DATABASE-GROUNDED CLARIFICATION THROUGH ORDERED PROGRAM-WORLD EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Interactive Text-to-SQL systems must do more than generate executable queries: they must identify underspecified user intent, determine which information can be obtained from the database environment, and decide what to ask the user under a limited interaction budget. Existing tool-augmented agents largely leave these decisions to free-form reasoning, which can lead to redundant questions, low-value exploration, and unsupported assumptions that silently alter the resulting SQL. We introduce DB-COPE, an inference-time framework that represents uncertain intent as explicit SQL decisions and instantiates grounded alternatives as complete program worlds. Structural validation checks that a claimed interpretive difference changes the SQL; unresolved, result-changing choices are clarified and the answers are carried forward in a canonical state for initial and follow-up queries. On the 600-task BIRD-INTERACT Full set, DB-COPE raises DeepSeek v4-Flash Phase 1 success from 16.17% to 23.50% and end-to-end Phase 2 success from 7.67% to 13.17%. DB-COPE likewise improves both phases by nine percentage points with GPT-5.6 Terra. On PRACTIQ, it reaches 82.48% nine-way accuracy, showing that the same representation transfers to static user-request classification. This result suggests that explicit SQL-decision representations can support program-grounded clarification by distinguishing user-request status.

1 INTRODUCTION

Text-to-SQL systems are usually evaluated after a request has already specified enough information to determine a target query. A database assistant often has to act earlier. A user may name a business quantity without defining its formula, omit the population to which a filter applies, or leave the desired result grain unclear. The system must decide whether the missing information is a database fact that it can retrieve or a preference that only the user can supply. In a multi-turn task, it must make this decision without exhausting the interaction budget and without losing constraints confirmed in earlier turns.

Context-dependent benchmarks test whether a parser can use a given dialogue history, and interactive parsers use feedback to repair uncertain or incorrect predictions (Yu et al., 2019b;a; Yao et al., 2019; Elgohary et al., 2020). Ambiguity-aware datasets instead ask whether a request is answerable or has multiple interpretations (Dong et al., 2025; Saparina & Lapata, 2024). These settings establish the value of context, feedback, and explicit ambiguity, but they largely fix either the interaction history or the source of supervision.

BIRD-INTERACT exposes the choice of information source itself (Huo et al., 2026). An agent can inspect schema and column metadata, retrieve domain knowledge, execute diagnostic SQL, ask the user, or submit a query, with all actions drawing from a shared budget. A free-form ReAct agent must therefore decide both what is unknown and how to resolve it. This policy can miss a consequential assumption shared by all candidate queries. Two queries may differ in their join paths, for example, while both silently use the same unsupported metric formula. Generating more candidates or allowing more tool calls does not reveal such a shared default.

We ask whether clarification can instead be tied to explicit SQL decisions. The useful comparison is not between arbitrary candidate strings, but between grounded interpretations whose differences change the resulting program. Schema evidence may settle a join, and an external definition may settle a named metric; neither can determine a user's preferred threshold or grouping grain. Choices

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.

in the latter category should remain unresolved until they are clarified rather than being filled by a plausible default.

We propose DB-COPE (**D**ata**b**ase-grounded **C**larification through **O**rdered **P**rogram-world **E**xploration). It first records the status of eight coarse SQL decisions, including formula, scope, grouping, joins, aggregation, sorting, and output structure. From the available evidence it constructs zero, one, or two complete Grounded SQL Worlds. Zero worlds indicate that a critical definition is still missing; one world indicates a supported implementation; two worlds expose grounded alternatives with different program consequences. DB-COPE parses the world queries and rejects a declared contrast that is not visible in the corresponding SQL structure. It asks about the highest-impact remaining decision, writes the answer back to the affected component state, and composes the final SQL from the resolved state. For a follow-up request, a smaller Phase 2 record separates inherited constraints from requested changes.

Across BIRD-INTERACT evaluations, DB-COPE improves both initial and follow-up success with DeepSeek v4-Flash and GPT-5.6 Terra . The gains are especially pronounced on data-management tasks, where an incorrect interpretation can affect both the submitted program and the resulting database state. The PRACTIQ diagnostic complements these end-to-end results by testing transfer to static request-state classification. DB-COPE reaches 82.48% nine-way accuracy, within four percentage points of definition-guided zero-shot and 3-shot/class. Category-level results are stronger on structurally grounded requests than on semantic ambiguity.

Contributions.

- **Decision-level clarification.** We formulate interactive clarification as resolving result-changing SQL decisions against database evidence, separating choices that the environment can determine from those that require user input.
- **Grounded program-world inference.** DB-COPE externalizes unresolved decisions through complete SQL worlds, verifies that competing interpretations induce structural program differences, and carries confirmed choices across initial and follow-up requests.
- **End-to-end and diagnostic evaluation.** We test the method in dynamic two-phase Text-to-SQL and in request-state recognition, and analyze the component contributions to task success and interaction behavior.

2 RELATED WORK

Context and feedback in Text-to-SQL. SParC, CoSQL, and CHASE evaluate SQL generation from dialogue histories that contain ellipsis, coreference, and changing constraints (Yu et al., 2019b;a; Guo et al., 2021). Their interaction sequences are collected before evaluation, so the parser uses a supplied history rather than deciding when to seek information. MISP gives the parser a more active role: it asks for feedback at uncertain parser states and incorporates the answer through a world model (Yao et al., 2019). SPLASH and STEPS instead repair an erroneous query through free-form feedback or editable natural-language explanations (Elgohary et al., 2020; Tian et al., 2023). These methods show that feedback can be attached to program decisions. The feedback channel or dialogue history is nevertheless specified in advance; choosing among schema inspection, domain knowledge, database execution, and user clarification is not part of the prediction problem.

Ambiguity and request status. TriageSQL and *Know What I Don't Know* classify requests that are ambiguous, unanswerable, or otherwise unsuitable for direct SQL generation (Zhang et al., 2020; Wang et al., 2023). PRACTIQ extends this setting with conversational examples and several ambiguity and unanswerability categories (Dong et al., 2025). Such classification determines whether a system should answer, clarify, or abstain, but it does not identify which SQL decision a clarification should resolve.

Other work represents the alternatives themselves. AMBROSIA covers scope, attachment, and vagueness that remain after the database context is known (Saparina & Lapata, 2024). Disambiguate First, Parse Later generates natural-language interpretations before logical-form prediction, while AmbiSQL and Sphinteract place user clarification inside a Text-to-SQL pipeline (Saparina & Lapata, 2025; Ding et al., 2026; Zhao et al., 2024). Across these approaches, ambiguity may

be represented as a request label, a natural-language interpretation, or an alternative query. These representations differ in whether they expose the program consequences of a semantic choice and whether assumptions shared by all alternatives remain visible.

