

When the Bottleneck Shifts: Diagnosing and Closing Information Gaps in Interactive Text-to-SQL

Mengyi Yan¹, Zhinian Ma¹, Yue Wang^{2*}, Yaoshu Wang²,
Xiaoke Zhu², Zizhe Wang³, Haoyi Zhou^{4*}, Zhumin Chen¹

¹School of Artificial Intelligence, Shandong University

²Shenzhen Institute of Computing Sciences

³Astroup Lab ⁴School of Software, Beihang University

{yanmy,chenzhumin}@sdu.edu.cn, 202635963@mail.sdu.edu.cn, ywangby@connect.ust.hk

{zhuxk,yaoshuw}@sics.ac.cn, buaawzz@hotmail.com, haoyi@buaa.edu.cn

Abstract

In interactive text-to-SQL, the evidence needed to write correct SQL is not given upfront but distributed across schema, knowledge bases, and users, and must be acquired through costly environment interactions under a finite budget. We observe that strong LLMs often fail not from deficient SQL synthesis, but from premature commitment to SQL before acquiring task-critical evidence. Through controlled trajectory analysis, we formalize this observation as the Information Sufficiency Hypothesis and identify a five-class gap taxonomy that structures the acquisition space. Guided by this diagnosis, we develop PASCAL, a training-free acquisition-control data agent framework that interposes a source-aware information-acquisition layer before SQL generation and schedules acquisition by cost-normalized priority. On BIRD-Interact, PASCAL with a local 35B backbone model achieves 51.0% Phase-1 and 37.0% two-phase success rate, outperforming frontier models leaderboard result by up to +14.7 pp, with gains concentrated on BI tasks requiring cross-source evidence acquisition. Code Repository at¹.

1 Introduction

Real-world text-to-SQL interaction requires more than writing correct SQL. Users issue ambiguous analytical queries whose meaning depends on business context, domain knowledge, and unstated assumptions (Bhaskar et al., 2023; Murzaku et al., 2025). The semantics needed to resolve these ambiguities often reside outside the database schema, in knowledge bases, documentation, the user’s own intent, or prior-phase results (Li et al., 2023; Chen et al., 2025b; Yu et al., 2019a; Huo et al., 2026).

*Corresponding authors.

¹https://github.com/authorlord/PASCAL_Release

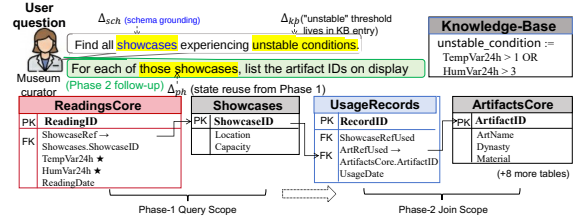


Figure 1: An interactive text-to-SQL task with three co-occurring information gaps from distinct environment sources. Each must be closed before SQL generation can begin.

In such settings, successful SQL generation depends on *information acquisition*: deciding what evidence is missing and obtaining it from the right source before composing a query. Figure 1 shows a representative task. A museum curator asks “*find all showcases experiencing unstable conditions*” and follows up with “*for each of those showcases, list the artifact IDs on display.*” Three information gaps, grounding *showcases* to *ShowcaseRef*, a KB definition (*unstable-condition*), and a prior reference (*those showcases*), must each be closed from a distinct source before SQL generation can begin. Existing text-to-SQL methods (Pourreza and Rafiei, 2023; Wang et al., 2025; Pourreza et al., 2025) improve SQL quality through decomposition, self-correction, or model adaptation, but operate under a static input paradigm: question, schema, and auxiliary knowledge are prepared before generation, leaving no room to decide what evidence is still missing at runtime.

Interactive text-to-SQL extends this setting by allowing missing information to be acquired by a data agent through sequential environment interactions (Luo et al., 2026). Conversational benchmarks (Yu et al., 2019a,b) introduce multi-turn context but still provide dialogue history as static input. BIRD-Interact (Huo et al., 2026) goes further: the agent can inspect schema, retrieve from a hierar-

chical KB, ask the user, execute SQL, and submit. Its official a-Interact scaffold exposes these actions via ReAct (Yao et al., 2023) but leaves their ordering to autonomous tool use, leading to overuse of SQL retry (*overthinking*) or tool calls that miss the task-relevant gap (*undirected exploration*); even state-of-the-art models such as Claude-Opus-4.6 achieve only $\sim 36\%$ success on BIRD-Interact.

The key bottleneck is interaction cost. Unlike code or math agents (Yang et al., 2024; Gou et al., 2024) in sandboxed environments, database agents touch production resources: SQL execution loads databases, KB retrieval hits enterprise systems, and user clarification consumes real attention (Chen et al., 2025b). Environment interactions, not inference tokens, are the scarce resource. We therefore formalize interactive text-to-SQL as a *finite-budget acquisition* problem: acquire task-relevant evidence within B interaction budget before committing to SQL. Controlled trajectory diagnosis (Section 3) reveals a recurring failure pattern: agents commit to SQL before acquiring task-critical evidence. We formalize this as the *Information Sufficiency Hypothesis* (ISH; Section 4.1).

Based on this diagnosis, we propose PASCAL, a training-free acquisition-control framework for interactive text-to-SQL agents. PASCAL schedules actions by cost-normalized priority over three layers (Fig. 2): proactive acquisition (schema grounding, KB retrieval), active clarification, and harness control (gating SQL submission, bounding retry). On BIRD-Interact, PASCAL with a local 35B backbone achieves 51.0% phase-1 and 37.0% two-phase success rate on query/follow-up questions over 300 tasks, exceeding the strongest baseline by +14.7/+13.0 pp. Gains trace to budget allocation, not model scale; additionally, the data agent runs on-premise, keeping production data local.

Contributions. (i) We formalize interactive text-to-SQL as a finite-budget acquisition problem (Section 2). (ii) Through controlled trajectory analysis, we identify premature SQL commitment as the dominant failure mode, formalize it as ISH, and derive a five-source gap taxonomy (Section 3). (iii) Motivated by the observation, PASCAL uses cost-normalized priority to coordinate proactive acquisition, active clarification, and harness control (Section 4). (iv) On the widely-adopted BIRD-Interact benchmark, we show that PASCAL with a 35B backbone reaches the strongest reported result in our BIRD-Interact evaluation and trace the

gains to acquisition routing via ablations (Section 5).

2 Problem Definition

We formalize interactive text-to-SQL as budgeted information acquisition: the agent operates in a multi-source environment under a finite interaction budget, where each environment action (schema exploration, KB retrieval, user clarification, SQL execution, or submission) consumes part of the budget, and the required information can be decomposed into source-specific gap classes whose closure costs differ.

