, we propose **DISGUAD**, a **DIS**agreement-**GU**ided **A**mbiguity **D**etection framework that specializes clarification selection while leaving the downstream SQL generator unchanged. DISGUAD treats disagreement among plausible SQL interpretations as direct evidence of which uncertainty matters to the final query. Specifically, it constructs a query-term inventory, elicits contrastive SQL interpretations through complementary probe families, compares their canonical SQL components, and attributes recurring differences to user-facing clarification targets. A lightweight prioritization and dialogue-state mechanism then selects one high-impact unresolved target for clarification. Experiments on BIRD-INTERACT Lite and Full with two LLM backbones demonstrate consistent improvements in task success and clarification quality. Compared with the original interaction policy, DISGUAD improves normalized Reward by 4.40–8.97 points and raises overall critical-ambiguity coverage from 48.62% to 82.75% and from 45.76% to 85.14%. Its clarification signals further improve two independently designed static text-to-SQL frameworks by up to 13.97 Reward points while keeping their SQL generators fixed. These results demonstrate that disagreement among plausible SQL queries provides actionable evidence for selecting clarification targets that matter to downstream generation. Our code is available at https://anonymous.4open.science/r/DisGuAD/.

1 INTRODUCTION

Generating executable SQL requires committing to decisions that users do not always state explicitly (Saparina & Lapata, 2024; Ding et al., 2026; Liu et al., 2026). A request may leave undefined how a metric should be computed, which schema entity a phrase refers to, or what filtering, aggregation, or ranking criterion should be applied (Yu et al., 2019a;b). Although LLM-based text-to-SQL systems can generate increasingly complex queries (Hong et al., 2025; Liu et al., 2025b), they are commonly formulated under the assumption that the input question fully specifies the intended semantics (Yu et al., 2018; Li et al., 2023b). This assumption breaks down when multiple reasonable interpretations lead to different SQL queries. Interactive text-to-SQL addresses such ambiguity by allowing an LLM agent to ask clarification questions before generating the final query (Huo et al., 2026). This setting is no longer only about producing SQL from a given request. The agent must also determine *what to ask*: among many linguistically plausible uncertainties, it must identify the one whose resolution is most useful for the resulting SQL query (Liu et al., 2025a).

Existing interactive pipelines commonly rely on the LLM itself to identify ambiguity and immediately verbalize a clarification question (Gao et al., 2024; Huo et al., 2026). Although flexible, this strategy does not explicitly test whether the selected uncertainty changes the SQL semantics.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

It consequently exhibits three limitations. (i) **Surface-level targeting:** a question may focus on an expression that appears linguistically underspecified but has little effect on projection, filtering, joins, aggregation, or other SQL decisions (Saparina & Lapata, 2024). (ii) **Incomplete ambiguity coverage:** consequential uncertainty is often implicit in a schema route, domain-specific metric, threshold, or external definition, and may therefore be missed by clarification based only on the surface form of the request (Huo et al., 2026). (iii) **Unfocused interaction:** the agent may combine multiple targets in one question or revisit an ambiguity that has already been queried, consuming interaction turns without resolving additional SQL decisions (Yao et al., 2019). These failures are visible in our clarification-target analysis: under two LLM backbones, the original interaction policy covers fewer than half of the benchmark-annotated critical ambiguities, with overall coverage of only 48.62% and 45.76%. The limitation is therefore not simply whether the agent can ask a fluent question, but whether it can locate the uncertainty that matters to the final query.

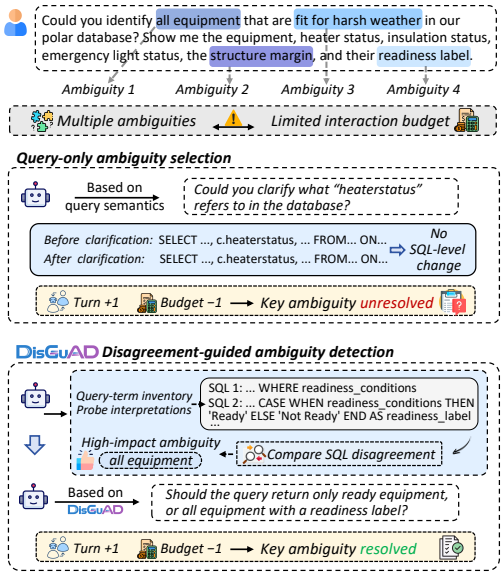


Figure 1: Ambiguity selection in interactive text-to-SQL. A query-only policy asks about `heaterstatus`, leaving the SQL unchanged and the key ambiguity unresolved. DisGuAD instead uses SQL disagreement to identify and resolve the high-impact result-scope ambiguity.

Effective clarification poses a critical challenge: identifying *SQL-impacting ambiguity* before the intended SQL is known. The key question is not whether a phrase admits several linguistic interpretations, but whether plausible interpretations of that phrase induce different SQL decisions. This observation suggests a direct source of evidence (Wang et al., 2024; Li et al., 2024; Pourreza et al., 2025; Li et al., 2026b;a). If independently elicited interpretations disagree on a projection, predicate, join path, aggregation level, ranking rule, or database operation, then the assumption responsible for that disagreement is a useful clarification target. Conversely, uncertainty that does not alter the induced SQL provides weaker evidence for interaction (Farquhar et al., 2024). We therefore treat disagreement between plausible SQL queries as an ambiguity sensor: contrastive SQL generation exposes alternative decisions, component-level comparison identifies where they diverge, and attribution maps those differences back to concepts that a user can resolve (Manakul et al., 2023; Kuhn et al., 2023).

In this paper, we propose **DisGuAD**, a **Dis**agreement-**Gu**ided **A**mbiguity **D**etection framework for interactive text-to-SQL. DisGuAD is a plug-in clarification policy that leaves the downstream SQL generator unchanged. Given an ambiguous request, it first constructs a query-term inventory containing potential natural-language anchors. It then elicits contrastive SQL interpretations through four complementary probe families that vary major query subgoals, relational routes, implicit definitions, and individual ambiguity axes. DisGuAD canonicalizes the generated queries, identifies recurring component-level disagreements, and maps the resulting evidence back to user-facing ambiguity candidates. A lightweight prioritization and dialogue-state mechanism selects the highest-value unresolved target and converts it into a deterministic clarification question. Experiments under the official conversational setting of BIRD-Interact (Huo et al., 2026) show consistent improvements on both the Lite and Full benchmarks with two LLM backbones. Compared with the original interaction policy, DisGuAD improves normalized Reward by 4.40–8.97 points and raises overall critical-ambiguity coverage from 48.62% to 82.75% and from 45.76% to 85.14% for the two backbones. Moreover, replacing the original clarification dialogue with DisGuAD interaction improves two independently designed static text-to-SQL frameworks (Gao et al., 2024; Pourreza & Rafiei, 2023), yielding gains of up to 13.97 Reward points while keeping their SQL generators fixed. These results show that SQL-query disagreement provides actionable evidence for selecting clarification targets and produces interaction signals that transfer across downstream generation pipelines.

Our contributions are summarized as follows:

- We identify SQL-impacting ambiguity selection as a central challenge in interactive text-to-SQL and formulate disagreement between plausible SQL queries as direct evidence of which uncertainty should be clarified.
- We propose DISGUAD, a plug-in framework that combines query-term extraction, contrastive SQL interpretation probes, component-level disagreement analysis, and user-facing ambiguity attribution without modifying the final SQL generator.
- Extensive experiments on BIRD-INTERACT demonstrate consistent improvements across two benchmark variants and LLM backbones. Further analysis shows substantially higher critical-ambiguity coverage and confirms that DISGUAD interaction generalization.