Agents in dynamic database environments. ReAct-style agents interleave language-model reasoning with tool calls (Yao et al., 2023). BIRD-INTERACT applies this pattern to an executable database environment: an agent can inspect metadata, retrieve external knowledge, run diagnostic queries, ask a user, and submit SQL under a common action budget (Huo et al., 2026). The benchmark also includes follow-up requests whose evaluation depends on the accepted first-phase program and the resulting database state.

The benchmark specifies the available evidence and action costs, but leaves the agent to decide what to inspect, when to ask, and how to retain a partial answer. DB-COPE works with the same information sources and instead makes the unresolved SQL decisions and retained constraints explicit across phases. The resulting comparison concerns how a fixed tool set is organized under a shared budget and evolving database state, rather than access to additional evidence.

3 PROBLEM SETTING

Interactive Task. An agent receives a natural-language request x and interacts with a relational database. We write $\mathcal{S}$ for the schema, $\mathcal{D}$ for the current database instance, $\mathcal{M}$ for column-level metadata, $\mathcal{K}$ for external domain knowledge, and $\mathcal{U}$ for the user. Together these sources form the environment $\mathcal{E} = (\mathcal{S}, \mathcal{D}, \mathcal{M}, \mathcal{K}, \mathcal{U})$. At step t, the agent selects an action from the request and the preceding action–observation history H_t. It may inspect schema or metadata, retrieve knowledge, execute a diagnostic query, ask the user, or submit SQL.

All information-acquisition and submission actions draw from the same initial budget B_0. If $\tilde{c}_t$ is the charge recorded for action a_t, a valid trajectory must satisfy

$$\sum_{t=1}^{T} \tilde{c}_t \le B_0, \qquad \mathrm{Exec}(\mathcal{D}, \hat{q}) \equiv \mathrm{Exec}(\mathcal{D}, q^\star). \tag{1}$$

Here $q^\star$ is the unobserved target program and $\hat{q}$ is the submitted program. Equivalence is determined by the benchmark's functional evaluator, not by SQL string identity. The admissible programs include both read-only queries and data-management statements; the latter may change the database state observed by a later request. Appendix B.5 gives the action prices and operational gates used in our implementation.

Intent decisions. A request may identify the relevant tables and columns yet still leave the query underdetermined. We organize such choices into eight coarse decision components:

$$\begin{aligned}\mathcal{C} = \{&\texttt{metric_formula}, \texttt{threshold}, \texttt{filter_scope}, \texttt{group_by},\\ &\texttt{join_path}, \texttt{aggregation}, \texttt{sorting}, \texttt{projection}\}.\end{aligned} \tag{2}$$

The inventory covers the quantity being computed, its cutoff and population, the result grain and relations, aggregation, ordering, and output fields. Not every component applies to every request, and the choices are not independent: changing the grouping grain may also require a different join, aggregation, and projection. The inventory is therefore a practical checklist rather than an exhaustive taxonomy of SQL. Appendix B.1 gives the component-to-SQL mapping.

For an applicable component $c \in \mathcal{C}$, let z_c denote the value established by the request, environment evidence, or a user answer. A component remains unresolved when the available evidence does not determine a single choice and different choices would change the query result or the database state. The agent must determine which choices can be settled from the environment and ask the user about the rest before constructing $\hat{q}$; silently choosing among unresolved alternatives amounts to an unsupported default.

Two-phase interaction. We use the a-Interact protocol of BIRD-INTERACT. Phase 1 begins with an initial request. Phase 2 is released only if the Phase 1 submission passes the functional evaluator. The accepted program is then executed before the follow-up request is issued, so a data-management statement may change the database observed in Phase 2. The interaction history and unused budget also carry over.

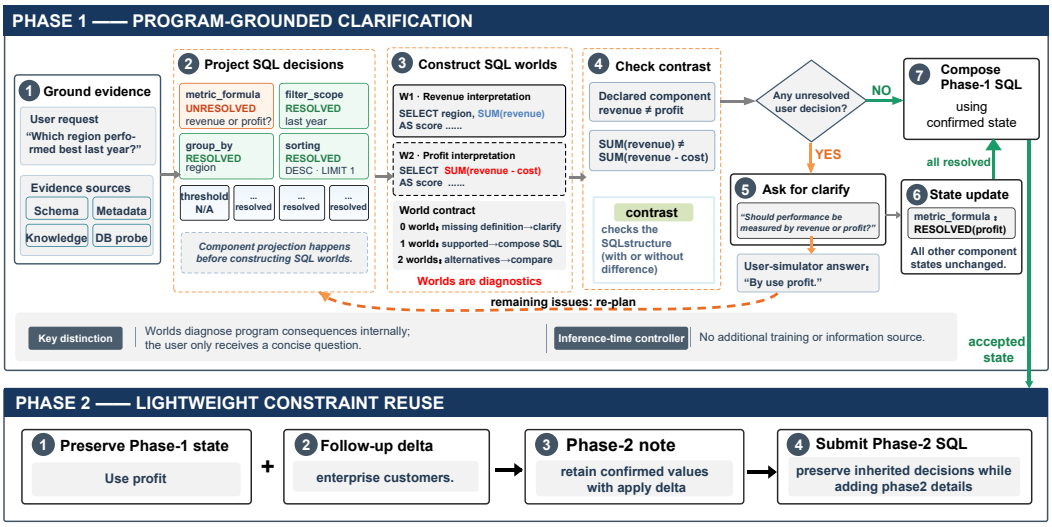

Figure 1: Overview of DB-COPE. In Phase 1, the agent projects grounded evidence onto explicit SQL decision components, constructs zero, one, or two Grounded SQL Worlds, and validates declared contrasts against SQL structure. Unresolved result-changing decisions trigger business-language clarification and sparse state updates. After a successful Phase 1, Phase 2 retains confirmed constraints and applies only the follow-up changes.

A follow-up may retain, replace, or add constraints. Choices confirmed in Phase 1 remain in force unless the follow-up explicitly changes them. Let $\mathcal{C}_{\text{keep}}$ denote the applicable decision components that the follow-up does not change. Their Phase 1 values carry into Phase 2; other components may be revised, newly introduced, or made irrelevant. End-to-end success therefore requires two correct programs under a shared budget while preserving the constraints that the follow-up leaves unchanged.

4 DB-COPE

Figure 1 presents the overall workflow of DB-COPE. DB-COPE adds structured internal planning to the original environment without changing the model, tools, or available evidence. Phase 1 resolves explicit decision components before submission, while Phase 2 separates inherited constraints from requested changes.

4.1 Component State and Evidence

A comparison between candidate programs cannot expose an unsupported choice that all candidates share. DB-COPE therefore projects the request and the available evidence onto every component before constructing SQL worlds. This keeps a decision unresolved even when preliminary candidates happen to use the same plausible default.

