WHAT DRIVES INTERACTION BUDGET SCALING IN TEXT-TO-SQL?

Anonymous authors
Paper under double-blind review

ABSTRACT

Interactive Text-to-SQL can resolve underspecified requirements through user clarification, yet larger interaction budgets do not consistently yield better SQL solutions. We study what makes additional interaction useful by comparing five representative methods and conducting controlled experiments on clarification evidence in BIRD-INTERACT-FULL. Context-only questioning is competitive initially, while guiding questions with differences between as few as two SQL candidates yields larger gains as the interaction budget increases. Wider pools bring earlier gains, but their advantage narrows with further interaction. Building on these findings, we introduce **DuetSQL**, which combines context-guided requirement elicitation with iterative SQL-contrast-guided clarification and uses execution agreement to control further sampling and questioning. Across three benchmarks, DuetSQL outperforms the strongest compared methods by 5–12 percentage points at matched interaction allowances. DuetSQL configurations also lie on the empirical Pareto frontiers in our comparisons under shared resource limits, offering favorable trade-offs between task success and resource use.

1 INTRODUCTION

Natural-language database requests often leave important requirements implicit: a term may admit multiple definitions, a calculation may depend on unstated conventions, or the expected output may be underspecified (Bhaskar et al., 2023; Saparina & Lapata, 2024). Interactive Text-to-SQL allows an agent to clarify these requirements with the user (Yao et al., 2019; Zhao et al., 2024; Chan et al., 2026). A larger interaction budget offers more opportunities to recover missing requirements. Ideally, these additional exchanges resolve uncertainty left after limited clarification and lead to SQL that better matches the user's intent.

However, a larger allowance does not ensure that these opportunities translate into better SQL (Figure 1). Existing approaches use task context or sampled candidate programs to guide clarification (Mu et al., 2024; Kobalczyk et al., 2025; Liu & Erdelt, 2026). Our comparison of several representative methods on BIRD-INTERACT-FULL (Huo et al., 2026) reveals limited or uneven gains from additional budget; even substantially more questions can yield little improvement in task success rate. The challenge is to use the available interaction to elicit requirements that help resolve the task. We therefore study *what drives interaction budget scaling*, focusing on how agents identify what to clarify and formulate questions whose answers help determine the final SQL.

We investigate the evidence available to the question generator under a shared interaction protocol, comparing questions generated from context alone, a single SQL, and contrasts between multiple SQL candidates. Context-only questioning is competitive at the initial budget, while using differences between just two SQLs supports stronger gains with additional interaction. Wider candidate pools can reach high success rates earlier, while additional interaction narrows the observed performance gap between pool widths. These findings suggest complementary roles for direct requirement elicitation and SQL contrasts, with candidate width and interaction budget considered jointly.

Further questions demand user effort, while wider pools and repeated sampling increase model computation. We therefore introduce **DuetSQL**, which combines context-guided and SQL-contrast-guided clarification within a shared dialogue. It first elicits requirements without candidate sampling, allowing the replies to constrain subsequent SQL exploration. Differences among the updated

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.

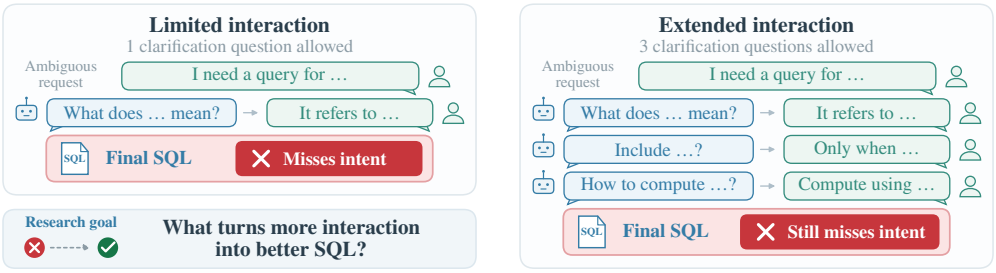


Figure 1: **What makes additional interaction useful?** For the same task and agent, more interaction may still leave the final SQL inconsistent with user intent.

candidates guide further questions, while execution agreement controls when that exploration ends. A final context-guided step confirms output requirements while preserving clarified constraints.

We evaluate DuetSQL on BIRD-INTERACT-FULL, BIRD-INTERACT-LITE (Huo et al., 2026), and AMBROSIA-INTERACT, our interactive adaptation of AMBROSIA (Saparina & Lapata, 2024). On BIRD-INTERACT-FULL, it achieves a task success rate of 45.67% at an additional question allowance of $P = 9$, compared with 31.50% for the strongest external method at the same allowance. Component ablations quantify each stage's contribution; comparisons under shared resource limits identify favorable trade-offs among success, tokens, and questions.

Our contributions are twofold:

- **An empirical study of interaction budget scaling.** We characterize representative methods' budget responses. Controlled comparisons show stronger gains from contrasting as few as two SQLs and earlier gains from wider pools, with their observed advantage narrowing as interaction increases.
- **DuetSQL for effective and efficient clarification.** We develop a framework combining context-guided and SQL-contrast-guided clarification. Through component and resource analyses, we identify practical configurations that offer strong task success under different interaction and computation limits.

2 INTERACTION PROTOCOL

Interactive Text-to-SQL seeks an SQL program that fulfills a user's intent through interaction with the user and a database environment. For a task, the initial input $x = (u, \mathcal{S}, \mathcal{K})$ comprises a natural-language request u, database schema $\mathcal{S}$, and supplied background knowledge $\mathcal{K}$. The system accesses database $\mathcal{B}$ through permitted operations and may clarify requirements left ambiguous or unspecified in u. The output $\hat{s}$ is an SQL program, possibly containing multiple statements, that retrieves information or modifies the database as requested. A task-specific verifier determines success, $y = \mathcal{V}(\hat{s}; \mathcal{B}) \in \{0, 1\}$, with $y = 1$ indicating that the task requirements are satisfied. A missing or invalid submission receives $y = 0$. The verifier's correctness feedback is hidden during solving; database execution results and errors remain accessible where the protocol permits.

We represent the task's user by $\mathcal{U}$ and write its reply to a clarification question q_t as $r_t = \mathcal{U}(q_t)$. The user answers based on the intended requirements and available background or database information; dialogue access is specified by the protocol, and replies may include refusals. Each task allows at most $Q_{\max}$ questions. For BIRD-Interact (Huo et al., 2026), $Q_{\max} = A + P$, where A is the task-specific base allowance (Appendix A.1) and P is the additional budget shared across tasks at each experimental setting. Thus, $P = 0$ still retains the base allowance. For AMBROSIA-INTERACT, P directly specifies the total question allowance, $Q_{\max} = P$. With Q_t questions already asked before step t and $Q_1 = 0$, the method chooses an action

$$a_t \in \begin{cases} \{\mathsf{ask}(q_t), \mathsf{submit}(\hat{s})\}, & Q_t < Q_{\max}, \\ \{\mathsf{submit}(\hat{s})\}, & Q_t = Q_{\max}. \end{cases} \tag{1}$$

Every issued question consumes one unit, even if the reply is refused or unavailable. Any permitted repair precedes the final submission $\mathsf{submit}(\hat{s})$, which fixes $\hat{s}$ and ends the task. Let

$Q \in \{0, \ldots, Q_{\max}\}$ be the total number of questions actually issued during the task. The interaction budget constrains clarification questions only; internal computation is left to the method.

At budget P, the task success rate $\mathrm{SR}(P)$ is the percentage of tasks with $y = 1$, and $\bar{Q}(P)$ is the mean actual question count over the same evaluation set, including failed tasks.

3 What Makes Additional Interaction Useful?

3.1 External Budget Responses

We first examine how representative interactive Text-to-SQL methods respond to additional interaction budget on BIRD-Interact-Full. We group them by candidate-pool use: BIRD-Interact Conv (Huo et al., 2026) and Sphinteract (Zhao et al., 2024) use no dedicated pool, with Sphinteract instead revising one current SQL; ClarifyGPT (Mu et al., 2024) and PleaSQLarify (Chan et al., 2026) use a fixed initial pool; and Active Task Disambiguation (ATD) (Kobalczyk et al., 2025) resamples a pool round-wise. We retain method-specific clarification and stopping rules where compatible with the shared protocol; Appendix C documents the adaptations and exceptions. For every task, $P \in \{0, 3, 5, 7, 9\}$ augments the base allowance A, so $P = 0$ still permits task-specific clarification.

Figure 2 reports task success rates over the same 600 tasks. Its summary columns show success at $P = 0$, the net change from $P = 0$ to 9 (Gain, in percentage points), and the highest success rate across tested budgets (Peak); definitions appear in Appendix E, Eq. (4). The comparison is descriptive because interaction procedures differ across methods.

We define the *full-information reference* as the task success rate obtained when the same model receives the original unambiguous request, schema and column descriptions, and complete unmasked external knowledge upfront. The model generates one initial SQL without user interaction, allowing at most one terminal execution-error repair. This yields 39.33%, a measured reference rather than an upper bound on iterative solving (Appendix A.1).