2.1 Interactive text-to-SQL as budgeted acquisition

A system agent S_θ and a user U operate over a shared environment $\mathcal{E} = (\mathcal{D}, \mathcal{M}, \mathcal{K}, \mathcal{U}, \mathcal{H}_0)$, where $\mathcal{D}$ is a relational database with an execution engine, $\mathcal{M}$ is schema metadata, $\mathcal{K}$ is an external knowledge base of domain definitions and formulas, $\mathcal{U}$ is the user, and $\mathcal{H}_0$ is the prior-phase state carried from completed phases (set once at phase start; $\mathcal{H}_0 = \emptyset$ when no prior phase exists). $\mathcal{E}$ is fixed within each task phase; the quantities that evolve during interaction are the agent’s acquired information and remaining budget, defined below.

Given a query q , S_θ sequentially issues actions $a_t \in \mathcal{A}$ —schema exploration, KB retrieval, user clarification, SQL execution, or submission—observing responses o_t and maintaining a history $h_t = (q, a_1, o_1, \dots, a_t, o_t)$. Each non-terminal action returns information items $\mathcal{I}_{t+1} = \mathcal{I}_t \cup g(a_t, o_t)$, where $g(a_t, o_t) \subseteq \mathcal{I}$ are the items revealed by o_t . Each action incurs a material cost $c(a_t)$ (database load, KB access, or user attention; list in Table 38). Budget B limits the interaction:

$$\sum_t c(a_t) \leq B \quad (1)$$

The agent must find a policy $\pi: \mathcal{H} \rightarrow \mathcal{A}$ that maximizes correct-SQL probability under budget constraint:

$$\begin{aligned} \max_{\pi} & ; P(\text{exec}(\sigma_\pi, \mathcal{D}) = \text{exec}(\sigma^*(q), \mathcal{D})) \\ \text{s.t.} & ; \sum_t c(a_t) \leq B, \quad a_t = \pi(h_{t-1}) \end{aligned}$$

where σ_π is the SQL submitted under π and $\sigma^*(q)$ is the ground truth for task q . Evaluation: a task passes *P1* if the Phase-1 execution result matches the gold, and passes *P2* if both phases match.

Gap class	Source	Example (Continue with Fig. 1)
Δ_{sch}	schema $\mathcal{M}$	<i>cases</i> has no table match; resolved to ShowcaseRef via exploration
Δ_{kb}	KB $\mathcal{K}$	<i>unstable</i> requires threshold from KB entry “Instability Criteria”
Δ_{usr}	user $\mathcal{U}$	<i>recent</i> requires further user-specific; no schema or KB definition exists
Δ_{exe}	execution $\mathcal{D}$	wrong row count reveals a missing naturalJOIN condition
Δ_{ph}	prior state $\mathcal{H}_0$	<i>those showcases</i> references the Phase-1 result set

Table 1: Gap taxonomy. Each required item has exactly one canonical source, which determines its closure mechanism.

2.2 Information gap and gap taxonomy

Definition 1 (Required information set). *For a task q with gold solution $\sigma^*(q)$, the required set $\mathcal{I}^*(q) \subseteq \mathcal{I}$ is the minimal subset of information items whose absence makes $\sigma^*(q)$ under-determined.*

Definition 2 (Information gap). *At turn t , let $\mathcal{I}_t \subseteq \mathcal{I}$ denote the items the agent has acquired. The information gap is $\Delta_t(q) = \mathcal{I}^*(q) \setminus \mathcal{I}_t$.*

We decompose information gaps by canonical source: the component of $\mathcal{E}$ that stores each item (Table 1). Because each source exposes a different access protocol (schema exploration, KB retrieval, user clarification), closing gaps from different sources requires different mechanisms. A survey of ten text-to-SQL benchmarks (Appendix C, Table 9) shows that the required acquisition channels grow as environments become richer; the five classes in Table 1 represent the union across these benchmarks.

Each information item has a unique canonical source under $\mathcal{E}$, so the five classes in Table 1 form a partition of $\Delta_t(q)$ (by construction of $\mathcal{E}$). Closure costs are heterogeneous ($\text{cost}(\Delta_{\text{sch}}) < \text{cost}(\Delta_{\text{kb}}) < \text{cost}(\Delta_{\text{usr}})$), so under finite budget the acquisition order is consequential: low-cost structural channels should be exhausted before committing budget to expensive sources such as user clarification.

3 Diagnosing the Acquisition Bottleneck

To diagnose whether the bottleneck lies in SQL generation or in information acquisition, we hold the model fixed and incrementally open acquisition channels, measuring how budget allocation and failure modes shift across configurations.

3.1 Setup

We use BIRD-Interact (Huo et al., 2026) and sample 100 tasks from `lite_300`, stratified by difficulty.

Config	P1	BI	DM	Gen%	Acq%	KB%	InfoF%	GenF%
U (undirected)	22.7	10.8	44.8	49.1	50.9	31.6	75.0	2.3
B (baseline)	38.0	28.2	56.2	93.5	6.5	0.0	62.0	0.0
B+S	44.3	36.9	58.1	81.7	12.2	0.0	55.7	0.0
B+S+K	57.3	53.3	64.8	66.2	25.1	13.2	29.9	12.8

Table 2: Diagnostic comparison.

All four configurations use the same 35B backbone (Qwen3.6-35B); they differ only in which gap classes (Section 2.2) the agent is allowed to access. **U (undirected)**: the official REACT agent (Yao et al., 2023; Huo et al., 2026) with all nine tools exposed and no routing; all five gap classes are reachable in principle. **B (baseline)**: Δ_{kb} closed (KB retrieval disabled); the agent can only explore schema (Δ_{sch}), execute SQL (Δ_{exe}), and clarify with the user (Δ_{usr}). **B+S**: B with Δ_{sch} partially pre-resolved (schema metadata injected at task start); Δ_{kb} still closed. **B+S+K**: B+S with Δ_{kb} opened (namespace listing plus typed definition retrieval). The progression $B \rightarrow B+S \rightarrow B+S+K$ incrementally opens one gap class at a time while holding model capability constant. U runs on the benchmark’s official a-Interact scaffold; the other three share the same prompt infrastructure.