2 RELATED WORK

Traditional text-to-SQL methods formulate the task as single-turn semantic parsing (Wang et al., 2020; Li et al., 2023a). Recent LLM-based systems further employ task decomposition, in-context example selection, candidate generation, and self-correction (Pourreza & Rafiei, 2023; Qu et al., 2024; Xie et al., 2025; Li et al., 2025; Lei et al., 2025; Hong et al., 2026). Interactive semantic parsing introduced model-based agents that decide when and where to request user feedback (Yao et al., 2019). Recent work studies ambiguity more directly: AmbiQT (Bhaskar et al., 2023) and AMBROSIA (Saparina & Lapata, 2024) associate ambiguous utterances with multiple plausible SQL interpretations, AmbiSQL (Ding et al., 2026) detects ambiguity using a fine-grained taxonomy and elicits multiple-choice clarification, and BIRD-Interact evaluates multi-turn agents that interact with users and database environments (Huo et al., 2026). In parallel, consistency-based LLM methods compare sampled generations to estimate semantic uncertainty or detect unreliable content (Kuhn et al., 2023; Manakul et al., 2023). DISGUAD instead uses component-level SQL disagreement to select user-facing clarification targets; extended comparisons appear in the Appendix A.

3 METHODS

3.1 PRELIMINARIES AND OVERVIEW

Task Formulation. Given a natural language user question q and a database $\mathcal{D}$ with schema $\mathcal{S}$, the conventional text-to-SQL task learns a mapping f converting the question into SQL query:

$$f : (q, \mathcal{S}) \to y, \tag{1}$$

where y is the SQL query that expresses the intent of q. This formulation assumes that the user question fully specifies the intended query. Interactive text-to-SQL relaxes this assumption by presenting the agent $\mathcal{M}_\alpha$ with an ambiguous question $\tilde{q}$. The agent interacts with a user $\mathcal{U}$ to acquire the missing information before generating the final SQL query. We define the environment as $\mathcal{E} = (\mathcal{D}, \mathcal{S}, \mathcal{K})$, where $\mathcal{K}$ denotes the available external knowledge, and let H_t denote the dialogue history through turn t. At each non-final turn, the agent asks a question $q_t^{(\text{ASK})}$, receives a response (observation) o_t, and updates the history for subsequent interaction:

$$q_t^{(\text{ASK})} = \mathcal{M}_\alpha^{(\text{ASK})}\big(\tilde{q}, \mathcal{E}, H_{t-1}\big),\ o_t = \mathcal{U}\big(q_t^{(\text{ASK})}, H_{t-1}\big),\ H_t = H_{t-1} \oplus (q_t^{(\text{ASK})}, o_t). \tag{2}$$

where $\oplus$ appends the new question–response pair and $H_0 = \varnothing$. At the terminal turn T, the agent produces the final PostgreSQL query $\widehat{y} = \mathcal{M}_\alpha^{(\text{SQL})}(\tilde{q}, \mathcal{E}, H_{T-1})$ at turn T. The superscripts distinguish the question-generation and SQL-generation modes of the same agent with parameters α.

DISGUAD Overview. Our proposed framework DISGUAD is a plug-in clarification policy for the interactive setting above. DISGUAD specializes clarification selection while leaving the underlying agent $\mathcal{M}_\alpha$ and its final SQL generator unchanged. Its central idea is to represent ambiguity through *SQL query disagreement*: if two plausible interpretations of the ambiguous question $\tilde{q}$ induce different SQL decisions, the uncertainty responsible for that difference is a useful clarification target. DISGUAD exposes such differences with contrastive SQL probes and maps them back to concepts that the user can clarify. The complete workflow can be formalized as:

$$q_t^{(\text{ASK})} = \text{DISGUAD}\big(\tilde{q}, \mathcal{S}, \mathcal{K}, H_{t-1}\big). \tag{3}$$

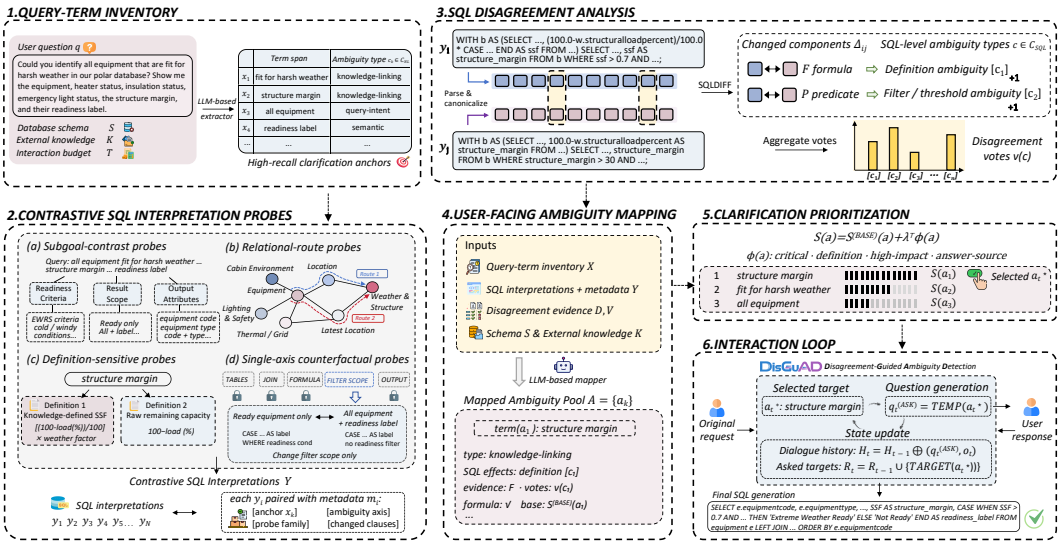


Figure 2: Overview of DISGUAD. It extracts clarification anchors, elicits contrastive SQL interpretations, and aggregates disagreement. The evidence is mapped to user-facing terms, prioritized against the dialogue state, and converted into a clarification question before SQL generation.

DISGUAD first elicits contrastive SQL interpretations of $\tilde{q}$, identifies the semantic decisions on which these interpretations disagree, and maps the supported disagreements back to concepts that can be clarified by the user. Finally, it selects the highest-priority unresolved ambiguity and verbalizes it using a deterministic question template. The following sections instantiate each component.

3.2 QUERY-TERM INVENTORY

The query-term inventory defines the natural-language anchors available for clarification. We use an LLM-based extractor to identify short spans in $\tilde{q}$ and assign each span a coarse ambiguity type:

$$\mathcal{X} = \textsc{Ext}(\tilde{q}, \mathcal{S}, \mathcal{K}; \widetilde{\mathcal{X}}) = \{(x_k, c_k)\}_{k=1}^{M},\ c_k \in \mathcal{C}^{(\mathrm{NL})}, \tag{4}$$

where x_k is a short text span in $\tilde{q}$ that may serve as a clarification anchor, and c_k denotes its coarse ambiguity type. The auxiliary set $\widetilde{\mathcal{X}}$ contains heuristically identified candidates supplied as weak extraction evidence, while $\mathcal{C}^{(\mathrm{NL})}$ comprises lexical, syntactic, semantic, schema-linking, query-intent, and knowledge-linking ambiguity types. These heuristic candidates improve extraction coverage but do not determine which term should ultimately be queried. Thus, $\mathcal{X}$ identifies potential clarification anchors, whose SQL relevance is determined by the subsequent disagreement analysis.