For each component $c \in \mathcal{C}$, DB-COPE stores

$$s_t(c) = \left\langle \rho_t(c), u_t(c), v_t^{(1)}(c), v_t^{(2)}(c), \bar{v}_t(c), E_t(c), r_t(c) \right\rangle . \tag{3}$$

The status $\rho_t(c)$ records whether the component applies and whether a result-relevant choice remains unresolved. The field $u_t(c)$ states what is still unknown, while $v_t^{(i)}(c)$ records the choice instantiated in world W_i. When the component is resolved, $\bar{v}_t(c)$ is its canonical value and instantiates the z_c defined in Section 3. $E_t(c)$ records the supporting evidence, and $r_t(c)$ describes the possible effect on execution. A partial answer narrows $u_t(c)$ without marking the component as resolved.

Resolution is source-sensitive. The request and user answers can establish intended business choices; schema and column metadata establish entities, attributes, and relationships; external

knowledge can supply a named formula or business rule; and database execution can verify a database fact. Evidence from the database cannot by itself determine a user preference, so a result-changing choice remains unresolved until it has support from the appropriate source.

4.2 GROUNDED SQL WORLDS

A semantic alternative is useful for clarification only when its program consequences can be inspected. DB-COPE therefore represents each candidate interpretation with complete SQL rather than a natural-language paraphrase. Let $\mathcal{Q}_{\mathrm{ro}}(\mathcal{S})$ contain complete, parseable, read-only PostgreSQL queries over $\mathcal{S}$. A Grounded SQL World is $W_i = \langle I_i, q_i, E_i, A_i \rangle$, where $q_i \in \mathcal{Q}_{\mathrm{ro}}(\mathcal{S})$ and I_i is the business interpretation realized by q_i. The evidence E_i supports the schema and business choices used in the world, while A_i records any result-changing intent assumptions that the available evidence does not yet license. Keeping these fields separate prevents a coherent implementation from being mistaken for a confirmed interpretation. World queries are diagnostic objects and need not be executed. For a data-management request, a world may compute candidate values or identify affected rows even though the final submitted program can modify the database.

DB-COPE permits only three world counts:

$$|\mathcal{W}_t| = \begin{cases} 0, & \text{a required definition prevents a complete query,} \\ 1, & \text{the evidence supports one complete implementation,} \\ 2, & \text{two coherent implementations have different consequences.} \end{cases} \tag{4}$$

Zero worlds do not by themselves make the request unanswerable. They indicate that completing a query would require inventing a missing definition, which is instead requested from the user. One world is allowed only when every result-relevant component is resolved and A_1 is empty.

With two worlds, each must be a coherent joint interpretation. The second changes every unresolved component for which the evidence supports an alternative value. This maximum-contrast rule exposes several decisions with one pair of programs instead of enumerating their Cartesian product; the formal constraint appears in Appendix B.2. Existing evidence may select one world only when no component remains unresolved and the selected A_i is empty. Otherwise neither world is selected, and their differing components become candidates for clarification. The worlds do not restrict the final answer: a later user response may support a combination of component choices drawn from both.

4.3 STRUCTURAL VALIDATION AND CLARIFICATION

DB-COPE parses every available world query with SQLGlot in the PostgreSQL dialect. For a two-world plan, it compares a component-specific AST fingerprint for each declared contrast. If the two fingerprints are identical, the planning record is rejected; a declared ASC/DESC contrast must likewise appear in the parsed order directions. This check establishes that a stated difference reaches the program. It does not establish that either interpretation is grounded or correct, nor does it compare the eventual submitted SQL with the resolved component state. Appendix B.2 lists the fingerprints and validator boundaries.

Let $\mathcal{C}_t^{\mathrm{unres}} = \{c \in \mathcal{C} : \rho_t(c) = \texttt{unresolved}\}$. If this set is empty and a supported world has been selected, the planning stage releases the agent to proceed with the final SQL. Otherwise, among the components that the request and available non-user evidence cannot settle, DB-COPE uses the recorded result effect $r_t(c)$ to choose a clarification target. This field gives a qualitative priority rather than a learned score. For a zero-world case, the question requests the missing definition. Questions use business language and omit SQL, schema identifiers, component names, and world labels.

4.4 ANSWER UPDATE AND SQL COMPOSITION

Suppose answer y_t addresses the previously unresolved components $\mathcal{C}_t^{\mathrm{ans}} \subseteq \mathcal{C}_t^{\mathrm{unres}}$. DB-COPE applies the sparse update

$$s_{t+1}(c) = \begin{cases} \mathrm{Update}_c(s_t(c), y_t), & c \in \mathcal{C}_t^{\mathrm{ans}}, \\ s_t(c), & \text{otherwise.} \end{cases} \tag{5}$$

A resolved update supplies a concrete canonical value and cites the answer as evidence. An incomplete answer narrows the issue but leaves it unresolved. Components not addressed by the answer are copied unchanged.

After every Phase 1 user response, the next internal action must record a nonempty valid update. Further exploration, another question, and submission remain blocked until this update succeeds; submission also remains blocked while any component remains unresolved. These gates prevent a user answer from remaining only in dialogue history or being overwritten by an unrelated revision.

Once no result-relevant component remains unresolved, the canonical values form $\hat{z} = (\bar{v}_T(c))_{c:\rho_T(c)=\texttt{resolved}}$. If planning required no clarification, these values are projected from the selected world. After clarification, they include the sparse updates and, in a two-world case, may combine component choices drawn from both original worlds. For a read-only request, the selected world query may serve as the final SQL. For a data-management request, the world remains diagnostic, and the model generates the modifying statement from the request, acquired evidence, and $\hat{z}$. Before submission, the model is instructed to check the program against the confirmed component values and requested output shape. After a failed submission, its next query should retain independently supported decisions unless evaluator feedback or the self-check implicates one of them. These self-check and retry rules are model instructions rather than deterministic SQL-level constraints. Appendix B.3 gives the update, submission, and retry rules.

Phase-2 constraint reuse. A follow-up can fail by dropping a confirmed Phase 1 constraint or by retaining one that the user has explicitly changed. Phase 2 therefore inherits the accepted Phase 1 history, component values, resulting database state, and unused budget, but uses a lighter planning record instead of rebuilding SQL worlds.

Before either asking the user or submitting SQL in Phase 2, DB-COPE creates a single record of the reusable tables and Phase 1 decisions, the changes stated in the follow-up, any newly introduced ambiguity, and a proposed SQL sketch. At most one clarification question may follow this record. For every unchanged component $c \in \mathcal{C}_{\text{keep}}$ defined in Section 3, the intended reuse condition is $\hat{z}_c^{(2)} = \hat{z}_c^{(1)}$. This is a reuse policy rather than a deterministic SQL-level invariant. Components outside this set may be replaced, introduced, or made irrelevant by the follow-up. Unlike Phase 1, Phase 2 does not construct new worlds, run the AST contrast validator, or apply a sparse component-state update after clarification. The controller enforces the presence and order of the planning record, but does not verify its semantic content against the submitted SQL. Appendix B.3 gives the record and clarification rules; action limits and budget reservation appear in Appendix B.5.