Besides P1 (Section 2.1), we track how the agent spends its interaction budget (Gen%, Acq%, KB%; Table 2) and where failures originate. **InfoFail%** counts tasks that fail because the agent never acquired a gold-annotated KB entry or never resolved a critical ambiguity; for this Phase-1 diagnostic subset, it focuses on Δ_{kb} and Δ_{usr} gaps. **GenFail%** counts tasks that fail despite having acquired sufficient information. Both are computed from the benchmark’s gold annotations.

3.2 Observations

(1) Budget allocation, rather than acquisition volume alone, explains the performance differences. The four configurations span a wide range of budget allocations (Table 2). B devotes 93.5% of its actions to SQL generation and retry, leaving only 6.5% for information acquisition (P1 = 38.0%). U (REACT) allocates over half its budget to acquisition (Acq% = 50.9), yet scores the lowest P1 of all four (22.7%): its KB queries lack namespace guidance, and the context fills with irrelevant definitions. The model is identical in both cases; the difference is entirely in the interaction strategy. The controlled progression $B \rightarrow B+S \rightarrow B+S+K$ shows that P1 improves monotonically as acquisition share grows *in a structured way*: 38.0% $\rightarrow$ 44.3% $\rightarrow$ 57.3%. Acquisition volume alone is insufficient when tool

calls are not conditioned on the currently unresolved gap.

(2) Targeted domain retrieval produces the largest gain. On both B and B+S the agent never invokes `get_knowledge_definition` ($KB\% = 0$). Adding schema grounding ($B \rightarrow B+S$) improves P1 by +6.3 pp; opening the KB channel ($B+S \rightarrow B+S+K$) adds +13.0 pp, more than twice the gain of all other additions combined. In contrast, U invokes KB retrieval far more ($KB\% = 31.6$; Obs. 1) yet achieves the lowest P1: the retrieved definitions rarely match the task-relevant gap. We observe the bottleneck is domain semantics—thresholds, formulas, and business rules that reside outside the schema (in production, typically in separate ontology layers). The progression opens channels in ascending cost order, confirming that cheap structural channels should be exhausted before the costlier semantic layer.

(3) The model can write SQL; the bottleneck is premature commitment. The last two columns of Table 2 decompose failures into two categories: *information failures* (InfoFail%), where the agent commits to SQL before acquiring necessary evidence, and *generation failures* (GenFail%), where the agent has sufficient evidence but produces incorrect SQL. On B and B+S, $GenFail\% = 0.0$: every failure traces to missing KB entries or unresolved ambiguities, not to SQL quality. This does not mean that sufficient information guarantees correct SQL; it means that information acquisition is the prior bottleneck—generation quality cannot even be evaluated until the necessary evidence is in place. Only after the KB channel opens does $GenFail\%$ rise above zero for the first time (12.8% on B+S+K), confirming that generation failures were masked, not absent: once information gaps close, SQL quality becomes the next limiting factor. While task-critical information remains missing, additional SQL retries spend budget without addressing the binding gap.

3.3 Design Implications

The above observations establish that the agent needs runtime acquisition control rather than more tools or more SQL retries. Obs. 1 rules out undirected tool use; Obs. 2 identifies KB access (Δ_{kb} , Section 2.2) as the highest-leverage missing source; and Obs. 3 shows that the bottleneck lies in the interaction strategy—when and where the agent acquires information—rather than in its SQL genera-

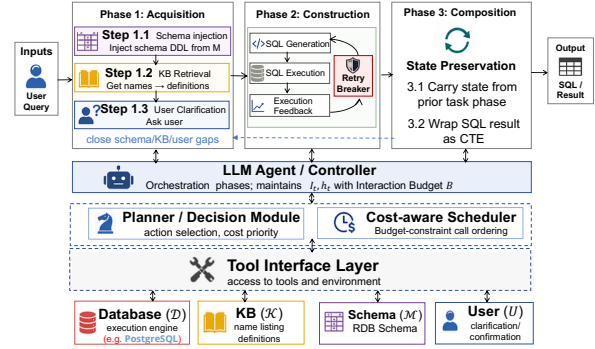


Figure 2: PASCAL architecture.

tion ability. Section 4.1 combines the acquisition-first priority established by Obs. 1–3 with the heterogeneous cost structure of the gap taxonomy to derive a cost-normalized acquisition schedule that guides the PASCAL protocol design.

4 The PASCAL Protocol

The observations of Section 3 show that the bottleneck in interactive text-to-SQL is not SQL generation but the interaction strategy that governs when and where the agent acquires information. PASCAL addresses this with a training-free, three-phase protocol (Figure 2): Phase 1 closes schema, KB, and user gaps before SQL generation; Phase 2 generates SQL with a retry breaker that redirects to acquisition on stagnation; Phase 3 carries acquired state into follow-up questions. Rather than modifying the model or learning a routing policy, PASCAL uses a closed-form priority rule grounded in the gap taxonomy (Section 2.2) so that every acquisition decision is traceable to a specific gap class and cost estimate. This section first derives the acquisition schedule from the diagnostic findings (Section 4.1), then details the gap-closing mechanisms it induces (Section 4.2–4.3). Pseudocode in Appendix B.

4.1 Diagnostic grounding and acquisition schedule

Section 3 suggests that many failures arise before SQL generation, when the agent has not yet acquired task-critical evidence. We refer to this pattern as the *Information Sufficiency Hypothesis* (ISH), which motivates an acquisition-first strategy: the protocol should prioritize closing information gaps before committing budget to SQL generation. We measure acquisition coverage with a post-hoc sufficiency score:

$$S(q) = \frac{1}{|C|} \sum_{c \in C} \frac{|I_c^*(q) \cap I_{T,c}|}{|I_c^*(q)|} \quad (2)$$

	Query decomp.	Exec. feedback	KB access	User clarify	Acq. routing	Budget aware
DIN-SQL NeurIPS'23	✓	✗	✗	✗	✗	✗
DAIL-SQL VLDB'24	✗	✗	✗	✗	✗	✗
MAC-SQL COLING'25	✓	✓	✗	✗	✗	✗
CHASE-SQL ICLR'25	✓	✓	✗	✗	✗	✗
ECLAIR AAAI'25	✗	✗	✗	✓	✗	✗
a-Interact ICLR'26	✗	✓	✓	✓	✗	✗
PASCAL	✗	✓	✓	✓	✓	✓

Table 3: Mechanism coverage.

where $\mathcal{C} = \{\text{tab}, \text{col}, \text{kb-id}, \text{kb-term}, \text{intent}\}$ indexes five scored acquisition facets covering Δ_{sch} , Δ_{kb} , and Δ_{usr} . $I_c^*(q)$ is the required set restricted to class c , $I_{T,c}$ is the corresponding acquired subset at trajectory end, and empty required sets contribute a ratio of 1. $S(q)$ is computed from gold annotations and serves as a diagnostic metric, not a runtime feature. Detailed in Appendix D, empirical validation in Section 5.4)

Runtime acquisition schedule. ISH motivates an acquisition-first design. The gap taxonomy (Section 2.2) and source-level costs (Section 2) structure the protocol’s five channels and their ordering at design time. At runtime, the remaining question is which channel to query next.