3.3 CONTRASTIVE SQL INTERPRETATION PROBES

To expose alternative SQL decisions, DISGUAD elicits N plausible PostgreSQL interpretations through four controlled probe families $\mathcal{P} = \{\texttt{SUB}, \texttt{REL}, \texttt{DEF}, \texttt{CF}\}$. This controlled elicitation is designed to reduce incidental variation caused by repeatedly sampling from one single prompt. For the i-th probe, let $p_i \in \mathcal{P}$ denote its probe family. We generate a plausible SQL query together with metadata describing the assumptions under which it was produced:

$$(y_i^{(\mathrm{PRB})}, m_i) = \textsc{Prb}_{p_i}(\tilde{q}, \mathcal{S}, \mathcal{K}, \mathcal{X}), \tag{5}$$

where $y_i^{(\mathrm{PRB})}$ is a plausible PostgreSQL interpretation and m_i is the metadata describing its underlying assumptions, including the probe family, ambiguity axis, anchored span, expected changed clauses, referenced schema elements, and relevant knowledge identifiers.

The four probe families expose complementary sources of query-level variation: **subgoal contrast** (SUB) varies major query decisions, including projections, tables, predicates, aggregations, ordering, limits, and database operations; **relational route** (REL) explores alternative table paths, joins,

grouping levels, filter scopes, and final projections; **definition sensitivity** (DEF) tests alternative definitions of scores, rates, ratios, and other domain-specific metrics; and **single-axis counterfactuals** (CF) vary one ambiguity axis while holding the remaining SQL decisions fixed.

Across these families, m_i traces each query variation back to the assumption that produced it. The probe-generated SQL queries serve only as diagnostic evidence for the subsequent disagreement analysis; none is selected or reranked as the final answer.

3.4 SQL Disagreement Analysis

Component-level Disagreement. Each probe-generated SQL query is parsed into the decisions that define its execution semantics. These decisions include the database operation (e.g., SELECT, UPDATE, or DELETE), projected attributes, referenced tables and join paths, filtering predicates, derived formulas, grouping keys, HAVING conditions, ordering criteria, and row limits. We canonicalize each component before comparison so that formatting, aliases, and other surface-level variations do not create spurious disagreements. For each pair of probe outputs, we then define:

$$\Delta_{ij} = \text{SQLDIFF}\left(y_i^{(\text{PRB})}, y_j^{(\text{PRB})}\right), \ 1 \le i < j \le N. \tag{6}$$

The operator returns the canonical SQL components on which the two interpretations disagree. Thus, Δ_{ij} captures structural disagreement between the SQL queries rather than textual variation.

Disagreement Voting. We group component-level differences into a predefined set of SQL ambiguity types $\mathcal{C}^{(\text{SQL})}$, including projection, schema or join, filter or threshold, definition or granularity, ranking, and database-operation ambiguity. The operator $\text{TYPE}(\Delta_{ij})$ returns the subset of these ambiguity types indicated by the changed components in Δ_{ij}. Because a difference observed in only one probe pair may be incidental, we define the disagreement vote for each type $c \in \mathcal{C}^{(\text{SQL})}$ as

$$v(c) = \sum_{1 \le i < j \le N} \mathbb{1}\left[c \in \text{TYPE}(\Delta_{ij})\right]. \tag{7}$$

Here, $\mathbb{1}[\cdot]$ is the indicator function, and $v(c)$ counts the number of probe pairs that disagree on ambiguity type c. A larger $v(c)$ provides stronger disagreement evidence that the corresponding ambiguity affects the resulting SQL query; smaller value suggests that the ambiguity may be incidental.

3.5 Mapping SQL Disagreement to User-Facing Ambiguity

Component-level disagreements identify unstable SQL decisions, but clause names and schema elements cannot be presented directly as clarification questions. DisGuAD therefore uses an LLM-based mapper to attribute the disagreement evidence to the clarification anchors in $\mathcal{X}$:

$$\mathcal{A} = \text{MAP}\Big(\mathcal{X}, \ \{(y_i^{(\text{PRB})}, m_i)\}_{i=1}^{N}, \ \{\Delta_{ij}\}_{i<j}, \ \{v(c)\}_{c \in \mathcal{C}^{(\text{SQL})}}, \ \mathcal{S}, \mathcal{K}\Big) = \{a_\ell\}_{\ell=1}^{L}. \tag{8}$$

Here, $\mathcal{A}$ is the resulting pool of user-facing ambiguity candidates. Each $a_\ell \in \mathcal{A}$ represents a clarification target grounded in the original request and records its ambiguity type, affected SQL decisions, supporting probe interpretations, disagreement vote, definition requirements, and base priority.

The mapper attributes observed changes in projections, predicates, joins, grouping, ordering, limits, and database operations to the interpretation of an anchor in $\mathcal{X}$. This attribution is particularly important for knowledge-intensive expressions such as "stability score", "release decision rule", or "risk indicator", whose SQL realizations may depend on a formula, threshold, JSON path, or external definition rather than a single schema attribute. Terms grounded in $\mathcal{K}$ may also be considered when $\tilde{q}$ refers to the corresponding concept. SQL artifacts such as clause names, table aliases, and generated column aliases are excluded, while generic spans such as "ID", "data" or "result" are retained only when the disagreement evidence reveals a concrete schema or projection conflict.

3.6 Clarification Prioritization and Dialogue State

Ambiguity Prioritization. DisGuAD assigns each mapped ambiguity $a \in \mathcal{A}$ a priority score:

$$\text{S}(a) = \text{S}^{(\text{BASE})}(a) + \boldsymbol{\lambda}^\top \boldsymbol{\phi}(a), \tag{9}$$

where $\mathrm{S}(a)$ is the target score and $\mathrm{S}^{(\textsc{Base})}(a)$ is the mapper-assigned base priority; $\boldsymbol{\phi}(a) \in \{0,1\}^4$ encodes whether a is critical, definition-dependent, high-impact, or associated with an identifiable answer source, and $\boldsymbol{\lambda} \in \mathbb{R}^4_{\geq 0}$ contains the weights to control the contribution of these properties.

Dialogue State. To avoid repeated questions, $\mathcal{R}_t$ records the normalized clarification targets queried through turn t, with $\mathcal{R}_0 = \varnothing$. Let $\textsc{Target}(a)$ denote the normalized clarification target associated with a. The eligible ambiguity set at turn t is:

$$\mathcal{A}_t = \{a \in \mathcal{A} \mid \textsc{Askable}(a) \wedge \textsc{Target}(a) \notin \mathcal{R}_{t-1}\}, \tag{10}$$

where $\textsc{Askable}(a)$ indicates whether a can be verbalized as a user-answerable question, and $\textsc{Target}$ returns its normalized clarification target. When $\mathcal{A}_t$ is nonempty, DisGuAD selects the highest-priority candidate and converts it into a clarification question:

$$a_t^\star = \arg\max_{a \in \mathcal{A}_t} \mathrm{S}(a),\ q_t^{(\textsc{Ask})} = \textsc{Temp}(a_t^\star). \tag{11}$$

The template $\textsc{Temp}(\cdot)$ requests a definition or formula for definition ambiguities, an ordering criterion or row limit for ranking ambiguities, an intended entity or attribute for schema ambiguities, and the intended action for operation ambiguities. If no mapped ambiguity is eligible, DisGuAD first selects an unqueried anchor from $\mathcal{X}$, which is ranked below all disagreement-supported candidates. If neither source provides an eligible target, question generation falls back to the underlying agent $\mathcal{M}_\alpha^{(\textsc{Ask})}$. After a mapped or inventory-based question is issued, its normalized target is added to $\mathcal{R}_t$. Under the benchmark protocol, the resulting question is emitted as `<s>[QUESTION]</s>`.