Later-budget gains are limited, while realized interaction varies. From $P = 3$ to 9, no method gains more than 2.00 percentage points. BIRD-Interact Conv, Sphinteract, and PleaSQLarify ask few additional questions as the budget grows. ATD increases its mean question count from 2.87 to 11.26 between $P = 0$ and 9, yet remains below 28% success. Thus, a larger allowance need not yield more questions, and more questions alone need not yield stronger gains.

Method	$P=0$	Gain	Peak
BIRD-Interact Conv	26.50	+2.83	29.33
ClarifyGPT	26.00	+5.50	31.50
PleaSQLarify	18.50	+0.33	19.00
Sphinteract	27.33	+1.67	29.00
ATD	21.83	+5.33	27.50

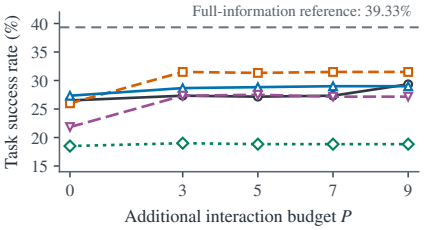


Figure 2: Budget responses on BIRD-Interact-Full.

These external results do not isolate why a method benefits from interaction: question generation, candidate updates, stopping, and final submission all vary together. ATD illustrates a relevant mechanism distinction. It resamples its candidate pool each round, but proposes candidate questions from the task and dialogue independently of that pool; the pool is then used to rank those proposals by expected information gain. This observation does not causally explain ATD's curve, but it separates resampling SQL candidates from using their current differences to formulate questions. We therefore next compare, under a shared protocol, question generation from context alone, one SQL, and SQL contrasts.

3.2 Controlled Clarification Conditions

SQL candidates express interpretations of an underspecified request, and their differences may reveal requirements worth clarifying (Mu et al., 2024; Liu & Erdelt, 2026). We study how the SQL evidence shown to the question generator changes the gains from additional interaction. Across $P = 0$–19, we ask whether contrasting two SQLs changes the response relative to conditions without SQL contrasts, and how increasing the candidate-pool width changes the curve.

Let D denote the number of SQL candidates sampled per round. The question generator always receives the task and dialogue context. *Context-only* (no SQL candidates, $D = 0$) asks directly from this context; *Single SQL* ($D = 1$) adds one interpretation to inspect; *SQL contrasts* ($D = 2, 4, 8, 16$) use differences among multiple candidates to identify what to ask. Each reply updates the context from which subsequent candidates and questions are generated.

All controlled conditions use the 600 BIRD-INTERACT-FULL tasks, record every integer P from 0 to 19, and continue to $Q_{\max} = A + P$ unless interrupted by a failure. SQL candidates are shown without diagnostic execution, repair, or grouping. We report task success rates, allowing at most one terminal execution-error repair; Appendix E.1 gives the implementation and evaluation details.

SQL contrasts support stronger budget gains. At $P = 0$, Context-only and $D = 2$ both achieve 27.83%, while Single SQL achieves 25.33%. Across the full integer $P = 0$–19 trajectories, the two conditions without SQL contrasts never exceed the 39.33% full-information reference: Context-only and Single SQL each peak at 38.83%. In contrast, $D = 2$ reaches 43.83% at $P = 19$, 4.50 percentage points above the reference. The observed separation therefore develops with additional budget rather than from a higher $D = 2$ starting result. The gain from $D = 1$ to $D = 2$ suggests that contrasting SQL interpretations matters beyond providing SQL evidence.

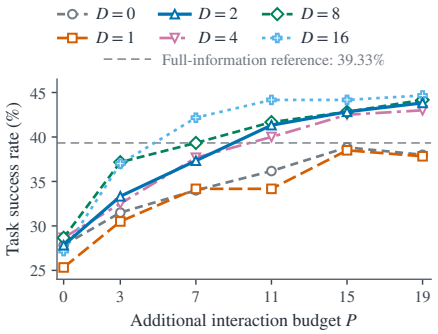


Figure 3: SQL evidence and pool width. Full-information reference: 39.33%.

Wider pools bring earlier gains. At $P = 7$, $D = 16$ reaches 42.17%, while $D = 2, 4, 8$ reach 37.33%, 37.67%, and 39.33%, respectively. By $P = 19$, the four SQL-contrast conditions lie between 43.00% and 44.67%, a spread of 1.67 points. Wider pools thus lead at this earlier budget, while further interaction narrows the gap between widths.

Key findings.

- Contrasting just two SQLs supports stronger budget gains and exceeds the full-information reference; Context-only and Single SQL remain below it throughout the tested budget range.
- Wider contrast pools can reach high success rates earlier; at high budgets, the observed performance gap between candidate widths narrows.

4 DUETSQL: CLARIFICATION WITH CONTEXT AND SQL CONTRASTS

Section 3.2 shows that context-only questioning is competitive initially, whereas SQL contrasts yield larger gains with further interaction. These findings motivate combining both clarification sources with explicit control over candidate computation.

DuetSQL combines **context-guided clarification** with **SQL-contrast-guided clarification** through a **shared dialogue** (Figure 4). Request Clarification first elicits requirements directly from the user without candidate sampling; these answers constrain subsequent candidate generation. The SQL-Contrast-Guided Loop uses differences among these candidates to guide further questions, whose answers refine the dialogue used to generate the next pool. Output Specification then returns to context-guided clarification to confirm the intended output and retain or revise the current SQL while preserving previously established requirements.

All stages share the schema $\mathcal{S}$, knowledge $\mathcal{K}$, and question allowance $Q_{\max}$. Starting from $\mathcal{H}_0 = (u)$, each question q_t and user reply $r_t = \mathcal{U}(q_t)$ are appended to form $\mathcal{H}_t$; the question index t continues across stages. Algorithm 1 in Appendix D specifies the complete control flow.

4.1 REQUEST CLARIFICATION

The initial strength of context-only questioning motivates **eliciting requirements before candidate sampling**. These replies constrain SQL generation without a pool for each question.

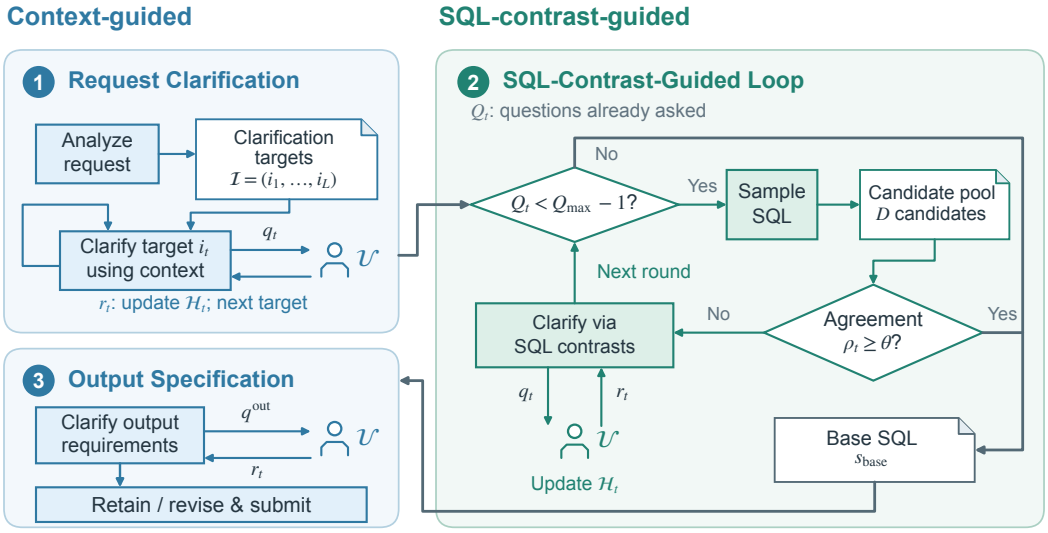


Figure 4: Overview of DuetSQL. Context-guided and SQL-contrast-guided clarification share a dialogue history $\mathcal{H}_t$; repeated $\mathcal{U}$ symbols denote the same user.

Using the schema $\mathcal{S}$ and supplied knowledge $\mathcal{K}$, Request Clarification analyzes the user's request u to identify ambiguities in meaning, scope, or calculation requirements that could affect the SQL. This initial *request analysis* produces a fixed, ordered list of *clarification targets*, $\mathcal{I} = f_{\text{req}}(u; \mathcal{S}, \mathcal{K}) = (i_1, \ldots, i_L)$.

DuetSQL addresses these targets through sequential question–reply rounds. At round t, it generates $q_t = g_{\text{req}}(i_t; \mathcal{S}, \mathcal{K}, \mathcal{H}_{t-1})$ from the next target i_t and current dialogue history. The pair (q_t, r_t) enters $\mathcal{H}_t$ before another question is generated, so later questions incorporate earlier answers.

The stage addresses up to H targets in list order, subject to $Q_{\max}$. Later stages begin only after the selected targets have been addressed and question budget remains.

4.2 SQL-Contrast-Guided Loop

The gains observed with as few as two SQLs motivate using candidate differences to expose competing interpretations and guide further clarification. The loop checks $Q_t < Q_{\max} - 1$ before constructing each candidate pool, reserving one question for Output Specification.