5 EXPERIMENTS

We evaluate DB-COPE in two complementary settings. The primary BIRD-INTERACT evaluation measures end-to-end interactive Text-to-SQL performance, while the PRACTIQ diagnostic examines how the same representation transfers to static request-state classification. This subsection describes the interactive setup; Section 5.4 specifies the PRACTIQ protocol.

5.1 INTERACTIVE EVALUATION SETUP

We evaluate all 600 BIRD-INTERACT Full tasks under a-Interact (Huo et al., 2026): 410 business-intelligence (BI) and 190 data-management (DM) tasks, each with a Phase 2 follow-up. We compare Original ADK/ReAct with DB-COPE using DeepSeek v4-Flash and GPT-5.6 Terra as system-agent backbones; unless otherwise specified, Gemini 2.5 Flash serves as the User Simulator. Within each backbone, the paired methods share the tasks, simulator, database states, environment tools, prices, budgets, and functional evaluator. DB-COPE adds only zero-cost internal records that expose no new environmental information. Appendix A gives the complete decoding, budget, and runtime settings.

We report Phase 1 success (P1-SR), end-to-end Phase 2 success (P2-SR), and mean task reward (NR), where Phase 2 success requires Phase 1 success and each task receives $0.7s_i^{(1)} + 0.3s_i^{(2)}$. Tables report $100 \times \text{NR}$ as Reward*; Appendix A.3 gives the full definitions.

Table 1: Performance on the 600-task BIRD-INTERACT Full set under a-Interact. Priority Question and Follow-Up denote Phase 1 and end-to-end Phase 2 success rates, respectively. Reward* = $100 \times$ NR. BI and DM denote Business Intelligence and Data Management queries. Green arrows indicate the absolute gains of DB-COPE over the corresponding base model.

Model / Method	Priority Question (Success Rate %) ↑			Follow-Up (Success Rate %) ↑			Reward* ↑
	BI	DM	Overall	BI	DM	Overall	
a-Interact Text-to-SQL							
Qwen-3-Coder-480B	8.05	24.74	13.33	3.90	4.74	4.17	10.58
O3-Mini	12.20	36.32	19.83	5.85	14.21	8.50	16.43
Gemini-2.5-Pro	10.49	41.58	20.33	5.85	20.00	10.33	17.33
Claude-Sonnet-3.7	11.46	41.58	21.00	5.61	16.84	9.17	17.45
Claude-Sonnet-4	15.85	53.68	27.83	8.05	22.63	12.67	23.28
GPT-5	15.61	58.42	29.17	10.98	30.00	17.00	25.52
GPT-5.6 Terra	15.38	48.57	27.00	9.23	31.43	17.00	24.00
GPT-5.6 Terra (DB-COPE)	**23.59** ↑8.21	**59.05** ↑10.48	**36.00** ↑9.00	**16.41** ↑7.18	**43.81** ↑12.38	**26.00** ↑9.00	**33.00** ↑9.00

Table 2: User-simulator robustness on the 600-task BIRD-INTERACT Full set with DeepSeek v4-Flash as the system agent. Success rates are percentages; Reward* = $100 \times$ NR.

User Simulator	Method	P1-SR	P2-SR	Reward*
Gemini 2.5 Flash	Original ADK/ReAct	16.17	7.67	13.62
Gemini 2.5 Flash	DB-COPE	**23.50**	**13.17**	**20.40**
DeepSeek v4-Flash	Original ADK/ReAct	18.17	8.67	15.32
DeepSeek v4-Flash	DB-COPE	**22.00**	**13.00**	**19.30**

5.2 MAIN RESULTS ON BIRD-INTERACT

Main findings. Table 1 shows the clearest gains with the stronger GPT-5.6 Terra backbone. Adding DB-COPE improves both Phase 1 and end-to-end Phase 2 success by nine percentage points, yielding a Reward* of 33.00 rather than 24.00. The resulting row has the highest overall success rates and Reward* in the table. The published systems provide useful context for this position, but the controlled evidence for the method comes from the within-Terra comparison, where the backbone and interaction environment are held fixed.

The Terra gains also extend across both BI and data-management tasks rather than arising from one task family. They are larger on data management at both stages: in Phase 2, success rises from 31.43% to 43.81%, compared with 9.23% to 16.41% on BI queries. Because Phase 2 credit is awarded only after a correct initial submission, this increase concerns complete two-phase trajectories in which the follow-up changes the request while earlier constraints may remain in force. The larger data-management gap is compatible with clarification being more consequential when an interpretation affects both the submitted program and the resulting database state.

Under the Gemini 2.5 Flash User Simulator, the paired DeepSeek task-level outcomes also favor DB-COPE in Phase 1 and Phase 2 (two-sided exact McNemar $p = 2.15 \times 10^{-6}$ and $p = 2.72 \times 10^{-5}$, respectively).

Table 2 shows that DB-COPE consistently outperforms Original ADK/ReAct on P1-SR, P2-SR, and Reward*. Thus, the main conclusion is robust to the choice of user-simulator model. Changing the simulator alters the realization of clarification responses while leaving the system agent, tasks, budget, and evaluator fixed. The unchanged method ordering therefore suggests that DB-COPE's benefit is not tied to one simulator's response style. Its persistence through Phase 2 further indicates that the effect carries into follow-up composition, although simulator robustness does not replace evaluation with human users.

Trajectory-level diagnosis. Paired DeepSeek trajectories show that DB-COPE recovers substantially more Original failures than it introduces and more often completes planning and SQL submission. Appendix B.4 reports the transition counts and representative successful and failed trajectories.

Table 3: Component-ablation results on the 600-task BIRD-INTERACT Full set with DeepSeek v4-Flash and the Gemini 2.5 Flash User Simulator. NR is on a $[0, 1]$ scale, success rates are percentages, and coins and asks are task averages.

Variant	NR	P1-SR	P2-SR	Coins	Asks
Full DB-COPE	**0.2040**	**23.50**	**13.17**	**15.36**	1.04
w/o Explicit Component Projection	0.1933	22.33	12.33	14.12	1.17
w/o Executable SQL Worlds	0.1883	21.83	11.83	14.13	1.29
w/o Sparse Component-State Updates	0.2007	23.17	12.83	14.73	1.37

5.3 MECHANISM ANALYSIS AND ABLATIONS

Ablation protocol. The ablation study isolates component projection, executable SQL worlds, and persistent state updates. Each variant is evaluated on the same BIRD-INTERACT Full tasks as the DeepSeek main comparison, with the Gemini 2.5 Flash User Simulator and all non-ablated settings held fixed. Table 3 reports the completed runs.