For each gap class c , a lightweight signal $\text{need}_t(c)$ estimates whether c remains unresolved at turn t , computed from the current history h_t (e.g., unresolved schema mentions, KB-name matches without retrieved definitions, user-facing ambiguities, or references to prior-phase results). The controller ranks candidates by $\text{priority}_t(c) = \text{need}_t(c)/\text{cost}(c)$, where $\text{cost}(c)$ is the design-time constant. This favors low-cost, high-need sources: structural grounding first, then KB retrieval, then user clarification. Execution feedback is handled reactively by the retry breaker; phase-carry gaps mechanically by state preservation. Table 3 compares mechanism coverage across methods.

4.2 Acquisition routing

The priority schedule of Section 4.1 specifies an ordering over gap classes; acquisition routing enforces it structurally. A lightweight controller gates the agent’s action space at each turn: low-cost channels (Δ_{sch} , then Δ_{kb}) are opened first, and SQL submission is blocked until a minimum number of acquisition steps are taken. This prevents the agent from bypassing information gathering to attempt premature SQL generation, the dominant failure mode identified in Section 3. The controller restricts the available action set at each turn but does not select actions; the model chooses freely within

the permitted set. As gap classes are addressed, the controller progressively unlocks higher-cost channels and eventually permits SQL submission.

4.3 Gap-closing mechanisms

Each gap class is closed by a dedicated mechanism scheduled by the routing controller. We detail the two with non-trivial design (KB retrieval and retry control) and summarize the remaining three; Figure 2 illustrates the framework.

Schema pre-injection (Δ_{sch}). Schema discovery is not the bottleneck, but the budget it consumes competes with higher-leverage channels. PASCAL injects the full schema diagram $\mathcal{M}$ into the agent’s context at task start as a zero-cost operation (Biswal et al., 2026; Lei et al., 2025), freeing budget for downstream acquisition. Residual structural gaps (computed columns, ambiguous aliases) are resolved by execution feedback.

Typed knowledge retrieval (Δ_{kb}). Injecting all KB content is impractical when definitions exceed the model’s effective window. PASCAL pre-processes KB content offline into typed, callable entries, each indexed by a canonical name and a short summary. The agent discovers entries through a *namespace listing* and fetches definitions on demand through *definition retrieval*. On the museum task (Fig. 1), namespace listing reveals “*Environmental Instability Criteria*”, which the agent retrieves to close Δ_{kb} for Phase-1 SQL.

Namespace exposure addresses two failure modes visible in trajectories: (i) the agent queries the KB with a term that does not match the canonical entry name, and (ii) the agent never queries the KB because parametric knowledge produces a plausible but incorrect threshold. Both indicate the agent lacks *namespace awareness*.

Informed clarification (Δ_{usr}). User clarification is the most expensive action and the only channel for Δ_{usr} . The routing controller gates clarification behind schema and KB acquisition, triggering it only when one of three observable signals indicates cheaper channels cannot close the gap: (i) a query term has no match in the KB namespace listing; (ii) N_v consecutive submissions return the same normalized SQL hash; or (iii) a retrieved KB entry contains a formula with an unbound parameter. Continue with Fig. 1, this yields “The KB defines instability as $\text{TempVar24h} > 1$ OR $\text{HumVar24h} > 3$; apply this threshold?” rather than “What does

‘unstable conditions’ mean?” — a single-parameter confirmation instead of a full specification request.

Retry control (Δ_{exe}). Execution-driven self-correction is standard in text-to-SQL (Deng et al., 2025), but under finite budget each retry competes with acquisition. PASCAL connects retry to the acquisition schedule through a *retry breaker* monitoring the submission history $H_t = \{e_1, \dots, e_t\}$ (error category + normalized SQL hash per entry):

$$\text{fire}(H_t) = \begin{cases} \text{STOP}, & |H_t| \geq N_{\text{cap}}, \\ \text{STUCK}, & \text{last } N_v \text{ hashes match}, \\ \text{SWITCH}, & \text{last } N_e \text{ errors match}. \end{cases}$$

STOP caps total submissions; STUCK escalates to `ask_user` (repeated identical SQL implies the agent cannot self-correct); SWITCH redirects to an untried acquisition channel. Thresholds are tightened when information failures dominate (forcing earlier escalation to acquisition) and loosened when hard tasks legitimately require extended generation attempts (Detailed in Appendix G.4).

Follow-up composition (Δ_{ph}). Multi-phase workflows require state preservation: a follow-up like “for each of those showcases” cannot be answered without the prior phase’s resolved state (Yu et al., 2019a; Huo et al., 2026). On Phase-1 success, PASCAL preserves acquired items $\mathcal{I}_T^{(1)}$ and the successful SQL σ_1 ; the Phase-2 agent starts with $\mathcal{I}_0^{(2)} = \mathcal{I}_T^{(1)} \cup \{\sigma_1\}$, and references to prior results resolve through a CTE materialization of σ_1 (Alg. 3). Δ_{ph} is empty in single-phase tasks.

5 Experiments

Four research questions structure the analysis. **RQ1:** Does the acquisition-first protocol close the performance gap? **RQ2:** Which acquisition mechanism drives the gain? **RQ3:** Does ISH generalise as a task-level diagnostic? **RQ4:** Do the gains persist when interaction conditions change?

5.1 Experimental Settings

Benchmark. We evaluate PASCAL primarily on BIRD-Interact (Huo et al., 2026), which instantiates the environment $\mathcal{E}$ of Section 2.1 with databases, schema metadata, a hierarchical knowledge base, and a user simulator. We use *lite-300* tasks for controlled experiments and *full-600* tasks for leaderboard comparison; cross-benchmark validation uses PRACTIQ (Dong et al., 2025).

Metrics. A task passes P1 if the Phase-1 SQL result matches the gold and passes P2 if both phases match. We report P1, P2, and the BIRD-Interact Normalised Reward (Huo et al., 2026); Interaction behaviour is measured by tool-calling entropy $H = -\sum_a p(a) \log_2 p(a)$, averaged over per-task action distributions.