4 Experiments

4.1 Experimental Setups

Dataset. We use BIRD-Interact (Huo et al., 2026), a dynamic interactive text-to-SQL benchmark that evaluates SQL generation through multi-turn interaction. Each task consists of an ambiguous priority question followed by a context-dependent follow-up question in a PostgreSQL environment with database metadata and hierarchical external knowledge. We evaluate our method on both BIRD-Interact-Full and BIRD-Interact-Lite under the official c-Interact setting, where the system follows a predefined clarification protocol before submitting the final SQL query.

Evaluation Metrics. Following BIRD-Interact, we report **Priority Question Success Rate (PQ-SR)**, **Follow-up Question Success Rate (FQ-SR)**, and **Normalized Reward (Reward)**. PQ-SR measures the success rate on the initial priority question, while FQ-SR measures the success rate on the context-dependent follow-up question. A SQL query is counted as successful only if it passes all executable test cases. Reward is the official normalized task-level score on a 0–100 scale, jointly measuring performance across the priority and follow-up questions.

Compared Methods. We compare two groups of methods. For interactive text-to-SQL, **Vanilla** directly generates SQL without clarification, **BIRD-Interact** uses the benchmark's standard interaction pipeline, and **DisGuAD** replaces the default ambiguity-selection policy with disagreement-guided clarification. To evaluate transferability, we inject clarification signals into two representative static text-to-SQL frameworks, **DAIL-SQL** (Gao et al., 2024) and **DIN-SQL** (Pourreza & Rafiei, 2023). For each framework, we compare **Vanilla**, **w/ Original**, and **w/ DisGuAD**, where the latter two add the clarification dialogue from the original policy or DisGuAD. Because the downstream SQL generator is fixed, this comparison isolates the value of the clarification signal.

Implementation Details. In all main experiments, DisGuAD generates $K = 8$ SQL candidates for each ambiguous user query. The candidate budget is evenly distributed across four complementary reasoning paths. We instantiate DisGuAD with two LLM backbones: DeepSeek-V4-Flash and GPT-5.6-Luna. For each experiment, we use the same backbone for the system model, user simulator, and ambiguity module to avoid confounding effects from cross-model mismatch. All LLM calls are made through the official model APIs. Experiments are conducted in the conversational setting using the official BIRD-Interact Docker evaluator for evaluation, together with the PostgreSQL database containers provided by the benchmark. Local orchestration, database management, and evaluation scripts are run on a machine equipped with six NVIDIA RTX 3090 GPUs.

Table 1: Main results on BIRD-INTERACT-LITE and BIRD-INTERACT-FULL. We report Priority Question Success Rate (PQ-SR), Follow-up Question Success Rate (FQ-SR), and normalized Reward. The upper block evaluates DISGUAD as an interactive text-to-SQL method, while the lower block evaluates whether DISGUAD-selected signals transfer to static text-to-SQL frameworks. Best results within each comparable group are in bold, and second-best results are underlined. Parenthesized gains are measured against the corresponding Original setting.

Framework	Setting	LITE			FULL		
		PQ-SR ↑	FQ-SR ↑	Reward ↑	PQ-SR ↑	FQ-SR ↑	Reward ↑
DeepSeek-V4-Flash							
Vanilla	No interaction	16.00	9.00	13.90	15.83	9.50	13.93
BIRD-INTERACT	Original	43.00	28.00	38.50	29.00	16.50	25.25
DISGUAD	DISGUAD	**50.67**(+7.67)	**31.33**(+3.33)	**44.87**(+6.37)	**33.83**(+4.83)	**21.00**(+4.50)	**29.98**(+4.73)
DAIL-SQL	Vanilla	10.00	5.33	8.60	10.67	6.17	9.32
DAIL-SQL	w/ Original	48.67	31.33	43.47	42.67	26.83	37.92
DAIL-SQL	w/ DISGUAD	**52.33**(+3.66)	**32.33**(+1.00)	**46.33**(+2.86)	**44.50**(+1.83)	**27.83**(+1.00)	**39.50**(+1.58)
DIN-SQL	Vanilla	10.00	5.00	8.50	11.83	7.17	10.43
DIN-SQL	w/ Original	50.67	32.00	45.07	41.83	25.67	36.98
DIN-SQL	w/ DISGUAD	**55.00**(+4.33)	**34.00**(+2.00)	**48.70**(+3.63)	**45.00**(+3.17)	**28.50**(+2.83)	**40.05**(+3.07)
GPT-5.6-Luna							
Vanilla	No interaction	17.00	9.00	14.60	18.17	10.17	15.77
BIRD-INTERACT	Original	30.00	18.33	26.50	21.67	12.17	18.82
DISGUAD	DISGUAD	**40.67**(+10.67)	**23.33**(+5.00)	**35.47**(+8.97)	**26.67**(+5.00)	**15.17**(+3.00)	**23.22**(+4.40)
DAIL-SQL	Vanilla	10.33	4.00	8.43	13.17	8.33	11.72
DAIL-SQL	w/ Original	31.67	18.67	27.77	21.50	14.67	19.45
DAIL-SQL	w/ DISGUAD	**47.33**(+15.66)	**27.67**(+9.00)	**41.43**(+13.66)	**33.17**(+11.67)	**19.67**(+5.00)	**29.12**(+9.67)
DIN-SQL	Vanilla	10.33	5.33	8.83	14.33	8.50	12.58
DIN-SQL	w/ Original	31.00	17.67	27.00	20.67	13.17	18.42
DIN-SQL	w/ DISGUAD	**46.67**(+15.67)	**27.67**(+10.00)	**40.97**(+13.97)	**33.33**(+12.66)	**20.67**(+7.50)	**29.53**(+11.11)

4.2 MAIN RESULTS

Table 1 reports the main results on BIRD-INTERACT-LITE and BIRD-INTERACT-FULL. The upper block compares DISGUAD with Vanilla and the standard BIRD-INTERACT policy. Across both datasets and both LLM backbones, DISGUAD achieves the best PQ-SR, FQ-SR, and Reward. Compared with Original, DISGUAD improves Reward by 6.37/4.73 points with DeepSeek-V4-Flash and by 8.97/4.40 points with GPT-5.6-Luna on LITE/FULL. It also largely outperforms Vanilla, whose Reward remains much lower under both backbones. These results show that ambiguous text-to-SQL cannot be reliably solved from the original query alone, and that selecting SQL-impactful clarification targets is crucial for final SQL quality.

The lower block evaluates whether DISGUAD's clarification signals can benefit static text-to-SQL frameworks. We use DAIL-SQL and DIN-SQL as representative static frameworks and compare Vanilla, w/ Original, and w/ DISGUAD. Adding clarification context improves over Vanilla in all settings, and replacing Original with DISGUAD further improves Reward in every case. The gains range from +1.58 to +3.63 points with DeepSeek-V4-Flash and from +9.67 to +13.97 points with GPT-5.6-Luna. Since the downstream SQL generator is fixed within each comparison, these gains isolate the benefit of DISGUAD's ambiguity-selection strategy rather than a stronger SQL decoder.

Overall, DISGUAD improves the interactive policy itself and provides reusable clarification signals for external SQL generators, supporting its role as a plug-in ambiguity-selection layer for interaction-augmented text-to-SQL.