Variants. *Without explicit component projection* derives clarification targets from differences between generated worlds instead of first constructing the complete eight-component inventory. It tests the role of discovering unsupported decisions that candidate programs may share. *Without executable SQL worlds* retains explicit component states but represents alternatives in natural language, removing the complete SQL instantiations used to check program-level consequences. *Without sparse component-state updates* retains user answers in the dialogue history but removes canonical state merging, the mandatory update-before-action gate, and the unresolved-state submission gate.

Ablation findings. Removing either explicit component projection or executable SQL worlds consistently degrades task success, with the larger drop caused by removing SQL worlds. Removing sparse updates mainly increases repeated clarification, supporting their role in interaction and state management rather than direct accuracy gains.

Budget robustness. DB-COPE remains superior across all tested budget settings and is less sensitive to restrictive budgets; full results are reported in Appendix A.4.

5.4 REQUEST-STATE DIAGNOSIS ON PRACTIQ

Task and protocol. PRACTIQ evaluates whether a request is answerable, ambiguous, or unanswerable through a nine-category taxonomy (Dong et al., 2025). We use a fixed 2,836-example sample generated with the released workflow and sampled with seed 42 to match the category counts of the original evaluation. The results therefore concern this regenerated set rather than the original test set. Under the `lexicalOnly` condition, every method receives the same question, schema, and retrieved example values; gold labels, reference SQL, later dialogue turns, and database execution are unavailable.

Comparison conditions. We compare three methods using DeepSeek v4-Flash. Definition-guided zero-shot directly predicts one of the nine labels from their detailed definitions. 3-shot/class adds three demonstrations per label (27 in total) in a fixed interleaved order. DB-COPE receives no label demonstrations. It first constructs a planning state and zero, one, or two SQL worlds, after which a deterministic adapter maps the structured output to the same taxonomy. Thus, the first two methods are specialized classifiers, whereas DB-COPE is evaluated on how well its interaction-oriented representation transfers to static request-state classification.

Metrics. We report accuracy and macro-F1 for the original nine labels and for their three-way collapse into Answerable, Ambiguous, and Unanswerable. We also report prompt, completion, and total token consumption. All methods produce valid predictions for all 2,836 examples.

Results. As shown in Table 4, definition-guided zero-shot already reaches 85.44% nine-way accuracy, reflecting the close alignment between its explicit label definitions and the benchmark taxonomy. Adding demonstrations yields a further gain of 1.02 points in nine-way accuracy and 1.97 points in three-way accuracy, making 3-shot/class the strongest classifier.

Table 4: Classification performance (%) and token use (millions) on the 2,836-example PRACTIQ `lexicalOnly` set with DeepSeek v4-Flash. Token counts include all model responses. Bold denotes the best classification result.

	9-way		3-way		Tokens (M)		
Method	Acc. ↑	Macro-F1 ↑	Acc. ↑	Macro-F1 ↑	Prompt	Completion	Total
zero-shot	85.44	80.54	87.45	86.90	3.61	3.55	7.16
3-shot/class	**86.46**	**81.35**	**89.42**	**88.76**	54.00	1.77	55.77
DB-COPE	82.48	76.88	84.91	82.50	9.44	14.79	24.23

DB-COPE reaches 82.48% nine-way accuracy, 2.96 points below zero-shot and 3.98 points below 3-shot/class. Although it does not surpass the specialized classifiers, it remains within four points of both while producing the structured states and SQL worlds used for interaction rather than directly optimizing label prediction.

Category-level results in Table 10 show that this transfer is strongest when the request state can be grounded in explicit schema or value evidence. DB-COPE achieves 94.37% recall on Unsupported Join and 90.00% on Nonexistent Filter Value, but only 47.52% on Ambiguous Filter Criteria. This pattern suggests that SQL-world reasoning is more effective for structural grounding than for descriptive or implicit ambiguity.

Token use. The three methods have different resource profiles. Definition-guided zero-shot uses 7.16M tokens in total. The 27 demonstrations make 3-shot/class the most prompt-intensive method, with 54.00M prompt tokens and 55.77M total. DB-COPE uses 24.23M tokens in total, including 14.79M completion tokens for its structured planning output. Its total usage is therefore higher than zero-shot but lower than 3-shot/class.

6 LIMITATIONS

Outcome transitions. Paired analysis shows that DB-COPE recovers more Original failures than it introduces and more often carries interaction through to SQL submission across ambiguity types. This does not imply that every task improves; transition counts and representative cases appear in Appendix B.4.

Residual failure modes. Most residual errors occur after SQL construction. A repair may drop a confirmed predicate or aggregation rule even when the canonical state remains correct; alternatively, it may preserve the clarification while retaining an incorrect formula, join, grain, or projection. Because the evaluator provides no component-level diagnosis for an executable but incorrect submission, a retry may alter a settled constraint without repairing the underlying error.

Evaluation scope. The interactive evidence is limited to two system-agent backbones on BIRD-INTERACT Full. The primary experiments use Gemini 2.5 Flash as the User Simulator, while the robustness check repeats the DeepSeek v4-Flash comparison using DeepSeek v4-Flash as the User Simulator. These model simulators provide controlled responses but cannot capture the full variation of human clarification behavior.

7 CONCLUSION

Interactive Text-to-SQL fails not only when an agent overlooks ambiguity, but also when clarification is disconnected from the SQL eventually executed. We introduced DB-COPE, which represents result-changing interpretations as grounded program worlds and carries resolved choices into subsequent planning. It improves both interaction phases across the evaluated backbones, while the PRACTIQ diagnostic shows transfer to request-state classification. Remaining errors concentrate in translating resolved decisions into SQL, especially during repair, indicating that reliable interaction must keep clarified choices binding as the final program is composed and revised.

AI USE STATEMENT

Generative AI tools assisted language editing, manuscript organization, LaTeX, and the overview figure. The authors checked the experimental code, data, and manuscript for correctness and consistency and take responsibility for the final work.

REPRODUCIBILITY STATEMENT

Details needed to reproduce our evaluations are given in Sections 3 and 5 and Appendix A: the benchmark protocol, controlled comparison, model and decoding settings, budgets and runtime limits, reporting metrics, ablations, and PRACTIQ sampling. Method-level records, validators, and state-transition rules are specified in Appendix B, with additional PRACTIQ category results in Appendix C.

REFERENCES

Zhongjun Ding, Yin Lin, Tianjing Zeng, Rong Zhu, Bolin Ding, and Jingren Zhou. AmbiSQL: Interactive ambiguity detection and resolution for text-to-SQL. In *Companion of the International Conference on Management of Data*, pp. 26–29, 2026. doi: 10.1145/3788853.3801581. URL https://doi.org/10.1145/3788853.3801581.