Baselines. We compare three groups: generation-oriented pipelines (DIN-SQL (Pourreza and Rafiei, 2023), DAIL-SQL (Gao et al., 2024), MAC-SQL (Wang et al., 2025), CHESS (Talaie et al., 2024)); interactive ReAct agents under a-Interact (Yao et al., 2023; Huo et al., 2026); and frontier leaderboard references.² PASCAL is orthogonal to these methods: it controls what to acquire and when, independently of the SQL generator or reasoning style. All methods use a fixed budget of 30 BIRD coins and 100 turns (Appendix G).

Implementation. PASCAL runs with Qwen3.6-35B by default and Qwen3.6-27B as an ablation, served by vLLM on 2×RTX 4090; following benchmark setting, all methods use the same Gemini-2.5-Flash-Lite as user simulator model.

5.2 RQ1: Does Acquisition-First Interaction Close the Performance Gap?

Table 4 compares PASCAL against generation-oriented pipelines, reported frontier systems, and official ReAct agents, all on *lite_300*. PASCAL achieves the strongest P1, P2, and Reward, showing that acquisition-first control closes much of the gap to reported frontier systems. Detailed results with case study appear in Appendix E-F.

PASCAL achieves state-of-the-art on BIRD-Interact. PASCAL turns local backbones that are weak under the standard ReAct scaffold (Yao et al., 2023) into the strongest interactive text-to-SQL agents reported on BIRD-Interact. PASCAL 35B reaches 51.0% P1 and 46.80 Normalised Reward on *lite_300* (Table 4), exceeding the strongest frontier model (Claude-Opus-4.6, 36.3% P1, 32.63 Reward) by +14.7 pp P1 and +14.2 Reward. The same Qwen-35B backbone obtains only 22.7% P1 and 12.27 Reward under the official scaffold, while all generation-oriented baselines remain below 20.7% P1. On the full 600-task split³, PASCAL 35B reaches 38.1% P1, surpassing both Claude-Opus-4.6 (33.0%) and GPT-5 (29.2%) on the offi-

²<https://bird-interact.github.io/>, until 26-05-25

³Full Mode, a-Interact, retrieved until 26-05-25

Method	BI%	DM%	P1%	P2%	Reward
<i>Generation (Qwen3.6-35B backbone)</i>					
DIN-SQL	10.8	14.3	12.0	6.7	10.40
DAIL-SQL	15.9	29.5	20.7	13.0	18.37
MAC-SQL	7.7	10.5	8.7	1.7	6.57
CHESS	14.4	8.6	12.3	6.0	10.43
<i>Frontier (a-Interact, lite_300)</i>					
Claude-Opus-4.6	24.2	58.8	36.3	24.0	32.63
Claude-Sonnet-3.7	22.1	56.2	34.0	17.7	29.10
GLM-4.7	18.6	48.4	29.0	9.0	23.00
MiniMax-M2.1	18.5	46.5	28.3	13.0	23.73
Kimi-2.5	17.8	44.9	27.3	11.3	22.53
Qwen3.6-27B	19.0	48.6	29.3	19.3	17.70
Qwen3.6-35B	10.8	44.8	22.7	11.3	12.27
<i>PASCAL</i>					
PASCAL 27B	41.8	60.0	48.3	35.4	44.42
PASCAL 35B	43.6	64.8	51.0	37.0	46.80

Table 4: Result of success rate on BIRD-Interact lite_300 tasks. BI=Business Intelligence ($n=195$); DM=Data Management ($n=105$).

cial leaderboard. This validates acquisition control as the main lever behind performance lift.

Acquisition-first interaction disproportionately benefits BI tasks. The protocol improves both BI and DM tasks, with larger gains on BI where success depends more heavily on external domain evidence. PASCAL 35B improves BI by +32.8 pp over the bare Qwen-35B backbone (10.8% $\rightarrow$ 43.6%) and DM by +20.0 pp (44.8% $\rightarrow$ 64.8%). PASCAL 27B shows the same pattern: BI +22.8 pp (19.0% $\rightarrow$ 41.8%) vs DM +11.4 pp (48.6% $\rightarrow$ 60.0%). DM tasks are mostly schema-resident and benefit primarily from schema pre-injection; BI tasks require domain definitions, thresholds, and user-specific interpretations that must be obtained from $\mathcal{K}$ or $\mathcal{U}$. The larger BI gain confirms that the protocol improves information gathering, with the strongest effect on tasks whose required evidence lies outside the schema.

Overall, RQ1 shows that PASCAL closes the performance gap by shifting interaction from generation retry to structured acquisition; the behavioural mechanisms behind this gain are analysed in the following content.

5.3 RQ2: Which Acquisition Mechanism Drives the Gain?

Aggregate success rates show that a method works but not how it interacts. Table 5 profiles budget allocation with four measures: **Gen%**, the share of actions spent on SQL execution and submission; **Acq%**, the share on schema, KB, and user tools (Gen% + Acq% = 100%); **KB%**, the subset

Method	Gen%	Acq%	KB%	H	Rwd
DAIL-SQL	96.9	3.1	0.0	0.81	18.37
a-Interact 27B	44.5	55.5	36.7	2.16	17.70
a-Interact 35B	49.1	50.9	31.6	2.20	12.27
PASCAL 27B	72.1	27.9	9.1	1.95	44.42
PASCAL 35B	80.0	17.8	7.6	1.78	46.80

Table 5: Tool allocation and per-task entropy (H) on lite_300.

Variant	P1% $\uparrow$	Rwd $\uparrow$	InfoFail% $\downarrow$	1st-KB $\downarrow$	Rpt-SQL $\downarrow$	Overthink $\downarrow$
No protocol	32.3	28.5	86.67	1.000	0.680	0.354
+ Schema + Prog.	36.9	33.5	67.70	1.000	0.430	0.307
+ KB retrieval	42.1	37.1	40.69	0.680	0.300	0.311
+ harness	51.0	46.80	31.10	0.427	0.258	0.154

Table 6: Ablation on lite-300 (PASCAL 35B).

of Acq% on KB retrieval; and per-task tool-calling entropy H . Table 6 traces four trajectory indicators through the additive ablation: **InfoFail%**, the fraction of tasks failing because required KB or intent items remain unacquired at submission; **1st-KB**, the normalised position of the first ground-truth KB call (0 = start, 1 = never); **Rpt-SQL**, the peak repeat count of the same normalised SQL; and **Overthink**, the fraction of turns after the last information gain.

Tool allocation reveals three interaction regimes.