4.3 CLARIFICATION TARGET ANALYSIS

We analyze whether DISGUAD identifies better clarification targets using the golden ambiguity annotations of benchmark. Clarification questions are automatically matched to golden ambiguity terms by normalized string matching and token overlap. We report five metrics: Cov@3 and Cov@All measure coverage of SQL-impacting critical ambiguity terms within the first three ques-

Table 2: Clarification-target quality of DISGUAD and the original policy against gold annotations on BIRD-INTERACT-FULL. Best results per backbone are bold.

Backbone	Method	Cov@3	Cov@All	Know. Cov@3	Single-target	Repeat ↓
DeepSeek-V4-Flash	Original	48.43	48.62	47.44	38.31	7.38
DeepSeek-V4-Flash	DISGUAD	**66.06**	**82.75**	**54.68**	**44.05**	**6.14**
GPT-5.6-Luna	Original	45.38	45.76	49.61	37.94	11.17
GPT-5.6-Luna	DISGUAD	**67.60**	**85.14**	**54.13**	**45.60**	**5.85**

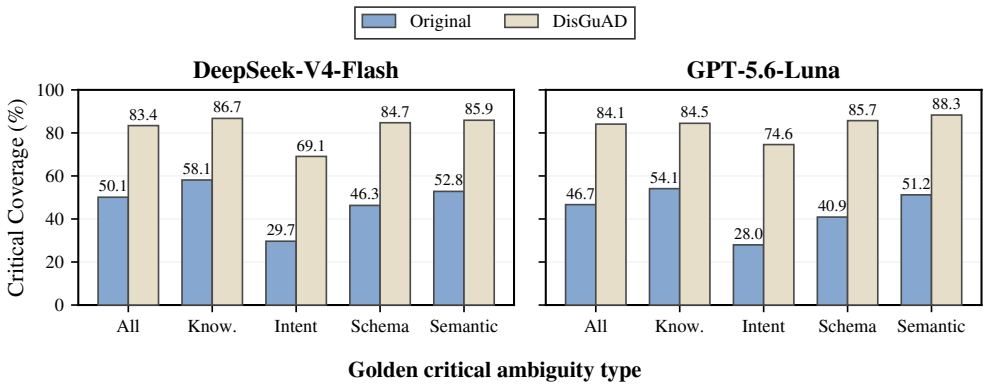


Figure 3: Golden ambiguity coverage on BIRD-INTERACT-FULL. DISGUAD improves SQL-impacting target coverage across both backbones; "Know." denotes knowledge-linking ambiguity.

tions and all questions, respectively; Know. Cov@3 measures early coverage of knowledge-related ambiguity; Single-target measures the fraction of questions that match exactly one golden ambiguity term; and Repeat measures redundant clarification.

Table 2 shows that DISGUAD improves all clarification-target metrics across both backbones. Most notably, Cov@All increases from 48.62% to 82.75% with DeepSeek-V4-Flash and from 45.76% to 85.14% with GPT-5.6-Luna, showing that DISGUAD covers substantially more SQL-impacting ambiguity targets. The gains in Know. Cov@3 and Single-target further indicate that DISGUAD better identifies knowledge-related ambiguities and produces more focused clarification questions, while the lower Repeat rate shows less redundant clarification.

Figure 3 further shows that the improvement is consistent across major ambiguity types, including knowledge-linking, intent, schema-linking, and semantic ambiguity. This supports the central motivation of DISGUAD: SQL disagreement among plausible interpretations helps identify user-facing ambiguity targets that are more likely to affect concrete SQL decisions.

4.4 SENSITIVITY TO CANDIDATE BUDGET

Figure 4 studies the effect of the SQL-candidate budget K on BIRD-INTERACT-FULL. Increasing K from 4 to 20 improves Initial success from 23.00% to 26.00%, Debug success from 25.17% to 28.50%, Follow-up success from 14.83% to 16.83%, and Reward from 22.07 to 25.00, indicating that a larger candidate budget exposes more diverse SQL-level disagreement signals and helps DISGUAD select more useful clarification targets. The gains are not strictly monotonic, as $K = 12$ slightly underperforms $K = 8$ on several metrics, suggesting that candidate quality and diversity also matter. This supports the use of SQL candidates as ambiguity sensors rather than final-answer candidates. We use $K = 8$ in the main experiments as a practical trade-off, since larger budgets can further improve performance but require additional LLM calls for candidate generation.

4.5 ABLATION STUDY

We conduct ablation studies to understand the contribution of the main components in DISGUAD. All ablations are conducted on BIRD-INTERACT-FULL with GPT-5.6-Luna as the backbone. We

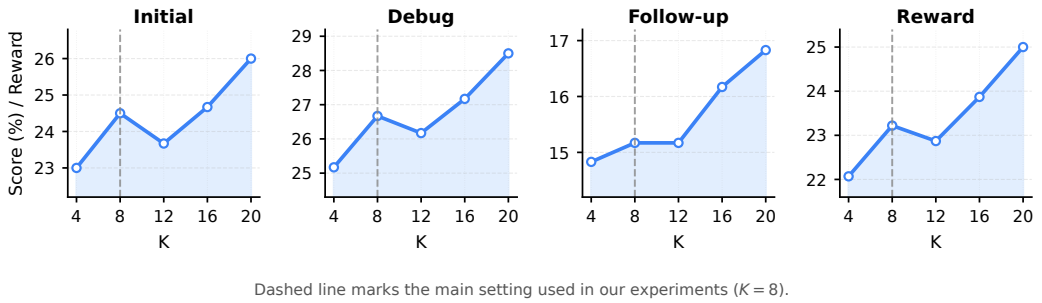


Figure 4: SQL-candidate budget sensitivity on BIRD-INTERACT-FULL. Larger K generally improves all metrics; the dashed line marks the main setting $K = 8$.

consider two groups of ablations. The first group removes LLM-assisted ambiguity modules, including the query-term inventory extractor and the disagreement-to-ambiguity mapper. The second group removes one contrastive SQL interpretation probe family at a time. Following the main results, we report PQ-SR, FQ-SR, and normalized Reward.

Table 3: Ablation study with GPT-5.6-Luna.

Variant	PQ-SR	FQ-SR	Reward
Full DISGUAD	**26.67**	**15.17**	**23.22**
w/o Query-term Inventory	23.33	13.33	20.33
w/o Disagreement Mapping	24.50	14.50	21.50
w/o Subgoal-contrast	26.17	14.83	22.77
w/o Relational-route	25.83	15.00	22.58
w/o Definition-sensitive	25.67	14.67	22.37
w/o Single-axis CF	26.00	15.00	22.70

Table 3 shows that both LLM-assisted ambiguity modules are important. Removing the query-term inventory decreases PQ-SR from 26.67% to 23.33% and Reward from 23.22 to 20.33. This indicates that extracting user-facing ambiguity anchors is necessary for connecting clarification to terms that users can understand and answer. Removing the disagreement mapping module also causes a clear drop, reducing PQ-SR to 24.50% and Reward to 21.50. This confirms that SQL-level disagreement alone is insufficient; DISGUAD needs the mapping step to attribute program differences back to askable ambiguity targets.

The probe-family ablations show that the contrastive SQL interpretation probes provide complementary evidence. Removing subgoal-contrast probes or relational-route probes lowers Reward from 23.22 to 22.77 and 22.58, respectively. This suggests that decomposing the query into SQL subgoals and exploring alternative relational routes both help expose ambiguity that affects the final SQL program. The ablations on definition-sensitive and single-axis counterfactual probes show a similar pattern, reducing Reward to 22.37 and 22.70, respectively. These results indicate that formula-oriented interpretations and controlled counterfactual variations provide useful additional disagreement signals. Overall, the full DISGUAD model achieves the best Reward, supporting the effectiveness of combining user-facing term extraction, SQL-disagreement attribution, and diverse contrastive SQL probes.