Mingwen Dong, Nischal Ashok Kumar, Yiqun Hu, Anuj Chauhan, Chung-Wei Hang, Shuaichen Chang, Lin Pan, Wuwei Lan, Henghui Zhu, Jiarong Jiang, Patrick Ng, and Zhiguo Wang. 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)*, pp. 255–273, 2025. doi: 10.18653/v1/2025.naacl-long.13. URL https://aclanthology.org/2025.naacl-long.13/.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. Speak to your parser: Interactive text-to-SQL with natural language feedback. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2065–2077, 2020. doi: 10.18653/v1/2020.acl-main.187. URL https://aclanthology.org/2020.acl-main.187/.

Jiaqi Guo, Ziliang Si, Yu Wang, Qian Liu, Ming Fan, Jian-Guang Lou, Zijiang Yang, and Ting Liu. CHASE: A large-scale and pragmatic chinese dataset for cross-database context-dependent text-to-SQL. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 2316–2331, 2021. doi: 10.18653/v1/2021.acl-long.180. URL https://aclanthology.org/2021.acl-long.180/.

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 for large language models via lens of dynamic interactions. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.05318.

Irina Saparina and Mirella Lapata. AMBROSIA: A benchmark for parsing ambiguous questions into database queries. In *Advances in Neural Information Processing Systems*, volume 37, pp. 90600–90628, 2024. doi: 10.52202/079017-2876. URL https://arxiv.org/abs/2406.19073.

Irina Saparina and Mirella Lapata. Disambiguate first, parse later: Generating interpretations for ambiguity resolution in semantic parsing. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16825–16839, 2025. doi: 10.18653/v1/2025.findings-acl.863. URL https://aclanthology.org/2025.findings-acl.863/.

Yuan Tian, Zheng Zhang, Zheng Ning, Toby Jia-Jun Li, Jonathan K. Kummerfeld, and Tianyi Zhang. Interactive text-to-SQL generation via editable step-by-step explanations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 16149–

16166, 2023. doi: 10.18653/v1/2023.emnlp-main.1004. URL https://aclanthology.org/2023.emnlp-main.1004/.

Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. 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*, pp. 5701–5714, 2023. doi: 10.18653/v1/2023.findings-acl.352. URL https://aclanthology.org/2023.findings-acl.352/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

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 *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 5447–5458, 2019. doi: 10.18653/v1/D19-1547. URL https://aclanthology.org/D19-1547/.

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 *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 1962–1979, 2019a. doi: 10.18653/v1/D19-1204. URL https://aclanthology.org/D19-1204/.

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 *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4511–4523, 2019b. doi: 10.18653/v1/P19-1443. URL https://aclanthology.org/P19-1443/.

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

Fuheng Zhao, Shaleen Deep, Fotis Psallidas, Avrilia Floratou, Divyakant Agrawal, and Amr El Abbadi. Sphinteract: Resolving ambiguities in NL2SQL through user interaction. *Proceedings of the VLDB Endowment*, 18(4):1145–1158, 2024. doi: 10.14778/3717755.3717772. URL https://doi.org/10.14778/3717755.3717772.

A Experimental Details and Reproducibility

This section consolidates the experimental conditions needed to interpret and reproduce the evaluations in Section 5. The operational details of DB-COPE itself are documented separately in Appendix B.

A.1 BIRD-INTERACT Protocol and Controlled Comparison

The primary evaluation uses all 600 tasks in the BIRD-Interact Full set, spanning 22 databases and comprising 410 business-intelligence (BI) queries and 190 data-management (DM) tasks. Every task has a Phase 2 follow-up. We use the a-Interact protocol described in Section 3: Phase 2 is released only after the Phase 1 submission passes the benchmark's functional evaluator, and the accepted program is executed before the follow-up is issued. Consequently, an accepted DM statement can change the database state observed in Phase 2. Interaction history and unused budget carry across the two phases. The functional evaluator judges execution-level equivalence rather than SQL string identity.

Within each system-agent backbone, Original ADK/ReAct and DB-COPE are run on the same task IDs and receive the same requests, initial database states, environment tools, tool prices, initial budgets, User Simulator, and functional evaluator. DB-COPE adds only the zero-cost internal actions for recording a Phase 1 plan, updating component states, and recording a Phase 2 plan. These actions expose no additional database, metadata, external knowledge, user, or evaluator information. Thus, the controlled comparison holds the environment-facing evidence and budget fixed while changing the agent's internal planning and interaction policy.

A.2 Models, Decoding, and Runtime Controls

The system-agent backbones are DeepSeek v4-Flash and GPT-5.6 Terra. Their decoding temperature is 0 and their output limit is 8192 tokens; all other sampling parameters are left at the respective provider defaults. The main comparisons, component ablations, and budget-sensitivity experiments use Gemini 2.5 Flash as the User Simulator with temperature 0. For the user-simulator robustness check in Table 2, we instead use DeepSeek v4-Flash as the User Simulator, also with temperature 0.

For a task with m_{amb} annotated critical and knowledge ambiguities and patience parameter p, the initial budget is $B_0 = 6 + 2m_{\text{amb}} + 2p$ bird-coins. The reported comparisons use $p = 3$, and any unused Phase 1 budget carries into Phase 2. The environmental action prices are those of the benchmark and are identical for Original ADK/ReAct and DB-COPE; Table 9 lists every price together with the additional initial-planning limits. Appendix B.5 also specifies the pre-execution budget gates, submission reserve, handling of rejected and terminal actions, clarification limits, and the common 60-turn model-call cap.

A.3 Metrics and Ablation Protocol

For task i, let $s_i^{(1)}$ and $s_i^{(2)}$ indicate functional-evaluator success in Phase 1 and Phase 2. Phase 2 success requires Phase 1 success. With $N = 600$, the reported metrics are

$$\begin{aligned} \text{NR} &= \frac{1}{N}\sum_{i=1}^{N}\left(0.7s_i^{(1)} + 0.3s_i^{(2)}\right), \\ \text{P1-SR} &= \frac{1}{N}\sum_{i=1}^{N} s_i^{(1)}, \qquad \text{P2-SR} = \frac{1}{N}\sum_{i=1}^{N} s_i^{(2)}. \end{aligned} \tag{6}$$

The per-task a-Interact reward is $0.7s_i^{(1)} + 0.3s_i^{(2)}$. The main comparison reports $100 \times \text{NR}$ as Reward*, while NR in the ablation table remains on the $[0, 1]$ scale.

The component ablations use the same 600 BIRD-Interact Full tasks as the DeepSeek v4-Flash main comparison and the same Gemini 2.5 Flash User Simulator. All settings other than the named ablated component are held fixed. The variants remove explicit component projection, executable SQL worlds, or sparse component-state updates, as defined in Section 5.3.

A.4 Budget Sensitivity

We compare Original ADK/ReAct and DB-COPE on the same 600 BIRD-Interact Full tasks while varying only patience $p \in \{0, 3, 7\}$ in the initial budget $B_0 = 6 + 2m_{\text{amb}} + 2p$. The default $p = 3$ condition uses the main-table results; all other model, simulator, task, tool, and evaluation settings are held fixed.