DAIL-SQL, representative of generation-oriented pipelines (Gao et al., 2024), concentrates 96.9% of actions on generation with entropy $H=0.81$: a single-channel loop that never queries KB or user. Official a-Interact disperses over half its budget to acquisition (KB% = 31–37%, $H=2.16$ –2.20) yet achieves the lowest Reward: the agent tries every tool but routes none. PASCAL generates after targeted acquisition (Gen% = 72–80%, KB% = 7.6–9.1%, $H=1.78$ –1.95) and achieves the highest (Table 5). Tool-calling entropy H is not monotonically related to success: too low indicates a single-action loop (DAIL-SQL), too high indicates undirected dispersion (a-Interact). PASCAL’s moderate H reflects concentrated use of the right tools at the right time: namespace exposure routes KB calls to relevant entries, and progressive access ensures acquisition precedes generation. These results show that targeted acquisition and ordering explain the performance gap better than acquisition volume.

Acquisition channels progressively resolve the information bottleneck.

Table 6 traces how each added mechanism changes both outcome and trajectory behaviour. Adding schema pre-injection and progressive access (row 2) reduces InfoFail% from 86.7 to 67.7 and Rpt-SQL from 0.68 to 0.43: the agent retries less and resolves more tasks, but 1st-KB remains 1.000: KB is still never called. Adding KB retrieval (row 3) produces a qualita-

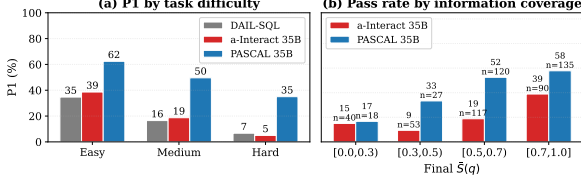


Figure 3: P1 by difficulty tier (left) and by final $S(q)$ bin (right) on lite_300.

Gap class	$\bar{S}_c$ (pass)	$\bar{S}_c$ (fail)	Δ	Low-cov.
δ_{sch} (table)	0.970	0.835	+0.135	11.6%
δ_{sch} (column)	0.860	0.676	+0.184	19.0%
δ_{kb} (id)	0.317	0.256	+0.062	67.3%
δ_{kb} (term)	0.615	0.575	+0.041	41.5%
δ_{usr} (intent)	0.322	0.451	-0.129	44.2%

Table 7: Residual gap analysis for PASCAL 35B on lite_300. Low-cov.: failed tasks with $S_c < 0.5$.

tive shift: InfoFail% drops to 40.7 (−27 pp), and 1st-KB falls to 0.680 as KB calls enter the trajectory for the first time. The InfoFail% drop directly reflects the opening of a previously closed information source: tasks that failed due to missing domain knowledge now have a channel to acquire it. Opening the channel is necessary, but Overthink rises slightly ($0.307 \rightarrow 0.311$); the agent acquires KB information yet does not know when to stop retrying generation.

The deployment harness controls exploration quality. The Overthink increase in row 3 exposes a gap between having information and using it effectively. Row 4 adds the deployment harness: Overthink drops from 0.311 to 0.154 (−0.157), 1st-KB falls from 0.680 to 0.427, InfoFail% falls to 31.1, and P1 rises by +8.9 pp to 51.0%. The harness makes KB acquisition earlier and cuts the unproductive tail after information stops growing. The retry breaker limits generation retries when information is still incomplete, preserving budget for acquisition; once key gaps are addressed, it relaxes limits to let the agent explore the SQL space, preventing both premature generation and uncontrolled retry after acquisition. Effective interactive text-to-SQL thus requires channel access, temporal ordering, and exploration control acting together.

Overall, RQ2 identifies three layers of mechanism contribution in Table 6. The progressive InfoFail% reduction provides mechanism-level support for ISH: as more information gaps closed, tasks limited by missing evidence drops correspondingly.

5.4 RQ3: When Does Information Sufficiency Explain Success and Failure?

RQ2 shows which acquisition pathway drives the gain; RQ3 tests whether ISH explains which tasks

benefit and which failures remain. We use three diagnostic metrics derived from the sufficiency score $S(q)$ (Eq. 2). (i) **Difficulty tiers.** following official annotation. (ii) **Coverage bins.** We partition tasks by their final $S(q)$ into four bins and report the P1 pass rate (Figure 3, right). (iii) **Per-class residual analysis.** Let $S_c(q)$ denote the per-task coverage of gap class c (the per-class term inside Eq. 2). We average S_c over P1-passed and P1-failed tasks separately, writing $\bar{S}_c(\text{pass})$ and $\bar{S}_c(\text{fail})$, and report $\Delta = \bar{S}_c(\text{pass}) - \bar{S}_c(\text{fail})$ together with the percentage of failed tasks whose $S_c < 0.5$ (Table 7).

Information coverage is consistent with ISH.

ISH predicts that task success depends on acquiring task-critical evidence before committing to SQL (Section 4.1). Across difficulty tiers, PASCAL’s advantage is largest on hard tasks that require evidence from multiple sources: PASCAL 35B reaches 35.0% P1 versus 5.0% for ReAct 35B (Figure 3, left). At the task level, P1 increases monotonically with final $S(q)$: 16.7% for $S(q) \in [0.0, 0.3]$ to 57.8% for $[0.7, 1.0]$ (Figure 3, right); under ReAct 35B the same partition yields only 15.0% to 38.9%. This indicates that information coverage is strongly associated with task-level success. Appendix F traces six per-task trajectories, further illustrating that baselines fail from premature commitment, not SQL quality.

The gap taxonomy reveals remaining errors.

Table 7 shows that structural coverage remains high on failed tasks ($\bar{S}_{sch} > 0.67$), meaning schema gaps are effectively closed by pre-injection. Residual errors concentrate on δ_{kb} (67.3% of failures have $S_{kb}^{id} < 0.5$) and δ_{usr} (44.2% have $S_{int} < 0.5$). Failed tasks show higher intent coverage than passed tasks ($\Delta = -0.129$): when KB retrieval cannot resolve the task, the agent redirects budget to user clarification. This per-class breakdown turns undifferentiated failures into concrete targets for improving each channel. The same diagnostic applies beyond BIRD-Interact: on PRACTIQ, a conversational benchmark of ambiguous and unanswerable queries, PASCAL reaches 86.5% overall, +14.6 pp over SOTA baseline (Appendix I).

5.5 RQ4: How Do Interaction Conditions Affect PASCAL?

RQ1–RQ3 link PASCAL’s gain to acquiring missing information before SQL generation. RQ4 tests this explanation under changes to evidence access, model capacity, and interaction controls.