At each round, DuetSQL samples D SQL candidates $s_{t,1}, \ldots, s_{t,D}$ conditioned on $\mathcal{S}$, $\mathcal{K}$, and $\mathcal{H}_{t-1}$. Each candidate makes an interpretation of the clarified request concrete in SQL. We execute each candidate, allow at most one repair on execution failure, and discard persistent failures without replacement. The resulting candidate pool $\mathcal{C}_t$ contains executable SQL candidates (original or repaired) paired with their query results or observed database effects, with at most one entry per sampled candidate. Appendix D.1 details the execution evidence and grouping.

Agreement has been used for candidate selection and adaptive sampling (Chen et al., 2022; Aggarwal et al., 2023). Here, we group $\mathcal{C}_t$ by matching query results or database effects and use agreement as a stopping heuristic for this loop. For a nonempty pool of $n_t = |\mathcal{C}_t| \leq D$ candidates, let m_t be the size of the largest group. SQL contrasts guide questioning below the agreement threshold:

$$\rho_t = \frac{m_t}{n_t}, \qquad q_t = g_{\text{ctr}}(\mathcal{C}_t; \mathcal{S}, \mathcal{K}, \mathcal{H}_{t-1}) \quad \text{if } \rho_t < \theta. \tag{2}$$

If $\rho_t \geq \theta$, the loop passes control to Output Specification. Otherwise, the question generator reads all executable candidates and their observations together with the task context and preceding replies. It is prompted to **identify a requirement on which candidates disagree**—such as a metric definition, data scope, or ordering—and formulate one brief question in natural language. The question concerns user intent, leaving schema mapping to the agent. After (q_t, r_t) enters $\mathcal{H}_t$, the loop returns to the budget check. If another round is permitted, resampling from the updated dialogue lets newly clarified requirements reshape the candidate contrasts.

The width D controls how many SQL candidates are sampled per round, whereas the threshold θ controls when their agreement ends SQL-contrast-guided questioning.

4.3 Output Specification

Candidates may **share omissions in output requirements** even when their executions agree. Output Specification uses context-guided clarification to confirm these requirements with the user.

Let s_{base} denote the SQL available before the final output question: the first original candidate at an agreement stop, or one SQL generated from the current dialogue when only one question remains. The stage asks a fixed question, $q_t = q^{\text{out}}$, for the task type. For queries, it confirms output items and ordering; for management tasks, it confirms the intended database state, objects to create, and operations to execute. Selecting this question requires no model call.

The reply updates $\mathcal{H}_t$. For an answerable reply, one model call retains or minimally revises s_{base} using the updated dialogue, preserving established requirements. An unanswerable reply leaves it unchanged. The resulting SQL is submitted as $\hat{s}$ after any permitted execution-error repair.

5 Experiments and Analysis

We evaluate DuetSQL's task success rate and resource use across datasets, followed by component contributions, cost–performance trade-offs, and generalization across models.

Experimental setup and metrics. We evaluate all 600 BIRD-INTERACT-FULL tasks, all 300 BIRD-INTERACT-LITE tasks, and a fixed 300-task AMBROSIA-INTERACT subset, using `deepseek-v4-flash` for all methods and `deepseek-v4-pro` as the user simulator (DeepSeek-AI, 2026) (USERSIM-GUARD validation: Appendix B.3). DuetSQL uses $H = 5$, $D = 8$, Output Specification, and agreement-based early stopping at $\theta = 0.8$. For BIRD-INTERACT-FULL and BIRD-INTERACT-LITE, we use $Q_{\max} = A + P$ with $P \in \{0, 3, 5, 7, 9\}$. For AMBROSIA-INTERACT, $P = Q_{\max} \in \{0, 1, 2, 3, 4, 5\}$ specifies the total question allowance. Actual question use depends on each method. We report task success rate, mean issued questions $\bar{Q}$, and mean model input plus output tokens $\bar{T}$ (in millions), averaged over the full evaluation set. Token costs cover the method's computation through final submission, excluding the user simulator and hidden evaluation. Appendices A.1 and A.2 detail evaluation and resource accounting.

5.1 Overall Performance and Resource Use

Table 1 compares task success and resource use at $P = 5$ for both BIRD-Interact datasets and $P = 3$ for AMBROSIA-INTERACT. We select these representative budgets near the middle of the evaluated ranges, applying the same allowance to all methods within each benchmark.

Table 1: Task success rates and resource use at intermediate interaction budgets.

	BIRD-INTERACT-FULL $P=5$			BIRD-INTERACT-LITE $P=5$			AMBROSIA-INTERACT $P=3$		
Method	Success ↑	$\bar{T}$ ↓	$\bar{Q}$ ↓	Success ↑	$\bar{T}$ ↓	$\bar{Q}$ ↓	Success ↑	$\bar{T}$ ↓	$\bar{Q}$ ↓
BIRD-Interact Conv	27.17	**0.058**	1.99	37.33	**0.058**	2.22	38.67	**0.003**	**0.49**
Sphinteract	28.83	0.086	2.20	31.33	0.096	2.54	61.67	0.010	0.93
ClarifyGPT	31.33	0.442	3.82	35.33	0.464	4.44	74.00	0.037	2.53
PleaSQLarify	18.83	0.878	**1.25**	16.33	0.925	**1.46**	51.00	0.085	1.09
ATD	27.50	2.763	7.61	33.67	3.274	8.64	80.67	0.112	2.99
DuetSQL	**42.33**	0.741	6.79	**49.33**	0.656	6.91	**85.67**	0.015	2.49

DuetSQL achieves the highest task success across all three datasets. Under the selected interaction budgets, DuetSQL reaches 42.33% on BIRD-INTERACT-FULL, 49.33% on BIRD-INTERACT-LITE, and 85.67% on AMBROSIA-INTERACT. These exceed the strongest external method on each dataset—ClarifyGPT (31.33%), BIRD-Interact Conv (37.33%), and ATD (80.67%), respectively—by 11.00, 12.00, and 5.00 percentage points. Complete budget results for BIRD-INTERACT-FULL and AMBROSIA-INTERACT appear in Appendices E.2 and E.3.

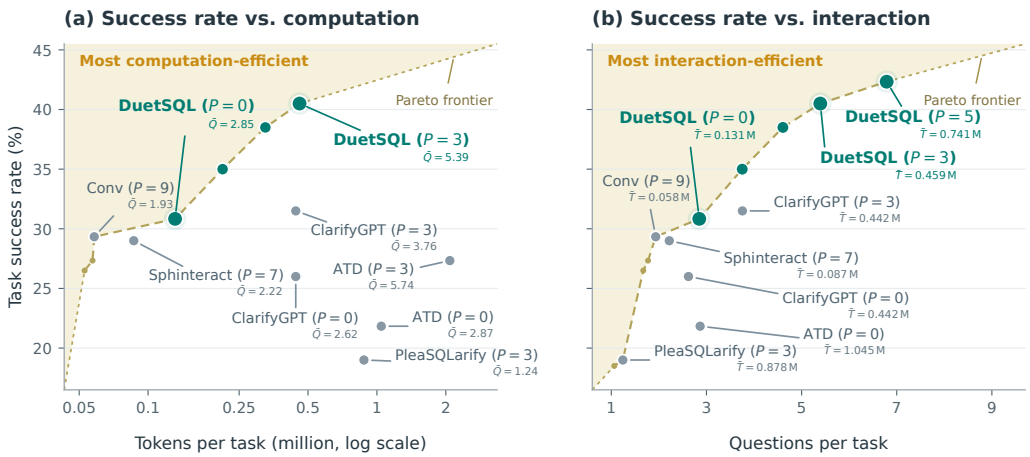


Figure 5: Task success and resource trade-offs on BIRD-INTERACT-FULL. Panel (a) compares tokens under $\bar{Q} \leq 6$; panel (b) compares questions under $\bar{T} \leq 1.10$ million. Smaller green points denote supplementary DuetSQL measurements at $P = 1, 2$. Dashed lines indicate empirical Pareto frontiers; connecting and extended segments are visual guides.

DuetSQL uses more tokens and questions than several BIRD baselines, whose candidate generation and stopping rules differ. To assess performance at comparable resource use, Figure 5 compares success against one resource while capping the other.

DuetSQL achieves higher success at comparable computation. With mean questions capped at six in Figure 5(a), DuetSQL at $P = 3$ reaches 40.50% success versus ClarifyGPT's 31.50%, using 0.459 versus 0.442 million tokens. Both satisfy the question ceiling, although DuetSQL asks more questions (5.39 versus 3.76). On AMBROSIA-INTERACT, DuetSQL also achieves higher success than ATD while using 86.8% fewer tokens at the reported budget.

DuetSQL offers favorable trade-offs across interaction levels. Under the shared token ceiling in Figure 5(b), DuetSQL achieves 30.83%, 40.50%, and 42.33% success with 2.85, 5.39, and 6.79 mean questions at $P = 0, 3, 5$, respectively. All displayed DuetSQL points lie on the empirical Pareto frontiers among the recorded eligible configurations in both panels, providing choices for lower resource use or higher task success. On AMBROSIA-INTERACT, DuetSQL also uses 16.7% fewer questions than ATD despite its higher success rate.

5.2 COMPONENT ANALYSIS

We remove individual components from the complete DuetSQL configuration. Table 2 averages success rates and resource use equally over $P \in \{0, 3, 5, 7, 9\}$ on BIRD-INTERACT-FULL.