DB-COPE consistently outperforms Original ADK/ReAct across all budget settings, with a larger advantage at low and default budgets than under the most permissive setting. Both methods benefit from additional budget, but the larger improvement for Original indicates that DB-COPE is less sensitive to budget constraints. At the highest budget, DB-COPE retains its performance advantage while using fewer coins and a comparable number of user interactions.

A.5 PRACTIQ Protocol

The PRACTIQ diagnosis uses a fixed 2,836-example regenerated sample under the `lexicalOnly` condition. Sampling used seed 42 to match the category counts of the original evaluation; the results

Table 5: Budget sensitivity on the 600-task BIRD-INTERACT Full set, using DeepSeek v4-Flash and the Gemini 2.5 Flash User Simulator. Patience p controls the initial bird-coin allocation. NR is on a $[0, 1]$ scale, P1-SR and P2-SR are percentages, and coins and asks are task averages.

Method	p	NR	P1-SR	P2-SR	Coins	Asks
Original ADK/ReAct	0	0.1212	15.17	5.00	9.85	0.53
DB-COPE	0	0.1868	22.83	9.00	10.49	1.16
Original ADK/ReAct	3	0.1362	16.17	7.67	15.34	0.18
DB-COPE	3	0.2040	23.50	13.17	15.36	1.04
Original ADK/ReAct	7	0.1892	21.67	12.50	18.33	1.39
DB-COPE	7	0.2160	24.50	14.83	16.88	1.37

concern this regenerated sample, not the original test set. Seed 42 is only the PRACTIQ sampling seed; it does not denote repeated model runs. Each method receives the same question, schema, and retrieved example values. Gold labels, reference SQL, later dialogue turns, and database execution are unavailable. All three methods use DeepSeek v4-Flash, and the reported classification metrics are nine-way and three-way accuracy and macro-F1. Section 5.4 gives the three comparison conditions and token-accounting convention.

A.6 IMPLEMENTATION DETAIL MAP

The strict Phase 1 planning record and its fields are specified in Appendix B.1. Complete SQL-world construction and the SQLGlot structural validator are detailed in Appendix B.2; sparse state updates and Phase 2 reuse are detailed in Appendix B.3. Action prices, budget gates, clarification limits, terminal-action accounting, and the turn cap appear in Appendix B.5 and Table 9. Per-category PRACTIQ results are reported in Appendix C.

B METHOD AND IMPLEMENTATION DETAILS

B.1 PLANNING RECORD AND DECISION REPRESENTATION

Phase 1 planning is recorded as strict JSON rather than free-form reasoning. The record contains one state for every component in Eq. 2, but only applicable, result-changing choices affect world construction or clarification. Metric, threshold, and scope choices usually appear in expressions or predicates; grouping and aggregation determine result grain and reduction; join choices determine sources and row multiplicity; and sorting and projection determine ordering and output shape. These mappings are guides to interpretation rather than a complete grammar of SQL.

Table 6 summarizes the top-level record. Each component state contains its status, the remaining issue when unresolved, its choice in each available world, supporting evidence, and the expected effect on execution. When worlds exist, the validator derives a resolved component's canonical value from the selected world or from the value shared by both unselected worlds. With zero worlds, a resolved component instead supplies that value directly.

The component inventory does not fully describe a data-management statement. Its operation type, target table, assigned values, and update effects are left to final SQL construction and functional evaluation. They are not represented as additional component states or checked deterministically by the structural validator.

B.2 WORLD CONSTRUCTION AND STRUCTURAL VALIDATION

For an unresolved component c, let $\mathcal{V}_t(c)$ contain the alternatives supported by the evidence available at step t. If two worlds are constructed, the second changes every unresolved component for which a grounded alternative exists:

$$\begin{aligned} \mathcal{U}_t^{\text{alt}} &= \{c \in \mathcal{C} : \rho_t(c) = \texttt{unresolved},\ |\mathcal{V}_t(c)| \geq 2\}, \\ v_t^{(2)}(c) &\neq v_t^{(1)}(c), \qquad c \in \mathcal{U}_t^{\text{alt}}. \end{aligned} \tag{7}$$

Table 6: Top-level fields in the Phase 1 planning record.

Field	Content
`component_states`	One structured state, including an evidence field, for each component
`worlds`	Zero, one, or two grounded interpretations with read-only SQL
`comparison`	The selected world identifier, if evidence determines one
`decision`	Either clarification or progression to SQL submission
`clarification_target`	One unresolved component and a business-language question goal

Table 7: AST structures inspected for declared world contrasts.

Decision	Structural fingerprint
Metric formula	Arithmetic, functions, and `CASE` expressions
Threshold and filter scope	Comparisons and `WHERE`, `HAVING`, or aggregate filters
Grouping and aggregation	Grouping expressions and aggregate-function expressions
Join path	Source tables, joins, join types, and join predicates
Sorting	Ordering expressions, directions, and `LIMIT`
Projection	Projected expressions and output positions

This maximum-contrast rule avoids enumerating the Cartesian product of component choices. Both worlds must remain coherent joint interpretations; the rule does not license unsupported combinations merely to increase their contrast.

The number of worlds determines the permitted next action. Zero worlds are used when producing even one executable interpretation would require an unsupported definition; the record must identify an unresolved component and request clarification. A single world is permitted only when every result-changing component is resolved, the world contains no unsupported assumption, and the decision is to proceed. With two worlds, an unresolved difference leads to clarification with neither world selected; if evidence resolves all differences, one world may be selected for submission.

Every world is a complete PostgreSQL `SELECT` or `WITH` statement. For a data-management request, it selects the affected rows or computes candidate values; the modifying statement is composed only after the relevant decisions have been resolved.

The validator parses available worlds with SQLGlot under the PostgreSQL dialect. For each component whose world choices are declared different, it compares the corresponding normalized AST fingerprint in Table 7.

A declared difference with identical fingerprints is rejected. An explicit ascending-versus-descending contrast also requires different parsed order directions. These checks establish only that a declared choice changes the relevant SQL structure. They do not establish that either interpretation is supported or correct, compare query results on every database instance, discover undeclared differences, or compare the final submitted SQL with the resolved component state.

B.3 State Updates and Phase-2 Reuse

Several Phase 1 transitions are enforced by the parser and controller. The validated plan names exactly one unresolved component as the first clarification target, and the prompt instructs the model to ask only about that target. After `ask_user` returns, the next action must be the zero-cost `update_component_states`; exploration, another question, and submission remain blocked until a valid nonempty update succeeds. The parser permits updates only to components that were previously unresolved, while the server retains all other states.

After the first update, at most one residual Phase 1 clarification is allowed. The prompt permits this question to contain numbered subquestions for several remaining issues, but the answer again

Table 8: Paired trajectory diagnosis on the same 600 DeepSeek v4-Flash tasks used in Table 2. Counts are task counts.