Schema attribution and task-level reversals use all 300 lite_300 tasks; the remaining controls use the same 100-task subset as Section 3.1. Appendix H reports the full settings and results.

The gain extends beyond schema access. To separate schema input from acquisition order, we give the official ReAct scaffold the same initial schema as PASCAL while leaving its interaction policy unchanged. On all 300 tasks, P1 moves from 22.7% to 23.0%, while PASCAL reaches 51.0% (Table 40). The shared schema resolves the initial structural lookup, while the official ReAct scaffold still permits SQL generation before KB and user information are acquired. PASCAL instead places these sources before generation. The remaining gap therefore reflects the effect of acquisition control beyond schema access.

Robustness to user feedback and backbone capability. We test PASCAL along two robustness axes on the same 100-task subset. For the user channel, replacing Gemini-2.5-Flash-Lite with GLM-5.2 gives 54.0% and 53.0% P1, while withholding exact formulas and thresholds with probabilities 0.3 and 0.5 gives 57.0% and 51.0%, compared with 31.0% and 37.0% for the official ReAct scaffold. For the backbone, replacing Qwen3.6-35B with Qwen3.6-Plus increases P1 from 54.0% to 65.0%, with most additional successes on Query tasks requiring multiple information sources (Appendix H, Tables 41 and 42). Late clarification reduces sensitivity to incomplete user feedback, while a stronger backbone further improves cross-source queries.

Interaction controls remain effective under different settings. We further test the two controls that govern PASCAL’s interaction policy on the same 100-task subset. First, we reverse the relative tool costs, making user clarification cheaper and SQL execution more expensive. P1 changes from 54.0% to 57.0%, while the agent uses user clarification more often and SQL execution less often. This shows that the cost-aware priority changes source allocation with the cost profile. Second, we vary the total interaction budget and the retry thresholds that trigger a return from SQL execution to information acquisition. Budgets of 15, 21, 30, and 45 yield 47.0%, 51.0%, 54.0%, and 55.0% P1, while five settings of the retry thresholds yield 49.0%, 51.0%, 54.0%, 47.0%, and 49.0% (Appendix H, Tables 43 and 44). The former controls which source is preferred, while the latter controls when

the agent stops retrying SQL and resumes information acquisition.

Failure case analysis. Across all 300 lite_300 tasks, PASCAL passes 86 tasks that the official ReAct scaffold fails, while seven tasks show the reverse outcome. These seven regressions fall into three patterns: repeated SQL resubmission, mis-attributed SQL errors, and late user clarification (detailed in Appendix H). The first two occur after information acquisition, while the third follows from PASCAL’s schema/KB-first ordering when the missing information is available only from the user. On easy Management tasks, PASCAL reaches its first SQL with fewer acquisition actions (3.45 vs. 5.10), showing that the main scheduling weakness lies in delayed access to user-only information.

6 Related work and Conclusion

Related work detailed in Appendix A. We identify information acquisition as the key bottleneck in interactive text-to-SQL and formalize it with ISH and a five-class gap taxonomy. Guided by this diagnosis, PASCAL provides a training-free acquisition-control framework that schedules source-specific acquisition by cost-normalized priority before SQL generation. On BIRD-Interact, PASCAL achieves 51.0% success rate, 14.7% above SOTA baseline, shows that cost-aware acquisition routing is a central factor in interactive text-to-SQL.

Acknowledgments

This work was supported by grants from the Natural Science Foundation of China (No. 62572035, 62372275), the Natural Science Foundation of Shandong Province (ZR2026QC1030), and supported by Shandong Provincial Key Laboratory of Computing-Network Integration, and Qingdao Key Laboratory of Trustworthy Artificial Intelligence, Shandong University.

The authors confirm that no part of the paper’s main content, including text, figures, tables, or code, was generated by artificial intelligence (AI) tools. Minor assistance by AI was limited to grammar checking and code development using standard productivity software, all reviewed and validated by the authors.

Limitations

Benchmark and environment scope. Our evaluation is conducted on two English-language benchmarks that share a similar five-source environment structure. The gap taxonomy and acquisition schedule are tied to this structure; environments with substantially different information sources, interaction modalities, or cost profiles would require revisiting both the taxonomy and the priority ordering. The current knowledge bases are structured; substantially noisier knowledge sources remain untested.

Simulated user interaction. The user clarification channel is evaluated with a rule-based simulator that returns deterministic, gold-grounded answers. Real users may provide incomplete, ambiguous, or even misleading feedback, which could affect both the clarification channel’s effectiveness and the cost assumptions underlying the acquisition schedule. The protocol’s robustness under noisy human feedback remains an open question. Users who introduce new ambiguity mid-dialogue remain untested.

Rule-based acquisition routing. The priority schedule relies on a closed-form rule with design-time cost estimates rather than a learned routing policy. This is a deliberate choice that ensures training-free deployment and interpretability, but it limits the protocol’s ability to adapt to deployment environments where the relative cost structure differs from the one assumed at design time. A new deployment would require mapping its available acquisition channels to the gap classes and re-estimating their relative costs.