All variants share the allowance $A + P$. When retained, Request Clarification uses $H = 5$ and the loop uses $D = 8, \theta = 0.8$. Each ablation removes one whole stage, including its questions and replies; the other stages keep their settings and use only the remaining dialogue. Appendix E.5 reports per-budget results.

Table 2: DuetSQL ablations averaged across interaction budgets.

Variant	Success (%)	$\bar{T}$ (M)	$\bar{Q}$
DuetSQL	**40.63**	0.716	6.45
w/o SQL-Contrast-Guided Loop	34.53	0.087	3.47
w/o Request Clarification	39.50	1.044	5.94
w/o Output Specification	36.03	0.823	6.15

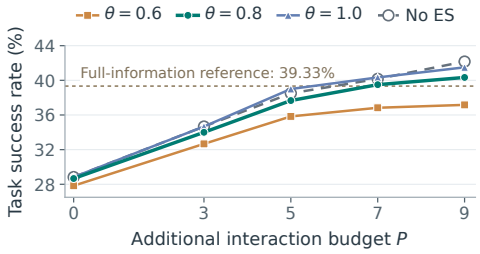
The SQL-Contrast-Guided Loop provides the largest gain in average task success. Retaining the loop improves mean success by 6.10 percentage points, at higher token and question costs. Without it, the method submits after Request Clarification and Output Specification, limiting gains from additional allowance. At $P = 9$, the gain reaches 10.00 points (Appendix E.5, Table 10).

Request Clarification reduces computation at comparable task success. Retaining this stage reduces mean model tokens by 31.4% while maintaining a similar average success rate, although

mean question use increases. This supports our design of starting with context-guided clarification, motivated by its competitive initial performance (Section 3.2). The success-rate difference varies with budget: omitting this stage performs better at the smallest tested allowance.

Output Specification further improves average task success. Retaining this stage adds 4.60 percentage points on average. It confirms output requirements within the existing question allowance and retains or revises the base SQL, whereas the ablation submits its original endpoint SQL. The comparison measures the complete terminal component.

5.3 Cost–Performance Trade-offs



Stopping	Success (%)	$\bar{T}$ (M)	$\bar{Q}$	$\bar{G}\uparrow$
$\theta = 0.6$	34.07	0.718	5.45	7.07
$\theta = 0.8$	36.03	0.823	6.15	6.31
$\theta = 1.0$	36.87	0.866	6.44	5.78
No ES	36.87	1.064	7.69	3.46

Figure 6: Stopping settings without Output Specification: budget curves and summary metrics.

We prioritize task success within a limited question budget. Wider pools bring earlier gains, motivating $D = 8$, although $D = 2$ approaches their performance with further interaction (Section 3.2).

Preserving task success at lower cost. Execution-group agreement limits further sampling and questioning through early stopping (ES). We fix $H = 5, D = 8$ on all 600 BIRD-INTERACT-FULL tasks. No ES ignores agreement; $\theta = 1$ requires unanimity. Using the earlier terminal-generation variant of Output Specification, at $P = 9$, $\theta = 0.8$ achieves 42.83% success versus 44.17% without ES, retaining 97.0% of the success rate with 25.4% fewer tokens and 22.2% fewer questions.

Balancing efficiency and task success rate. To select θ, we compare task success and incremental resource efficiency across four stopping settings without Output Specification (Figure 6). For sampled budgets $P_0 < \cdots < P_K$, with K adjacent intervals, we define

$$\Delta_i X = X(P_i) - X(P_{i-1}), \qquad G_i = \frac{\Delta_i \mathrm{SR}}{\Delta_i \bar{T}\, \Delta_i \bar{Q}}, \qquad \bar{G} = \frac{1}{K} \sum_{i=1}^{K} G_i. \tag{3}$$

With Δ_iSR in percentage points and $\bar{T}$ in millions of tokens, G_i measures success gains relative to the product of additional tokens and questions on the fixed budget grid.

$\theta = 0.6$ has the highest $\bar{G}$ but stays below the full-information reference. Our default $\theta = 0.8$ has the highest $\bar{G}$ and lowest resource use among settings reaching this reference.

5.4 Generalization Across Models

We test the SQL-contrast finding and DuetSQL with `qwen3.5-122b-a10b` (Qwen Team, 2026), using the same 600 BIRD-INTERACT-FULL tasks, user simulator, and evaluation protocol.

The benefit of SQL contrasts recurs on Qwen. We repeat the Context-only ($D = 0$), Single SQL ($D = 1$), and SQL contrasts ($D = 2$) controls from Section 3.2, generating new Qwen candidates and dialogues at $P \in \{0, 3, 7, 11, 15, 19\}$. $D = 0$ and $D = 2$ start similarly, but $D = 2$ pulls ahead at higher budgets. Only $D = 2$ exceeds Qwen's full-information reference (34.17%) at the displayed budgets from $P = 11$ onward (Figure 7). The weaker gains with a single SQL support the role of contrasting alternatives in improving budget use on Qwen.

DuetSQL retains its advantage on Qwen. All methods use Qwen with unchanged configurations. We prioritize comparisons with BIRD-Interact Conv and ClarifyGPT because they lie near the resource frontiers in Figure 5. At $P = 5$, DuetSQL exceeds ClarifyGPT by 6.17 percentage points (Table 3). At $P = 3$, it still outperforms ClarifyGPT at $P = 5$ with fewer tokens but more questions.

The combined design thus remains effective on Qwen and offers a lower-token configuration when more questions are acceptable.

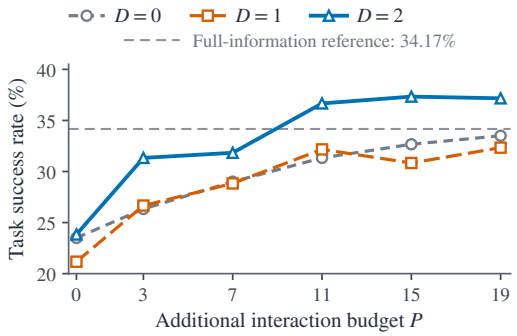


Figure 7: Controlled clarification on Qwen.

Table 3: Qwen performance and costs.

Method	P	Success (%)	$\bar{T}$ (M)	$\bar{Q}$
BIRD-Interact Conv	5	26.17	0.056	1.66
ClarifyGPT	5	29.83	0.465	3.94
DuetSQL	3	32.67	0.276	5.42
DuetSQL	5	**36.00**	0.521	6.52

6 Related Work

Clarification and task disambiguation. Clarification and task disambiguation help resolve uncertainty in natural-language requests and identify user intent. Benchmarks cover conversational search, question answering, and tool use (Aliannejadi et al., 2019; Kumar & Black, 2020; Aliannejadi et al., 2020; Min et al., 2020; Suri et al., 2026). Clarification methods use user replies to guide code generation and conversational agents (Mu et al., 2024; Kobalczyk et al., 2025; Chen et al., 2025; Acikgoz et al., 2026).

Text-to-SQL and user interaction. Text-to-SQL has been extensively studied, with substantial work on translating natural-language requests into SQL in a single turn (Pourreza & Rafiei, 2023; Gao et al., 2024; Pourreza et al., 2025; Liu et al., 2026). User interaction helps clarify ambiguous requests and recover unspecified requirements. Related benchmarks address conversational database querying, ambiguous query interpretation, and interactive task completion (Yu et al., 2019; Bhaskar et al., 2023; Saparina & Lapata, 2024; Chen et al., 2025; Huo et al., 2026). Interactive semantic parsing and Text-to-SQL methods use clarification and corrective feedback to align queries with user intent (Yao et al., 2019; Elgohary et al., 2020; 2021; Zhao et al., 2024; Chan et al., 2026; Ding et al., 2026; Qiu et al., 2025; Liu & Erdelt, 2026; Wang et al., 2026).

7 Conclusion

We studied how interactive Text-to-SQL turns additional question allowances into task success. Existing methods show limited or uneven gains, while controlled comparisons show that contrasting as few as two SQLs can support stronger scaling. Wider pools bring earlier gains, but their observed advantage narrows at larger budgets.

Building on these findings, DuetSQL combines context-guided requirement elicitation with iterative SQL-contrast-guided clarification and uses execution agreement to control candidate exploration. Experiments on BIRD-Interact-Full and AMBROSIA-Interact show improvements in task success rate at representative budgets. Component and resource analyses identify configurations that retain strong performance with fewer tokens and questions, guiding resource allocation.

Limitations. Our main evaluation uses a fixed pair of models for task solving and user simulation. Broader experiments with Claude (Anthropic, 2025) and other model families are needed to assess whether the observed budget responses and preferred configurations generalize across models. Our resource analysis measures mean issued questions $\bar{Q}$ and mean model tokens $\bar{T}$, but does not yet quantify database execution costs or end-to-end task completion time, both of which matter in practical deployments. Measuring these dimensions across database workloads and system configurations would provide a more complete assessment of system efficiency.

AI USE STATEMENT

We used generative AI tools to help draft and revise sections of this manuscript, improve language clarity, and develop and refine research code. AI tools also assisted with creating plotting scripts and schematic diagrams and refining figure layouts. The authors directed the research and manuscript development and take responsibility for all AI-assisted text, code, and figures, as well as the accuracy and integrity of the methodology, experimental results, and conclusions.