Diagnostic	Count	Interpretation
Original Phase 1 fail → DB-COPE pass	65	Recovered by DB-COPE
Original Phase 1 pass → DB-COPE fail	21	Regression under DB-COPE
Original Phase 1 failures without SQL submission	268/503	Incomplete Original trajectories
DB-COPE Phase 1 failures without SQL submission	68/459	Fewer incomplete trajectories

requires an immediate sparse update. Once the question limit is reached, the prompt permits further resolution only from exact knowledge or execution evidence. Submission remains blocked while any component is marked unresolved.

These structural checks do not establish that an answer actually supports the new value: the parser requires a resolved update to cite evidence but does not judge its semantic validity. Final-program self-checking is similarly prompt-based. Before submission, the model is instructed to realize the confirmed formula, scope, grain, ordering, and output shape in SQL. After a rejected submission, it is instructed to make the smallest supported correction and retain independently grounded decisions. The implementation does not compare successive SQL queries component by component or verify the final query against the canonical state.

Phase 2 uses one lighter planning record instead of constructing new worlds. Before either asking a Phase 2 question or submitting follow-up SQL, the agent must call `record_phase2_planning_note` exactly once. The prompt asks the record to contain five parts: reusable Phase 1 components, follow-up changes, unresolved Phase 2 ambiguity, whether clarification is needed, and a proposed SQL sketch. It permits at most one question for a newly introduced ambiguity in the formula, threshold, filter scope, sorting, or result grain.

The runtime enforces the presence and order of this record and the one-question limit, but the tool stores the record as free-form structured text. It does not parse the five parts or compare the proposed sketch, inherited decisions, or follow-up changes with the submitted SQL. Phase 2 also disables `update_component_states` and does not invoke the world constructor or AST contrast validator.

B.4 Trajectory-Level Diagnosis and Qualitative Cases

Table 8 shows that, on the paired DeepSeek run, DB-COPE converts 65 Original Phase 1 failures into successes while regressing on 21 tasks, and reduces Phase 1 failures that end without any SQL submission from 268 to 68. Thus, the gain comes partly from carrying more interactions through planning and SQL submission, not only from improving final SQL correctness.

We inspect three tasks from the paired DeepSeek v4-Flash runs with the Gemini 2.5 Flash User Simulator. These examples were selected to illustrate distinct behaviors and are not estimates of their frequency in the full evaluation.

Unsupported assumption versus targeted clarification (`sports_events_16`). The request asks for the average pit-stop duration after excluding *bad records*, without defining that phrase. Original ADK/ReAct asks no question and assumes an unsupported upper cutoff of 120 seconds (`ms_count` $\leq$ 120000); its Phase 1 submission fails. DB-COPE instead records the filter as unresolved and asks what makes a record bad. After the first wording is unclear to the user, it asks a simpler question; the user specifies that missing, zero, and negative durations should be excluded. The resulting predicate, `ms_count` $>$ 0 `AND ms_count IS NOT NULL`, passes Phase 1 and is retained when the follow-up adds the minimum and maximum durations, which also passes.

Clarification before exploration exhausts the budget (`virtual_idol_M_3`). This data-management request asks to archive *older* interaction records for inactive users with basic memberships. Original ADK/ReAct explores the schema and date distribution until only the submission reserve remains. Its trace recognizes that the age cutoff is unresolved, but it can no longer both ask and submit, and the task ends without a submission. DB-COPE identifies the cutoff as the first result-changing decision and asks immediately. After the user defines *older* as more than three years,

Table 9: Action prices and Phase 1 clarification-planning limits. A dash denotes an action governed outside the initial planning stage.

Action	Cost	Initial planning limit
`get_schema`	1	1
`get_all_column_meanings`	1	1
`get_column_meaning`	0.5	4
`get_all_external_knowledge_names`	0.5	1
`get_knowledge_definition`	0.5	2
`get_all_knowledge_definitions`	1	unavailable
`execute_sql`	1	1
First `ask_user`	2	1
`submit_sql`	3	after plan
`record_planning_note`	0	exactly 1
`update_component_states`	0	–
`record_phase2_planning_note`	0	–

Table 10: Per-category recall (%) on the 2,836-example PRACTIQ `lexicalOnly` set with DeepSeek v4-Flash. Bold denotes the best result in each category.

Method	Ambig. Filter Criteria	Ambig. SELECT Column	Ambig. Values Within Column	Ambig. WHERE Column	Nonexist. Filter Value	Nonexist. SELECT Column	Nonexist. WHERE Column	Unsupported Join	Answerable
zero-shot	**93.73**	**91.81**	43.44	**89.52**	53.53	**92.74**	**89.41**	91.55	86.17
3-shot/class	91.75	91.23	42.62	87.62	72.94	92.12	88.56	89.20	87.72
DB-COPE	47.52	66.67	**46.72**	74.29	**90.00**	86.72	84.75	**94.37**	**94.20**

the canonical state is updated and the agent submits a copy-and-delete CTE. Both that statement and the follow-up that reruns the archive operation and counts the archive rows pass.

Correct state, but a repair drops an output constraint (`exchange_traded_funds_5`). In this failure case, the Phase 2 planning note correctly records that the answer must contain the top ten funds and, for each, the top three holdings, including ticker, holding rank, security name, and percentage weight. The first Phase 2 submission includes all four fields but fails evaluation. The subsequent repair preserves the top-ten and top-three filters but removes `positionrank` from the projection, despite that requirement remaining in the recorded state, and fails again. This example exposes the boundary of the current enforcement: the controller requires a Phase 2 record and orders actions, but it does not compare a repaired SQL statement against the canonical state component by component.

B.5 Budget and Runtime Controls

The environmental action prices are unchanged from the benchmark. For m_{amb} annotated critical and knowledge ambiguities and patience p, both methods receive $B_0 = 6 + 2m_{\text{amb}} + 2p$ bird-coins; the main comparison uses $p = 3$. The three internal state-management actions added by DB-COPE reveal no new environment information and have zero cost. Table 9 reports all action prices and the additional limits used during initial Phase 1 planning.

During initial planning, a cost-bearing grounding action is rejected without charge if it would reduce the remaining balance below 6 bird-coins. Following the first answer and state update, one additional Phase 1 question is allowed, for a total maximum of two. In Phase 2, exactly one planning record and at most one clarification are permitted. When no more than 4 bird-coins remain, `execute_sql` is disabled to preserve the submission budget.

An action rejected by a pre-execution gate is not charged. Both methods allow a terminal `submit_sql` call when the remaining balance is below its nominal price; the call consumes the balance and ends the trajectory. Reported bird-coin use follows this runtime accounting. Both methods also use the same 60-turn model-call cap. These limits and reserves are deterministic runtime settings rather than learned action values.

C ADDITIONAL PRACTIQ RESULTS

Table 10 reports per-category recall for the nine PRACTIQ labels. These results complement the aggregate accuracy and macro-F1 scores in Table 4.