5 CONCLUSION

In this paper, we propose DISGUAD, a plug-in clarification policy that identifies SQL-impacting ambiguity through disagreement among plausible SQL interpretations. DISGUAD elicits contrastive queries, compares their canonical components, and attributes recurring differences to user-facing clarification targets while leaving the downstream SQL generator unchanged. Experiments on BIRD-INTERACT demonstrate consistent improvements in PQ-SR, FQ-SR, and Reward across two LLM backbones. Clarifications selected by DISGUAD also improve two widely used text-to-SQL frameworks, demonstrating that its interaction signals transfer across downstream SQL generators. Ablation studies further confirm the contributions of query-term extraction, disagreement mapping, and controlled SQL probes. These results establish SQL disagreement as actionable evidence for deciding what an interactive text-to-SQL agent should ask.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript writing and code development. Specifically, generative AI was used for language editing, improving readability, and drafting or revising parts of the manuscript, as well as for assisting with code writing and debugging. We did not use generative AI tools for research methodology design, hypothesis formulation, or experimental design. All AI-assisted text and code were reviewed and verified by the authors. We take full responsibility for the final content of this work, including all text, code, claims, and results.

REFERENCES

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. Benchmarking and improving text-to-SQL generation under ambiguity. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Jiuhai Chen and Jonas Mueller. Quantifying uncertainty in answers from any language model and enhancing their trustworthiness. In *Association for Computational Linguistics (ACL)*, 2024.

Zhongjun Ding, Yin Lin, Tianjing Zeng, Rong Zhu, Bolin Ding, and Jingren Zhou. Ambisql: Interactive ambiguity detection and resolution for text-to-sql. In *Conference on Management of Data (SIGMOD)*, 2026.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. Speak to your parser: Interactive text-to-SQL with natural language feedback. In *Association for Computational Linguistics (ACL)*, 2020.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 2024.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. In *International Conference on Very Large Data Bases (VLDB)*, 2024.

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

Zijin Hong, Hao Chen, Zheng Yuan, Qinggang Zhang, Luyao Zhuang, Qing Liao, Feiran Huang, Yangqiu Song, and Xiao Huang. Errorllm: Modeling sql errors for text-to-sql refinement. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2026.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. BIRD-INTERACT: Re-imagining text-to-SQL evaluation via lens of dynamic interactions. In *International Conference on Learning Representations (ICLR)*, 2026.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation. In *International Conference on Learning Representations (ICLR)*, 2023.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, Zhaoqing Su, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations (ICLR)*, 2025.

Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. Deepeye-sql: A software-engineering-inspired text-to-sql framework. In *Conference on Management of Data (SIGMOD)*, 2026a.

Boyan Li, Ou Ocean Kun Hei, Yue Yu, and Yuyu Luo. DPC: Training-free text-to-SQL candidate selection via dual-paradigm consistency. In *Association for Computational Linguistics (ACL)*, 2026b.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Conference on Artificial Intelligence (AAAI)*, 2023a.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. In *Conference on Management of Data (SIGMOD)*, 2024.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. Omnisql: Synthesizing high-quality text-to-sql data at scale. In *International Conference on Very Large Data Bases (VLDB)*, 2025.

Jinyang Li, Binyuan Hui, GE QU, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023b.

Yuntao Li, Bei Chen, Qian Liu, Yan Gao, Jian-Guang Lou, Yan Zhang, and Dongmei Zhang. "what do you mean by that?" a parser-independent interactive approach for enhancing text-to-SQL. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Jingyu Liu, Jingquan Peng, Xiaopeng Wu, Xubin Li, Tiezheng Ge, Bo Zheng, and Yong Liu. Do not abstain! identify and solve the uncertainty. In *Association for Computational Linguistics (ACL)*, 2025a.

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

Yujia Liu, Jiayan Lin, Zijin Hong, Zheng Yuan, Shengyuan Chen, Hao Chen, Qinggang Zhang, Xiao Huang, and Feiran Huang. Evaluating llms on conversational text-to-sql under chain ambiguity and intent drift. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

Potsawee Manakul, Adian Liusie, and Mark Gales. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Yang Nan, Pengfei He, Ravi Tandon, and Han Xu. Can multiple responses from an LLM reveal the sources of its uncertainty? In *Findings of Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. In *International Conference on Learning Representations (ICLR)*, 2025.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *Findings of Association for Computational Linguistics (ACL)*, 2024.

Irina Saparina and Mirella Lapata. Ambrosia: A benchmark for parsing ambiguous questions into database queries. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Association for Computational Linguistics (ACL)*, 2020.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *International Conference on Computational Linguistics (COLING)*, 2024.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. In *Conference on Management of Data (SIGMOD)*, 2025.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

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. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019a.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. SParC: Cross-domain semantic parsing in context. In *Association for Computational Linguistics (ACL)*, 2019b.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

A Extended Related Work

Text-to-SQL Generation. Text-to-SQL systems have progressed from specialized semantic parsers to LLM-based generation through improved schema linking, structured decoding, task decomposition, and self-correction (Zhong et al., 2017; Wang et al., 2020; Scholak et al., 2021; Li et al., 2023a; Gao et al., 2024). Recent methods also generate and compare multiple SQL candidates (Pourreza et al., 2025; Li et al., 2026b). Unlike candidate-selection approaches, DisGuAD uses alternative queries to identify what should be clarified rather than to choose the final SQL.

Conversational and Interactive Text-to-SQL. Conversational benchmarks evaluate context-dependent SQL generation over pre-collected dialogue sequences (Yu et al., 2019b;a), whereas interactive parsers actively request structured or free-form user feedback (Yao et al., 2019; Li et al., 2020; Elgohary et al., 2020). BIRD-Interact further evaluates dynamic interaction with users and database environments (Huo et al., 2026). DisGuAD complements these settings by selecting one unresolved, SQL-impacting ambiguity at each interaction turn.

Table 4: Case study on the `polar_2` database. The original interaction asks clarification questions but fails to target the SQL-impacting ambiguities, while DISGUAD asks about ambiguity targets that change filtering, joins, computed expressions, and output labeling.

Ambiguity	Original behavior	DISGUAD behavior
fit for harsh weather	Does not clarify the full readiness rule and applies implicit filters.	Clarifies the Extreme Weather Ready rule: SSF $>$ 0.7, heater not Off, insulation not Poor, and emergency light On or Testing.
structure margin	Uses a threshold without clarifying the formula.	Clarifies it as Structural Safety Factor rounded to two decimals.
all equipment	Interprets the request as returning only equipment satisfying readiness conditions.	Includes each structurally evaluable equipment record and assigns a readiness label.
readiness label	Treats readiness as a filtering criterion.	Treats readiness as a classification output: `Extreme Weather Ready` or `Not Ready`.
Join semantics	Uses inner joins, excluding equipment with missing cabin, lighting, or insulation records.	Uses left joins, so missing auxiliary records do not remove equipment from the output population.

Ambiguity in Text-to-SQL. AmbiQT (Bhaskar et al., 2023) and AMBROSIA (Saparina & Lapata, 2024) associate ambiguous requests with multiple plausible interpretations and SQL queries, while AmbiSQL resolves detected ambiguity through multiple-choice clarification (Ding et al., 2026). Rather than relying only on surface ambiguity or a predefined taxonomy,DISGUAD identifies clarification targets from recurring differences among plausible SQL queries.