REPRODUCIBILITY STATEMENT

We provide the research code as supplementary material to support reproduction of our experiments. Section 2 and Appendix A.1 describe the task setup, interaction protocol, and evaluation procedure; Appendix A.2 defines resource accounting. The user simulator and its validation are documented in Appendices B.1–B.3. Baseline configurations are provided in Appendices C.1–C.5. Algorithm 1 gives DuetSQL's control flow, and Appendix D.1 details execution evidence and grouping. Appendix E reports additional experimental settings and results across interaction budgets.

REFERENCES

Emre Can Acikgoz, Jinoh Oh, Jie Hao, Joo Hyuk Jeon, Heng Ji, Dilek Hakkani-Tur, Gokhan Tur, Xiang Li, Chengyuan Ma, and Xing Fan. SpeakRL: Synergizing reasoning, speaking, and acting in language models with reinforcement learning. In Giuseppe Riccardi, Seyed Mahed Mousavi, Maria Ines Torres, Koichiro Yoshino, Zoraida Callejas, Shammur Absar Chowdhury, Yun-Nung Chen, Frederic Bechet, Joakim Gustafson, Géraldine Damnati, Alex Papangelis, Luis Fernando D'Haro, John Mendonça, Raffaella Bernardi, Dilek Hakkani-Tur, Giuseppe "Pino" Di Fabbrizio, Tatsuya Kawahara, Firoj Alam, Gokhan Tur, and Michael Johnston (eds.), *Proceedings of the 16th International Workshop on Spoken Dialogue System Technology*, pp. 312–325, Trento, Italy, February 2026. Association for Computational Linguistics. URL https://aclanthology.org/2026.iwsds-1.32/.

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.761. URL https://aclanthology.org/2023.emnlp-main.761/.

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, pp. 475–484, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331265. URL https://doi.org/10.1145/3331184.3331265.

Mohammad Aliannejadi, Julia Kiseleva, Aleksandr Chuklin, Jeff Dalton, and Mikhail Burtsev. Convai3: Generating clarifying questions for open-domain dialogue systems (clariq), 2020. URL https://arxiv.org/abs/2009.11352.

Anthropic. System card: Claude opus 4 & claude sonnet 4, 2025. URL https://www.anthropic.com/claude-4-system-card.

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. Benchmarking and improving text-to-SQL generation under ambiguity. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=a0yFO9gKc5.

Robin Shing Moon Chan, Rita Sevastjanova, and Mennatallah El-Assady. PleaSQLarify: Visual pragmatic repair for natural language database querying. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pp. 1–18. ACM, April 2026. doi: 10.1145/3772318.3791265. URL http://dx.doi.org/10.1145/3772318.3791265.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests, 2022. URL https://arxiv.org/abs/2207.10397.

Maximillian Chen, Ruoxi Sun, Tomas Pfister, and Sercan O Arik. Learning to clarify: Multi-turn conversations with action-based contrastive self-training. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SIE6VFps9x.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

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*, SIGMOD Companion '26, pp. 26–29, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400724503. doi: 10.1145/3788853.3801581. URL https://doi.org/10.1145/3788853.3801581.

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*, 2020.

Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo Ramos, and Ahmed Hassan Awadallah. NL-EDIT: Correcting semantic parse errors through natural language interaction. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5599–5610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.444. URL https://aclanthology.org/2021.naacl-main.444/.

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. *Proc. VLDB Endow.*, 17(5):1132–1145, January 2024. ISSN 2150-8097. doi: 10.14778/3641204.3641221. URL https://doi.org/10.14778/3641204.3641221.

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 *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=nHrYBGujps.

Kasia Kobalczyk, Nicolás Astorga, Tennison Liu, and Mihaela van der Schaar. Active task disambiguation with LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=JAMxRSXLFz.

Vaibhav Kumar and Alan W Black. ClarQ: A large-scale and diverse dataset for clarification question generation. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7296–7301, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.651. URL https://aclanthology.org/2020.acl-main.651/.

Leonhard Liu and Patrick K. Erdelt. Two surfaces of ambiguity: Complementary detection for Text-to-SQL. In *VLDB 2026 Workshop: Applied AI for Database Systems and Applications (AIDB)*, 2026. URL https://vldb.org/2026/Workshops/VLDB-Workshops-2026/AIDB/aidb26_10.pdf.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. Xiyan-sql: A novel multi-generator framework for text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–14, 2026. doi: 10.1109/TKDE.2026.3657851.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. AmbigQA: Answering ambiguous open-domain questions. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5783–5797, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.466. URL https://aclanthology.org/2020.emnlp-main.466/.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, ChenXue Wang, Shichao Liu, and Qing Wang. ClarifyGPT: A framework for enhancing LLM-based code generation via requirements clarification. *Proc. ACM Softw. Eng.*, 1(FSE), July 2024. doi: 10.1145/3660810. URL https://doi.org/10.1145/3660810.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=p53QDxSIc5.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CvGqMD5OtX.

Luyu Qiu, Jianing Li, Chi Su, and Lei Chen. Interactive text-to-sql via expected information gain for disambiguation, 2025. URL https://arxiv.org/abs/2507.06467.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Irina Saparina and Mirella Lapata. AMBROSIA: A benchmark for parsing ambiguous questions into database queries. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/a4c942a8405cc910f0a833d28d2573cc-Abstract-Datasets_and_Benchmarks_Track.html.

Ivan Sekulić, Mohammad Aliannejadi, and Fabio Crestani. Evaluating mixed-initiative conversational search systems via user simulation. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, pp. 888–896. ACM, February 2022. doi: 10.1145/3488560.3498440. URL http://dx.doi.org/10.1145/3488560.3498440.

Manan Suri, Puneet Mathur, Nedim Lipka, Franck Dernoncourt, Ryan A. Rossi, and Dinesh Manocha. Structured uncertainty guided clarification for llm agents, 2026. URL https://arxiv.org/abs/2511.08798.

Yibo Wang, Nikki Lijing Kuang, Philip S. Yu, Zhewei Yao, and Yuxiong He. Learning to retrieve: Dual-level long-term memory for text-to-sql agents, 2026. URL https://arxiv.org/abs/2606.00547.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

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 (EMNLP-IJCNLP)*, pp. 5447–5458. Association for Computational Linguistics, November 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 Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 1962–1979, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1204. URL https://aclanthology.org/D19-1204/.

Fuheng Zhao, Shaleen Deep, Fotis Psallidas, Avrilia Floratou, Divyakant Agrawal, and Amr El Abbadi. Sphinteract: Resolving ambiguities in NL2SQL through user interaction. *Proc. VLDB Endow.*, 18(4):1145–1158, December 2024. ISSN 2150-8097. doi: 10.14778/3717755.3717772. URL https://doi.org/10.14778/3717755.3717772.

A Experimental Protocol

A.1 Tasks and Interaction

BIRD-Interact. We evaluate all 600 tasks in BIRD-INTERACT-FULL (Huo et al., 2026). Each method receives the ambiguous request, public schema and column meanings, and supplied background knowledge. Private ambiguity annotations and reference solutions remain hidden. The base allowance A counts the task's critical and knowledge ambiguity annotations; the harness supplies this integer without revealing the annotations. The question limit is $Q_{\max} = A + P$, so $P = 0$ retains the base allowance. Comparisons use $P \in \{0, 3, 5, 7, 9\}$, while the controlled clarification study examines all integer P from 0 to 19.

The simulated user follows BIRD-Interact's function-driven procedure and may answer or refuse a clarification. Validation protocol and results appear in Appendices B.2 and B.3. Every issued question consumes one unit, including refusals and unavailable replies. Internal SQL generation, execution, and repair do not consume question slots. Methods may submit before exhausting the allowance. When budgets share a trajectory, each submission uses only information available at its own checkpoint; larger budgets reuse a natively terminated state without adding questions.

AMBROSIA-Interact. We adapt AMBROSIA (Saparina & Lapata, 2024) using 300 distinct original questions, with 100 each for attachment, scope, and vague ambiguities, sampled with seed 42. One interpretation and its paired reference SQL are selected per question before evaluation and held fixed across methods. The methods receive the ambiguous request and public DDL schema; the selected intent and reference remain private to the direct-answer user simulator and evaluator. Here $P = Q_{\max} \in \{0, 1, 2, 3, 4, 5\}$ is the total question allowance, with no annotation-derived base allowance. Final SQL is evaluated against the selected intent on the supplied SQLite database.

Full-information reference. For each BIRD task, `deepseek-v4-flash` receives the original unambiguous request, schema and column meanings, and full unmasked external knowledge. With reasoning enabled and temperature 0.7, it generates one SQL without user interaction or multi-candidate selection. At most one execution-error repair is allowed, yielding $236/600 = 39.33\%$. This is a measured reference, not a theoretical upper bound.

A.2 Resource Accounting

$\bar{Q}(P)$ is the mean number of issued questions, including those followed by a refusal or user-service failure. A question-generation failure before issuance adds no question, and API retries do not add user questions. $\bar{T}(P)$ is the mean input plus output token usage across the model calls required by each method. It includes request analysis, question generation, candidate generation and repair, and terminal generation and repair required for the reported submission; user-simulator and hidden-evaluator calls are excluded. Cached-input and reasoning-output tokens are counted once within the provider's input and output totals.

Costs cover task completion at each budget, including stopping decisions; reused computation is counted once. Database execution costs and monetary charges are not quantified.

B Validation of `deepseek-v4-pro` as the User Simulator

B.1 Simulator Choice

Simulated users support scalable evaluation of conversational and tool-using agents (Sekulić et al., 2022; Yao et al., 2024). The original BIRD-Interact study evaluates user simulators based on `Gemini-2.0-Flash`, `GPT-4o`, and `Claude-Haiku-4.5` (Huo et al., 2026). We use `deepseek-v4-pro` and validate its ability to answer legitimate clarifications while withholding privileged solution information with USERSIM-GUARD. Our simulator follows BIRD-Interact's function-driven, two-stage procedure: select an action, then generate a response grounded in private task annotations, reference SQL, and the database schema. These inputs are not directly exposed to the evaluated methods; the response prompt prohibits disclosing SQL, solution steps, and inappropriate schema information.

Table 4: `deepseek-v4-pro` on USERSIM-GUARD v1.5 (700 items per category). Individual passes and thresholds are percentages; the strict intersection reports the count and percentage accepted by both evaluations.

Category	Pass 1	Pass 2	Strict intersection	Threshold
AMB	94.14	95.14	648/700 (92.57%)	≥ 90
LOC	97.86	96.86	674/700 (96.29%)	≥ 90
UNA	96.43	96.43	675/700 (96.43%)	≥ 80

B.2 VALIDATION WITH THE ORIGINAL EVALUATOR

USERSIM-GUARD v1.5 contains 2,100 clarification questions, with 700 in each of three categories.[1] **AMB** covers annotated ambiguities, **LOC** covers reasonable clarifications beyond those annotations, and **UNA** covers requests that should be declined, such as requests for reference SQL. Before evaluation, one LOC response was regenerated after correcting a corrupted local question to the official v1.5 text; all 2,100 items were retained.

We use `Qwen3-235B-A22B-Instruct-2507` (Qwen Team, 2025), the same evaluator model specified in Appendix O.2 of BIRD-Interact (Huo et al., 2026), with temperature 0 and a 2,048-token output limit. Each fixed response is judged twice and passes only if both calls accept it, following the original strict consistency rule. Under the released rubrics, fully appropriate or acceptable answers pass for AMB and LOC, while UNA requires an appropriate refusal. The acceptance thresholds are 90% for AMB and LOC and 80% for UNA.

B.3 VALIDATION RESULTS

Table 4 reports strict acceptance rates of 92.57% on AMB, 96.29% on LOC, and 96.43% on UNA, exceeding all three thresholds. Under the original evaluator and consistency rule, these results support the evaluated `deepseek-v4-pro` simulator on this fixed question set.

C IMPLEMENTATION DETAILS

We describe the five external methods evaluated in Sections 3.1 and 5.1. All use `deepseek-v4-flash`, with `deepseek-v4-pro` as the user simulator and reasoning enabled. Unless otherwise specified, our BIRD-Interact evaluation uses BIRD-INTERACT-FULL (600 tasks), with $Q_{\max} = A + P$ and $P \in \{0, 3, 5, 7, 9\}$; $P = 0$ retains the task-specific base allowance A. The user-simulator validation results are in Appendix B.3; the AMBROSIA-INTERACT setup and results are in Appendices A.1 and E.3.

Adaptation principles. Our adaptation policy is to retain each method's question-generation and selection strategy, candidate-update policy, and stopping logic wherever compatible with the shared SQL task and user-simulator interface. $Q_{\max}$ sets a common upper bound; methods may stop earlier under their own rules. The subsections below distinguish retained mechanisms from changes to execution evidence, available user feedback, and final SQL generation. The results characterize these documented adaptations.

Evaluation covers the first task attempt, without benchmark debug or follow-up rounds. Comparisons allow at most one terminal execution-error repair, except for Sphinteract, which retains its native current-SQL submission with in-loop repair. Method-specific generation and repair settings are given below. Task success rates and mean resource use include failed tasks. We report mean issued questions $\bar{Q}$ and mean model input plus output tokens $\bar{T}$; tokens cover the selected method path, excluding the user simulator and hidden evaluation (Appendix A.2).

Table 5 follows the candidate-pool categories in Section 3.1: no dedicated pool, a fixed initial pool, or a pool resampled round-wise. Here M denotes the initial pool size and D the number of candidates sampled per round. Sphinteract revises a single SQL without maintaining a candidate pool.

[1] Dataset release: `https://huggingface.co/datasets/birdsql/usersim-guard-v1.5`.

Table 5: Candidate pools and database execution within clarification. Final-submission repair checks and hidden correctness evaluation are excluded. ATD denotes Active Task Disambiguation.

Method	Candidate pool	Database execution
BIRD-Interact Conv	None	No
ClarifyGPT	Fixed initial ($M = 25$)	Yes
PleaSQLarify	Fixed initial ($M = 50$)	Yes
ATD	Round-wise ($D = 10$)	No
Sphinteract	None	Yes

C.1 BIRD-INTERACT CONV

BIRD-Interact Conv follows the benchmark's conversational *c-Interact* workflow (Huo et al., 2026). Given the task request, schema, and supplied knowledge, the agent asks clarification questions one at a time, updates the dialogue with each reply, and submits its SQL when ready or when $Q_{\max}$ is reached. It uses no dedicated candidate pool or database execution during clarification. We supply column meanings in the initial system prompt. The native SQL receives at most one terminal repair on an execution error or timeout, without access to correctness feedback.

C.2 CLARIFYGPT

ClarifyGPT (Mu et al., 2024) uses execution behavior within a fixed initial candidate pool to guide clarification. Our SQL adaptation replaces generated function-test inputs with the task database: Query candidates are grouped by result sets and Management candidates by database-state changes. Passing the execution-consistency check permits direct submission; otherwise, at most five group representatives guide a batch of questions. The questions are asked in order within $Q_{\max}$, without regenerating the pool or question batch. Unanswerable replies consume budget but are omitted from the final prompt. After clarification, one final SQL is generated from the task and retained answers.

We sample $M = 25$ candidates at temperature 0.8 and use temperature 0 for question and final SQL generation, with nucleus sampling ($p = 0.95$). The question prompt retains three original code-based clarification demonstrations. The final SQL may receive at most one terminal execution-error repair, without correctness feedback or changes to the clarification procedure.

C.3 PLEASQLARIFY

PleaSQLarify (Chan et al., 2026) progressively filters a fixed initial pool of $M = 50$ SQL candidates. SQL syntax features and execution-output similarity define hypotheses and functional groups; expected information gain selects the next binary decision. We phrase this decision as a constrained Yes/No question in task-domain language. Each reply filters the hypotheses and updates their posterior and question rankings, without adding new candidates. Clarification ends when the remaining pool is functionally homogeneous, contains at most one hypothesis, a reply cannot select a binary branch, or $Q_{\max}$ is reached.

The original method selects a representative from the surviving pool. Our evaluated adaptation generates one final SQL from the public task context and available dialogue, including refusals and non-binary replies, without passing SQL candidates or scores to this step. An empty executable pool ends clarification but still permits terminal generation. Final SQL generation and repair use temperature 0; we allow at most one terminal execution-error repair without correctness feedback.

C.4 ACTIVE TASK DISAMBIGUATION (ATD)

Active Task Disambiguation (ATD) (Kobalczyk et al., 2025), in its binary code-generation variant, uses execution to construct a Yes/No matrix over candidate programs and input–output assertions. Our SQL adaptation uses SQL candidates as rows and natural-language binary intent questions as columns. Questions are proposed from the task context and dialogue, independently of the candidate pool. For each candidate, the model predicts the implied Yes/No answers from its SQL text, treating it as the intended solution. Under a uniform candidate prior, the question with the highest expected information gain is asked. The reply, including refusals, enters the dialogue; the next round regener-

ates the pool and question matrix. Neither zero information gain nor an unanswerable reply triggers early stopping before $Q_{\max}$.

Each round samples $D = 10$ SQL candidates and proposes five questions, of which one is asked. Candidates are not executed or refilled; an incomplete parsed pool is recorded as a method failure. SQL sampling uses temperature 1 and nucleus sampling ($p = 0.95$); answer prediction uses temperature 0. At each evaluated dialogue prefix, one final SQL is generated from the public task context and question–reply pairs. Final generation and repair use temperature 0; we allow at most one terminal execution-error repair, excluding timeouts and parse errors. Terminal generation and repair do not affect subsequent clarification, and no correctness feedback is provided.

C.5 SPHINTERACT

Sphinteract (Zhao et al., 2024) maintains and revises one current SQL without sampling a candidate pool. Its Summarize, Review, and Ask (SRA) prompt reviews SQL and dialogue histories to propose a multiple-choice clarification question. Each reply, including an unanswerable response, enters the history and conditions a SQL revision. Each generated SQL is executed, with at most one error-guided repair if execution fails. We use CQs&ES (clarification questions with early stopping), which submits the current SQL when SRA detects no remaining ambiguity or $Q_{\max}$ is reached. This retains native in-loop repair, without additional terminal SQL generation or repair.