Disagreement and Uncertainty in LLM Generation. Consistency-based methods compare sampled LLM generations to estimate semantic uncertainty, detect unsupported content, or diagnose uncertainty sources (Kuhn et al., 2023; Manakul et al., 2023; Chen & Mueller, 2024; Nan et al., 2025; Liu et al., 2025a). These methods operate over free-form responses. DISGUAD instead compares structured SQL components and attributes their disagreement to user-facing clarification targets.

B CASE STUDY

We present a representative case from the `polar_2` database to illustrate how DISGUAD identifies SQL-impacting ambiguity targets. The user asks:

> *Could you identify all equipment that are fit for harsh weather in our polar database? Show me the equipment, heater status, insulation status, emergency light status, the structure margin, and their readiness label.*

This request contains several SQL-impacting ambiguities, including the definition of "fit for harsh weather", the formula for "structure margin", the intended output identifier for "equipment", and whether "readiness label" means filtering only ready equipment or assigning a label to all relevant equipment.

The original interaction policy asks clarification questions, but they do not resolve the ambiguity targets that determine the final SQL semantics. As shown in Table 4, the resulting SQL still makes two incorrect implicit assumptions. First, it treats readiness as a filtering condition, returning only equipment that satisfies the harsh-weather criteria. Second, it uses inner joins to cabin, thermal, and lighting tables, which removes equipment records when auxiliary status information is missing. As a result, the query loses the intended semantics of "all equipment" and "readiness label": the expected behavior is to retain each structurally evaluable equipment record and classify it as either `Extreme Weather Ready` or `Not Ready`.

DISGUAD instead decomposes the uncertainty into user-facing clarification targets. It asks about the definition of "fit for harsh weather", the formula for "structure margin", the meaning of "readiness label", whether "all equipment" should include non-ready equipment, and which equipment identifier should be returned. After receiving these clarifications, DISGUAD generates SQL that uses left joins to preserve the output population and a `CASE` expression to assign the readiness label. This separates inclusion criteria from classification criteria, producing the correct final SQL.

This case highlights that DISGUAD improves clarification by identifying ambiguity terms whose resolution changes concrete SQL decisions, including filtering, join type, computed expressions, and output labeling. The comparison also shows why asking a clarification question is not sufficient by itself: the question must target the ambiguity that actually affects the SQL program.

C PROMPT TEMPLATES FOR DISGUAD

This appendix reports the prompt templates used by DISGUAD to decide clarification questions. The templates correspond to the main components in Section 3: query-term inventory construction, contrastive SQL interpretation probes, mapping SQL disagreement to user-facing ambiguity, and template-based clarification question realization. DISGUAD does not modify the downstream final SQL generation prompt in the main interactive experiments. Instead, it changes the clarification policy by selecting which ambiguity should be asked before final SQL synthesis.

We use braces to denote runtime placeholders, such as {`user_query`}, {`database_schema`}, and {`external_knowledge`}. The prompts are formatted for readability while preserving the logic used in our implementation.

C.1 QUERY-TERM INVENTORY PROMPT

This prompt instantiates the query-term inventory construction described in Section 3.2. The goal is to extract user-facing spans that may later serve as clarification anchors. The extractor does not decide the final question by itself; it only provides a natural-language search space for subsequent SQL-disagreement attribution.

System Prompt

```
You extract user-facing ambiguity terms for interactive text-to-SQL.
Return only JSON. Do not use hidden benchmark labels.
Keep rationales concise.
```

User Prompt Template

```
Database:
{database_name}

Original ambiguous user request:
{user_query}

Database schema:
{database_schema}

Visible external knowledge:
{external_knowledge}

Heuristic term candidates for reference only:
{heuristic_terms}

Extract user-facing terms or phrases that may need clarification
before generating SQL.

Use these coarse ambiguity categories:
- Lexical Ambiguity: a token has multiple meanings in context.
- Syntactic Ambiguity: a phrase can attach to different parts
  of the request.
- Semantic Ambiguity: wording is vague or underspecified.
- Schema Linking Ambiguity: a query term could map to multiple
  schema elements.
```

```
- Query Intent Ambiguity: operation, ranking, filtering,
  aggregation, limit, or DDL/DML intent is unclear.
- Knowledge Linking Ambiguity: a concept may link to an external
  definition or formula, but the link is implicit or unclear.

Important:
- Prefer spans that appear verbatim in the user query.
- Include metric or formula terms such as score, index, rate,
  ratio, percentage, coefficient, or factor.
- Include scope terms such as latest, recent, top, first, or batch.
- Include action terms such as create, update, or manage when
  DDL/DML semantics may matter.
- Include schema-like terms such as ID, status, type, name,
  facility, or telescope when mapping may be ambiguous.
- Do not include generic stopword spans.
- Do not reveal or use benchmark golden ambiguity labels.
- Do not copy the heuristic list mechanically; use it only as
  weak evidence.

Return valid JSON with the following structure:
{
  "terms": [
    {
      "term_id": "t1",
      "span": "...",
      "normalized_term": "...",
      "coarse_category_candidates": ["Semantic Ambiguity"],
      "why_potentially_ambiguous": "...",
      "possible_sql_effects": [
        "select expression",
        "where predicate"
      ],
      "priority_hint": "critical | useful | low"
    }
  ]
}
```

C.2 CONTRASTIVE SQL INTERPRETATION PROBES

This prompt instantiates the contrastive probing stage described in Section 3.3. Given the ambiguous request, schema, external knowledge, and query-term inventory, DISGUAD elicits multiple plausible SQL interpretations. These interpretations are not used for reranking or final SQL selection. Instead, they serve as diagnostic probes for exposing SQL-level disagreement. In the main experiments, we set the candidate budget to $K = 8$.

Probe Families

```
1. Subgoal-contrast probes (SUB)
   Decompose the user request into major SQL subgoals, including
   output columns, tables and joins, filters, metric or formula,
   aggregation grain, ordering or limit, and database operation.
   Generate plausible SQL interpretations that differ because
   one subgoal has a different interpretation.

2. Relational-route probes (REL)
   Explore alternative relational realizations of the same request,
   including table paths, join paths, grouping levels, filter scopes,
   and final projections when multiple schema routes are plausible.

3. Definition-sensitive probes (DEF)
   Focus on metric, score, index, rate, ratio, percentage,
   coefficient, and factor phrases. Generate interpretations that
   explore different formulas, knowledge links, thresholds, or
   numerator/denominator choices.

4. Single-axis counterfactual probes (CF)
   Change one ambiguity axis while keeping other SQL decisions as
   stable as possible. Possible axes include metric formula,
   schema entity, join path, aggregation grain, filter scope,
   ranking or limit, output projection, database operation,
   and null handling.
```

System Prompt

```
You generate plausible PostgreSQL interpretations for ambiguous
text-to-SQL tasks. Return only JSON. Do not execute SQL.
Generate interpretations that are intentionally diverse along
user-facing ambiguity axes.
```