Our adaptation omits the original user-acceptance check because the simulator only answers clarification questions, and replaces the four-interaction limit with $Q_{\max}$. For Management tasks, clarification targets the intended operation, database object, expected effect, and literal values. Query clarification uses the eight released SRA demonstrations; Management clarification and SQL generation use no demonstrations. All stages use temperature 0, and no correctness feedback is provided.

D DUETSQL IMPLEMENTATION DETAILS

Algorithm 1 summarizes DuetSQL's control flow. For the main comparisons, Output Specification was evaluated on saved dialogue prefixes; those skipping this stage retain their recorded submissions. Section 5.3 describes the earlier terminal-generation variant used in the stopping comparison.

Reported settings. We use $H = 5$, $D = 8$, and $\theta = 0.8$. Candidates are executed from the same isolated database state, with at most one execution-error repair per candidate. All executable candidates are shown for clarification; grouping determines the stopping statistic.

Algorithm conventions. All generation uses the public schema, knowledge, and current dialogue. $\textsc{Ask}(q_t)$ returns the user reply $r_t = \mathcal{U}(q_t)$; it does not update the dialogue or counters. These updates are explicit below: $\oplus$ appends a question–reply pair, and every issued question consumes one unit of budget, including refusals. $\textsc{GenerateSQL}(\mathcal{H})$ generates one SQL from the dialogue. $\textsc{Submit}(s)$ returns the final SQL after at most one execution-error repair and ends the task. It reuses execution and repair records for unchanged SQL, without using hidden correctness feedback.

Pools are sampled at temperature 0.7; other generation calls use temperature 0. For an answerable reply, Output Specification uses one model call to retain or revise s_{base}, preserving established requirements. If no base SQL was parsed, this call instead generates one SQL from the updated dialogue. A refused or unavailable reply leaves the base unchanged. Unchanged SQL reuses its existing execution and repair records.

Algorithm 1 DuetSQL control flow

Require: $u, \mathcal{S}, \mathcal{K}, \mathcal{B}, \mathcal{U}, Q_{\max}; H, D, \theta$
Ensure: Final SQL $\hat{s}$

$t \leftarrow 1$; $Q_1 \leftarrow 0$; $\mathcal{H}_0 \leftarrow (u)$
if $Q_{\max} = 0$ **then**
  $\hat{s} \leftarrow$ GENERATESQL($\mathcal{H}_0$); **return** SUBMIT($\hat{s}$)
end if
Stage 1: Request Clarification
$\mathcal{I} \leftarrow f_{\text{req}}(u; \mathcal{S}, \mathcal{K})$
for each of the first $\min(H, |\mathcal{I}|)$ targets i_j **do**
  $q_t \leftarrow g_{\text{req}}(i_j; \mathcal{S}, \mathcal{K}, \mathcal{H}_{t-1})$; $r_t \leftarrow$ ASK(q_t)
  $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \oplus (q_t, r_t)$; $Q_{t+1} \leftarrow Q_t + 1$; $t \leftarrow t + 1$
  if $Q_t = Q_{\max}$ **then**
    $\hat{s} \leftarrow$ GENERATESQL($\mathcal{H}_{t-1}$); **return** SUBMIT($\hat{s}$)
  end if
end for
Stage 2: SQL-Contrast-Guided Loop
while $Q_t < Q_{\max} - 1$ **do**
  Sample D SQL candidates $s_{t,1}, \ldots, s_{t,D}$ from $\mathcal{H}_{t-1}$; $s_{\text{base}} \leftarrow s_{t,1}$
  Execute each candidate; repair execution errors at most once; retain executable evidence $\mathcal{C}_t$
  if $\mathcal{C}_t = \emptyset$ **then return** SUBMIT(s_{base})
  Group $\mathcal{C}_t$ by query results or database effects
  $\rho_t \leftarrow$ largest group size divided by $|\mathcal{C}_t|$
  if $\rho_t \geq \theta$ **then break** ▷ Proceed to Stage 3
  $q_t \leftarrow g_{\text{ctr}}(\mathcal{C}_t; \mathcal{S}, \mathcal{K}, \mathcal{H}_{t-1})$; $r_t \leftarrow$ ASK(q_t)
  $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \oplus (q_t, r_t)$; $Q_{t+1} \leftarrow Q_t + 1$; $t \leftarrow t + 1$
end while
Stage 3: Output Specification
if $Q_t = Q_{\max} - 1$ **then** $s_{\text{base}} \leftarrow$ GENERATESQL($\mathcal{H}_{t-1}$)
$q_t \leftarrow q^{\text{out}}$; $r_t \leftarrow$ ASK(q_t)
$\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \oplus (q_t, r_t)$; $Q_{t+1} \leftarrow Q_t + 1$
$\hat{s} \leftarrow s_{\text{base}}$ ▷ Keep the base if the reply is unanswerable
if r_t is answerable **then**
  $\hat{s} \leftarrow$ retain or minimally revise s_{base} using $\mathcal{H}_t$
end if
return SUBMIT($\hat{s}$)

D.1 Execution Evidence and Grouping

Each executable candidate in $\mathcal{C}_t$ is paired with an observation obtained from its own copy of the same initial task database. The clarification model receives the SQL and a compact observation preview, as illustrated in Figure 8. Grouping uses the full execution records described below.

Query. The result preview contains the total numbers of rows and columns, column names, and up to the first 20 rows. Candidates are grouped by a signature computed over the column names and all returned rows in their observed order; the preview limit does not truncate this signature. Matching signatures place candidates in the same group, whose size contributes to the agreement statistic ρ_t.

Management. We compare database state before and after execution. The effect preview reports whether any change occurred and lists created, dropped, or modified objects by type, such as tables, views, functions, and triggers. The underlying record additionally captures object identities, interfaces (e.g., view columns or function arguments), and definitions. For affected tables, views, and materialized views, it includes row counts and order-independent hashes of their contents, allowing data changes to be distinguished even when object definitions remain unchanged.

Management grouping hashes the list of changes together with the detailed object records and data summaries. Matching signatures form a group; successful candidates with no observed change share a group. These finest state-effect groups determine the stopping statistic. For example, two candidates that create the same view with the same columns but use `total > 100` and `total >= 100` have matching object identities and interfaces but different definitions. They therefore belong to different groups even if their current result rows coincide.

Query evidence

```
Candidate 1:
SELECT order_id FROM orders
WHERE total >= 100;
Result Preview 1:
{"n_rows": 2, "n_cols": 1,
 "col_names": ["order_id"],
 "first_rows": [[7], [12]],
 "truncated": false}
```

Management evidence

```
Candidate 1:
CREATE VIEW eligible AS
SELECT order_id FROM orders
WHERE total >= 100;
Effect Preview 1:
{"effect_status": "effect",
 "diff_preview": {"views": {
   "created": ["public.eligible"],
   "dropped": [], "modified": []}}}
```

Figure 8: Illustrative execution evidence supplied with each SQL candidate. JSON is formatted for readability; unchanged object types are omitted from the Management example. These previews are separate from the grouping signatures.

For both task types, ρ_t is the largest group size divided by $|\mathcal{C}_t|$. All executable candidates remain available to the clarification model; group membership only determines this stopping statistic. Agreement concerns the recorded execution on the task database and does not establish semantic equivalence on every possible database.

E Additional Experimental Results

Budget-response summaries. The task success rate curve is the primary evidence for a method's response to the additional allowance P. On a fixed budget grid with $P = 0$ and largest value $P_{\max}$, we report three descriptive summaries:

$$\text{Initial} = \text{SR}(0), \qquad \text{Gain} = \text{SR}(P_{\max}) - \text{SR}(0), \qquad \text{Peak} = \max_{P \in \mathcal{P}} \text{SR}(P). \tag{4}$$

Success rate is measured in percent on the fixed evaluation set. Gain is the net change across the stated range, while Peak is the best observed dataset success rate on that grid; neither is a per-task selection across checkpoints. The full curve remains necessary because it exposes plateaus and regressions that these summaries omit.

E.1 Controlled Clarification Settings and Results

Table 6 reports the results for Section 3.2 on all 600 BIRD-INTERACT-FULL tasks, using `deepseek-v4-flash` as the agent and `deepseek-v4-pro` as the user simulator. The question generator receives the public task context and dialogue, supplemented by no SQL ($D = 0$), one SQL ($D = 1$), or all D sampled SQLs ($D = 2, 4, 8, 16$).

These controls omit Request Clarification and Output Specification and continue to $Q_{\max} = A + P$ unless interrupted by failure. Candidates are shown without diagnostic execution, pool repair, or grouping. At each evaluated budget, the first sampled SQL is submitted for $D \geq 1$; Context-only generates a separate SQL that is not shown to the question generator. At most one terminal execution-error repair is allowed, and terminal feedback does not enter subsequent clarification.

We report $P \in \{0, 3, 7, 11, 15, 19\}$, matching Figure 3. Each width uses one sampled trajectory without repeated seeds; comparisons are descriptive and do not establish statistical significance.

E.2 BIRD-INTERACT-FULL Results

Table 7 reports task success rates, mean questions, and mean model tokens for the six methods across $P \in \{0, 3, 5, 7, 9\}$, supplementing the main $P = 5$ comparison. Supplemental DuetSQL measurements at $P = 1, 2$ achieve 35.00% and 38.50% success, using 0.212M and 0.325M model tokens and averaging 3.752 and 4.607 questions, respectively. They refine the resource frontiers without changing the main budget grid.

Table 6: Task success rates (%) for the controlled clarification conditions on BIRD-INTERACT-FULL (600 tasks). $D = 0$ denotes Context-only, $D = 1$ Single SQL, and $D \geq 2$ SQL contrasts.

P	$D = 0$	$D = 1$	$D = 2$	$D = 4$	$D = 8$	$D = 16$
0	27.83	25.33	27.83	28.67	28.67	27.17
3	31.50	30.50	33.33	32.50	37.17	37.00
7	34.00	34.17	37.33	37.67	39.33	42.17
11	36.17	34.17	41.33	40.00	41.67	44.17
15	38.83	38.50	42.83	42.50	42.83	44.17
19	38.00	37.83	43.83	43.00	44.17	44.67

Table 7: Full budget results on BIRD-INTERACT-FULL (600 tasks).

Method	$P = 0$	$P = 3$	$P = 5$	$P = 7$	$P = 9$
Task success rate (%)					
BIRD-Interact Conv	26.50	27.33	27.17	27.33	29.33
Sphinteract	27.33	28.67	28.83	29.00	29.00
ClarifyGPT	26.00	31.50	31.33	31.50	31.50
PleaSQLarify	18.50	19.00	18.83	18.83	18.83
ATD	21.83	27.33	27.50	27.17	27.17
DuetSQL	30.83	40.50	42.33	43.83	45.67
Mean questions $\bar{Q}$					
BIRD-Interact Conv	1.672	1.920	1.988	1.950	1.933
Sphinteract	1.772	2.150	2.200	2.218	2.227
ClarifyGPT	2.622	3.755	3.817	3.827	3.833
PleaSQLarify	1.062	1.242	1.253	1.253	1.253
ATD	2.867	5.738	7.605	9.448	11.262
DuetSQL	2.848	5.393	6.788	8.047	9.193
Mean model tokens $\bar{T}$ (million)					
BIRD-Interact Conv	0.053	0.057	0.058	0.058	0.058
Sphinteract	0.069	0.083	0.086	0.087	0.087
ClarifyGPT	0.442	0.442	0.442	0.442	0.442
PleaSQLarify	0.873	0.878	0.878	0.878	0.878
ATD	1.045	2.083	2.763	3.441	4.114
DuetSQL	0.131	0.459	0.741	1.003	1.244

E.3 AMBROSIA-INTERACT BUDGET GRID

Table 8 reports all total-question limits on the fixed 300-task subset described in Appendix A.1. Evaluation follows the external-method submission rules in Appendices C.1–C.5 and DuetSQL's control flow in Algorithm 1. DuetSQL includes Output Specification and agreement stopping at $\theta = 0.8$. Each limit retains the same 300 tasks, and resource means include the work required by the selected submission, including reused prefixes and applicable repairs. Other budgets' terminal readouts are excluded. BIRD-Interact Conv runs independently at each limit because the limit enters its native prompt; the other methods reuse causal prefixes while retaining native stopping. No outcome is selected using reference correctness.

DuetSQL achieves the highest success for $Q_{\max} \in \{2, 3, 4, 5\}$. At $Q_{\max} = 3$, it reaches 85.67%, compared with 80.67% for ATD, while using 86.8% fewer tokens. At $Q_{\max} = 5$, DuetSQL reaches 87.67% versus ATD's 87.33%, while using 88.1% fewer tokens and 43.1% fewer issued questions.

E.4 INITIAL-POOL DIAGNOSTICS

Table 9 summarizes ClarifyGPT's fixed initial pools, measured once per task. Execution pass rate is the fraction of sampled slots whose SQL executes successfully; executable-candidate and successful behavior-group counts describe the resulting pool size and diversity. These execution diagnostics do not establish semantic correctness.

Table 8: Success and resource use across all total-question budgets on the 300-task AMBROSIA-INTERACT subset. Token counts are in thousands (k).

Task success (%) Method	$Q_{\max}=0$	$Q_{\max}=1$	$Q_{\max}=2$	$Q_{\max}=3$	$Q_{\max}=4$	$Q_{\max}=5$
BIRD-Interact Conv	16.67	32.67	38.33	38.67	37.67	37.67
Sphinteract	19.33	53.67	61.33	61.67	61.67	61.67
ClarifyGPT	19.33	59.33	70.33	74.00	73.67	74.00
PleaSQLarify	17.67	45.00	49.00	51.00	51.00	51.00
ATD	18.67	51.33	68.33	80.67	86.67	87.33
DuetSQL	17.67	58.33	77.00	85.67	87.00	87.67
Issued questions per task Method	$Q_{\max}=0$	$Q_{\max}=1$	$Q_{\max}=2$	$Q_{\max}=3$	$Q_{\max}=4$	$Q_{\max}=5$
BIRD-Interact Conv	0.000	0.373	0.457	0.487	0.460	0.487
Sphinteract	0.000	0.733	0.920	0.933	0.933	0.933
ClarifyGPT	0.000	0.920	1.833	2.530	2.747	2.780
PleaSQLarify	0.000	0.650	0.940	1.093	1.183	1.240
ATD	0.000	0.997	1.993	2.990	3.987	4.983
DuetSQL	0.000	1.000	1.907	2.490	2.753	2.837
Tokens per task (k) Method	$Q_{\max}=0$	$Q_{\max}=1$	$Q_{\max}=2$	$Q_{\max}=3$	$Q_{\max}=4$	$Q_{\max}=5$
BIRD-Interact Conv	1.7	2.8	3.1	3.2	3.2	3.2
Sphinteract	1.3	6.5	9.6	10.3	10.4	10.4
ClarifyGPT	36.6	36.6	36.6	36.7	36.7	36.7
PleaSQLarify	81.1	83.1	84.4	85.0	85.3	85.4
ATD	1.3	37.0	73.6	111.8	151.0	191.8
DuetSQL	1.4	4.6	8.0	14.7	20.1	22.8

Table 9: ClarifyGPT initial-pool diagnostics ($M = 25$), computed once for the shared pools across all P checkpoints. Diagnostics use tasks with complete pool records; one failed Query task has no complete pool record and is excluded here. Final success rates in Table 1 retain all 600 tasks.

Metric	Overall	Query	Management
Tasks with recorded pools	599	409	190
Sampled candidate slots	14,975	10,225	4,750
Execution pass rate (%)	70.64	69.59	72.91
Executable candidates per task (mean)	17.66	17.40	18.23
Distinct successful behavior groups (mean)	9.85	9.16	11.34
Tasks with no executable candidates	22	14	8

E.5 DUETSQL COMPONENT RESULTS ACROSS INTERACTION BUDGETS

Table 10 reports the full success-rate results for the component analysis in Section 5.2. Each variant removes the named component from the current DuetSQL configuration. The loop uses $D = 8$ and early stopping at $\theta = 0.8$ when retained; Request Clarification uses up to $H = 5$ questions when enabled. Without the SQL-Contrast-Guided Loop, the same completed Request Clarification prefix proceeds directly to one base SQL and Output Specification. This removes candidate-pool sampling, execution grouping, agreement decisions, and candidate-guided questioning together. It retains the original allowance and uses only the Request Clarification dialogue, without subsequent loop answers. Without Request Clarification, the loop starts from the original request and retains $D = 8$, $\theta = 0.8$, and the current Output Specification; no request-clarification answers are inherited. All results use the same 600 tasks and per-task allowance $A+P$. Table 11 provides the corresponding question and token use; the main component table averages each metric equally over the five budgets.

Table 10: Task success rates (%) for DuetSQL component ablations on BIRD-INTERACT-FULL.

Variant	$P = 0$	$P = 3$	$P = 5$	$P = 7$	$P = 9$
DuetSQL	30.83	40.50	42.33	43.83	45.67
w/o SQL-Contrast-Guided Loop	30.33	35.33	35.67	35.67	35.67
w/o Request Clarification	34.50	39.17	40.67	40.67	42.50
w/o Output Specification	28.67	34.00	37.67	39.50	40.33

Table 11: Resource use of DuetSQL component ablations across budgets. $\bar{T}$ is mean tokens in millions and $\bar{Q}$ is mean issued questions.

Variant	$P = 0$	$P = 3$	$P = 5$	$P = 7$	$P = 9$
Mean tokens $\bar{T}$ (million)					
DuetSQL	0.131	0.459	0.741	1.003	1.244
w/o SQL-Contrast-Guided Loop	0.075	0.090	0.090	0.090	0.090
w/o Request Clarification	0.372	0.828	1.095	1.346	1.579
w/o Output Specification	0.198	0.591	0.863	1.114	1.346
Mean issued questions $\bar{Q}$					
DuetSQL	2.848	5.393	6.788	8.047	9.193
w/o SQL-Contrast-Guided Loop	2.590	3.650	3.703	3.703	3.703
w/o Request Clarification	2.658	4.900	6.208	7.410	8.517
w/o Output Specification	2.758	5.122	6.448	7.648	8.753