User Prompt Template

```
Database:
{database_name}

Original ambiguous user request:
{user_query}

Database schema:
{database_schema}

Visible external knowledge:
{external_knowledge}

Query-term inventory:
{query_terms}

Clarification memory:
{clarification_memory}

Generate exactly {num_candidates} plausible PostgreSQL interpretations
using the following contrastive probe plan:
{probe_plan}

Important rules:
- Every SQL interpretation must be plausible for the original
  user request.
- Each probe family should explore a different ambiguity source.
- Single-axis counterfactual probes should primarily change exactly
  one ambiguity axis while keeping other SQL decisions stable.
- Prefer schema-valid SQL using visible tables and columns.
- Do not invent hidden benchmark labels or unavailable external
  knowledge.
- Anchor the main assumption of each interpretation to a user-facing
  query span or visible knowledge term.
- Keep SQL concise but complete enough for clause-level comparison.

Return valid JSON with the following structure:
{
  "candidates": [
    {
      "candidate_id": "sub_1",
      "probe_family": "SUB",
      "primary_axis": "metric_formula",
      "anchored_query_span": "...",
      "coarse_category": "Semantic Ambiguity",
      "sql_effect_type_hint": "projection_or_metric_ambiguity",
      "assumption": "...",
      "sql": "SELECT ...",
      "expected_changed_clauses": ["select"],
      "used_tables": ["..."],
      "used_columns": ["..."],
      "used_knowledge_ids": [],
      "unresolved_terms": ["..."],
      "confidence": 0.7
    }
  ]
}
```

C.3 Mapping SQL Disagreement to User-Facing Ambiguity

This prompt corresponds to the disagreement mapping stage in Section 3.5. The input consists of the query-term inventory, probe-generated SQL interpretations, candidate metadata, SQL-difference evidence, and ambiguity-type vote counts. The goal is to attribute SQL-level disagreement to user-facing terms that can be asked about in natural language.

This step treats SQL differences as evidence, not as direct clarification targets. For example, DISGUAD should not ask the user whether a `GROUP BY` clause should change. Instead, it should ask about the natural-language term that explains why plausible SQL interpretations differ in grouping, filtering, projection, join path, ordering, or formula computation.

System Prompt

```
You map ambiguity evidence to user-facing query terms for
interactive text-to-SQL. Return only JSON.
Use query terms as anchors. Do not reveal hidden benchmark labels.
```

User Prompt Template

```
Original ambiguous user request:
{user_query}

Query-term inventory:
{query_terms}

User-query ambiguity categories already detected:
{query_term_categories}

Visible external knowledge:
{external_knowledge}

Clarification memory and interaction history:
{clarification_memory}

SQL interpretation metadata and SQL:
{sql_candidates}

SQL difference evidence:
{sql_diff}

SQL-level ambiguity summary derived from candidate differences:
{sql_ambiguity_summary}

SQL-difference ambiguity type votes:
{prioritized_sql_ambiguities}

Query-first heuristic mappings:
{heuristic_mappings}

Task:
Reason from observed SQL disagreement back to likely user-facing
causes. For each important SQL-level ambiguity, decide which
original-query term, previous-answer term, or visible knowledge term
most likely caused that SQL disagreement.

Use the query-term inventory as the primary anchor. SQL difference
evidence is evidence for ambiguity, not a source of askable terms.

Coarse categories:
- Lexical Ambiguity
- Syntactic Ambiguity
- Semantic Ambiguity
- Schema Linking Ambiguity
- Query Intent Ambiguity
- Knowledge Linking Ambiguity

Rules:
- Prefer terms or spans from the original query.
- You may include a knowledge term from the user's previous answers
  if it appears in memory and needs follow-up.
- You may include a visible external-knowledge metric name only when
  the original query appears to refer to that concept indirectly.
- Do not ask about terms already asked in memory unless the user
  answer introduced a new knowledge term.
- Do not use whole sentences as terms.
- Never use SQL syntax or implementation artifacts as terms:
  SELECT, FROM, WHERE, JOIN, CASE, WHEN, THEN, ELSE, TRUE, FALSE,
  NULL, NULLIF, COALESCE, CAST, JSON_AGG, CTE, alias names, table
  aliases, parentheses, operators, or literal SQL keywords.
- Downrank low-value terms such as "ID", "score", "result", or
  "data" unless candidate evidence shows a real schema or projection
```

```
  conflict and the term appears in the user query.
- If SQL differences are real but cannot be anchored to a query term,
  answer term, or visible knowledge term, mark them as
  unanchored_sql_noise and do not make them askable.
- Prefer terms that explain high-vote SQL ambiguity types and would
  materially change SELECT, WHERE, GROUP BY, JOIN, ORDER BY, LIMIT,
  or DDL/DML behavior.
- A good mapping should explain why this user term caused this
  observed SQL ambiguity, not merely repeat a candidate assumption.
- The best ambiguity should be the one most likely to change
  the final SQL.

Return valid JSON with the following structure:
{
  "ambiguities": [
    {
      "term": "...",
      "source": "original_query | user_answer | visible_knowledge |
     unanchored_sql_noise",
      "coarse_category": "Knowledge Linking Ambiguity",
      "ambiguity_type": "knowledge_linking_ambiguity",
      "sql_effect_type": "projection_or_metric_ambiguity",
      "caused_sql_ambiguity_types": [
        "aggregation_or_metric_ambiguity"
      ],
      "matched_sql_votes": 5,
      "query_span": "...",
      "question_need": "...",
      "candidate_ids": ["c1", "c2"],
      "changed_sql_parts": ["select", "columns"],
      "candidate_interpretations": ["...", "..."],
      "evidence_examples": ["..."],
      "reason": "...",
      "importance": "critical | useful | low",
      "askable": true,
      "requires_user": true,
      "requires_formula": true,
      "confidence": 0.75,
      "priority_score": 0.8
    }
  ]
}
```

C.4 Template-Based Clarification Question Realization

After selecting an ambiguity target, DisGuAD realizes the final clarification question using deterministic templates. We use templates rather than an additional LLM question rewriter in the main experiments. This separates the decision of *which ambiguity to ask about* from the surface form of the question.

Let `term` denote the selected ambiguity target and `sql_effect` denote its predicted SQL effect type.

Main Templates

```
If the ambiguity requires a formula:
  Could you clarify the exact definition or formula for {term}?

If the SQL effect is related to sorting or ranking:
  Could you clarify the intended ordering, ranking, or limit for {term}?

If the SQL effect is related to schema linking:
  Could you clarify which entity or attribute you mean by {term}?

If the SQL effect is related to DDL/DML behavior:
  Could you clarify the intended create, insert, update, or upsert
  behavior for {term}?

If a specific question_need is available:
  Could you clarify {term}? {question_need}

Otherwise:
  Could you clarify what you mean by {term}?
```

Fallback Slot Questions

```
metric_formula:
  Are there any remaining metric, score, or formula definitions
  that should change the SQL?

filter_threshold:
  Are there any remaining filters, thresholds, or inclusion rules
  that should change the SQL?

output_shape:
  Are there any remaining output columns, aliases, or JSON keys
  that should change the SQL?

grouping_grain:
  Should the results use any grouping grain beyond what you
  already specified?

ranking_limit:
  Are there any remaining ordering, ranking, or row-limit
  requirements for the final result?

default:
  Are all SQL-relevant requirements now complete, or is there
  one remaining detail I should use?
```

Interaction Format

```
<s>{clarification_question}</s>
```

D PROMPT SCOPE AND REPRODUCIBILITY NOTES

We report only the prompt templates used by DISGUAD to decide clarification questions. The appendix excludes prompts that are not part of DISGUAD's ambiguity-selection policy, including the original BIRD-INTERACT system prompt, the user simulator prompt, downstream final SQL generation prompts, DAIL-SQL and DIN-SQL prompts, answer-observation prompts, canonical-plan prompts, SQL validation prompts, and SQL repair prompts.

All prompt templates are fixed across the reported experiments. Within each experiment, the same LLM backbone is used for the system model, user simulator, and DISGUAD ambiguity module. Across backbone comparisons, only the LLM backbone changes.