References

- Linus Aronsson, Arman Rahbar, and Morteza Haghir Chehreghani. 2025. [A survey on active feature acquisition strategies](#). *Preprint*, arXiv:2502.11067.
- Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. [Benchmarking and improving text-to-SQL generation under ambiguity](#). In *EMNLP*.
- Asim Biswal, Chuan Lei, Xiao Qin, Aodong Li, Balakrishnan Narayanaswamy, and Tim Kraska. 2026. [Agentsm: Semantic memory for agentic text-to-sql](#). *Preprint*, arXiv:2601.15709.
- Albert Chen, Manas Bunde, Gaurav Ahlawat, Patrick Stetz, Zhitao Wang, Qiang Fei, Donghoon Jung, Audrey Chu, Bharadwaj Jayaraman, Ayushi Panth, Yatin Arora, Sourav Jain, Renjith Varma, Alexey Ilin, Iuliia Melnychuk, Chelsea Chueh, Joyan Sil, and Xiaofeng Wang. 2025a. [Text-to-sql for enterprise data analytics](#). *Preprint*, arXiv:2507.14372.
- Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Mike Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2025b. [BEAVER: An enterprise benchmark for text-to-SQL](#). In *The 4th Table Representation Learning Workshop at ACL 2025*.
- Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. [RefoRCE: A text-to-SQL agent with self-refinement, format restriction, and column exploration](#). In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*.
- Wenxuan Ding, Nicholas Tomlin, and Greg Durrett. 2026a. [Calibrate-then-act: Cost-aware exploration in llm agents](#). *Preprint*, arXiv:2602.16699.
- Zhongjun Ding, Yin Lin, Tianjing Zeng, Rong Zhu, Bolin Ding, and Jingren Zhou. 2026b. [Ambisql: Interactive ambiguity detection and resolution for text-to-sql](#). In *Companion of the International Conference on Management of Data*, pages 26–29.
- 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. 2025. [PRACTIQ: A practical conversational text-to-SQL dataset with ambiguous and unanswerable queries](#). In *NAACL*.
- Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. [Text-to-SQL empowered by large language models: A benchmark evaluation](#). *Proceedings of the VLDB Endowment*, 17(5).
- Daniel Golovin and Andreas Krause. 2011. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujia Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. [ToRA: A tool-integrated reasoning agent for mathematical problem solving](#). In *ICLR*.
- Omid Reza Heidari, Siobhan Reid, and Yassine Yaakoubi. 2025. [AGENTIQL: An agent-inspired multi-expert framework for text-to-SQL generation](#). In *NeurIPS 2025 Workshop on Efficient Reasoning*.
- Thanh Dat Hoang, Thanh Trung Huynh, Matthias Weidlich, Thanh Tam Nguyen, Tong Chen, Hongzhi Yin, and Quoc Viet Hung Nguyen. 2025. [A multi-agent text2sql framework using small language models and execution feedback](#). *Preprint*, arXiv:2512.18622.
- Harper Hua, Zhen Han, Zhengyuan Shen, Meng-Chieh Lee, Sheng Guan, Qi Zhu, Sullam Jeoung, Yueyan Chen, Yunfei Bai, Shuai Wang, and 1 others. 2026.

- Sql-trail: Multi-turn reinforcement learning with interleaved feedback for text-to-sql. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 36224–36246.
- Tenghao Huang, Sihao Chen, Muhao Chen, Jonathan May, Longqi Yang, Mengting Wan, and Pei Zhou. 2025. [Teaching language models to gather information proactively](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 15588–15599. Association for Computational Linguistics.
- 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, and 5 others. 2026. [BIRD-INTERACT: Re-imagining text-to-SQL evaluation via lens of dynamic interactions](#). In *The Fourteenth International Conference on Learning Representations*.
- Ai Jian, Xiaoyun Zhang, Wanrou Du, Jingqing Ruan, Jiangbo Pei, Weipeng Zhang, Ke Zeng, and Xunliang Cai. 2026. [Trust-sql: Tool-integrated multi-turn reinforcement learning for text-to-sql over unknown schemas](#). *Preprint*, arXiv:2603.16448.
- Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. [KaggleDBQA: Realistic evaluation of text-to-SQL parsers](#). In *ACL*.
- Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. [Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows](#). In *The Thirteenth International Conference on Learning Representations*.
- Jinyang Li, Binyuan Hui, GE QU, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. [Can LLM already serve as a database interface? a BIG bench for large-scale database grounded text-to-SQLs](#). In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Tengxiao Liu, Zifeng Wang, Jin Miao, I-Hung Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, Samira Daruki, Yi Liang, William Yang Wang, Tomas Pfister, and Chen-Yu Lee. 2026. [Budget-aware tool use enables effective agent scaling](#). In *Third Conference on Language Modeling*.
- Yuyu Luo, Guoliang Li, Ju Fan, and Nan Tang. 2026. [Data agents: Levels, state of the art, and open problems](#). *SIGMOD Tutorial*.
- John Murzaku, Zifan Liu, Md Mehrab Tanjim, Vaishnavi Muppala, Xiang Chen, and Yunyao Li. 2025. [Eclair: Enhanced clarification for interactive responses](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 28864–28870.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025. [CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL](#). In *The Thirteenth International Conference on Learning Representations*.
- Mohammadreza Pourreza and Davood Rafiei. 2023. [DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Irina Saparina and Mirella Lapata. 2024. [AMBROSIA: A benchmark for parsing ambiguous questions into database queries](#). In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. [CHESS: Contextual harnessing for efficient SQL synthesis](#). *arXiv preprint arXiv:2405.16755*.
- Dat Tran and Douwe Kiela. 2026. [Single-agent llms outperform multi-agent systems on multi-hop reasoning under equal thinking token budgets](#). *Preprint*, arXiv:2604.02460.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. [MAC-SQL: A multi-agent collaborative framework for text-to-SQL](#). In *COLING*, pages 540–557, Abu Dhabi, UAE.
- Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. [Budgetthinker: Empowering budget-aware llm reasoning with control tokens](#). *Preprint*, arXiv:2508.17196.
- Zekun Xu, Siyu Xia, Chuhuai Yue, Jiajun Chai, Mingxue Tian, Xiaohan Wang, Wei Lin, Haoxuan Li, and Guojun Yin. 2025. [Mtir-sql: Multi-turn tool-integrated reasoning reinforcement learning for text-to-sql](#). *Preprint*, arXiv:2510.25510.
- Haolin Yang, Jipeng Zhang, Zhitao He, Alexander Zhou, and Yi R. Fung. 2026. [Mars-sql: A multi-agent reinforcement learning framework for text-to-sql](#). *Preprint*, arXiv:2511.01008.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. [Swe-agent: agent-computer interfaces enable automated software engineering](#). In *NIPS*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *ICLR*.

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, and 5 others. 2019a. [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 (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. [Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task](#). In *EMNLP*, pages 3911–3921, Brussels, Belgium.

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. 2019b. [SPaC: Cross-domain semantic parsing in context](#). In *ACL*, pages 4511–4523, Florence, Italy.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Related Work

We organise related work along four axes—generation methods, interactive and multi-turn querying, ambiguity and clarification, and cost-aware information acquisition—and summarise the positioning in Table 8.

Single-turn text-to-SQL with training. Reinforcement learning has recently been applied to text-to-SQL. MTIR-SQL (Xu et al., 2025) uses GRPO multi-turn RL with a 4B backbone and reports 64.4% execution accuracy on BIRD-dev. SQL-Trail (Hua et al., 2026) trains a 7B agent with 1,873 trajectories and reaches 60.1%. TRUST-SQL (Jian et al., 2026) handles unknown schemas with RL and 3B–8B models. MARS-SQL (Yang et al., 2026) decomposes text-to-SQL into a multi-agent RL framework with a ReAct-style think–act–observe loop, reaching 77.8% on BIRD-dev. All

four methods optimise $P(\text{correct SQL} \mid \text{context})$ in a single-turn setting where the full context is given in the prompt. Our diagnosis (ISH, Section 4.1) is that the dominant factor shifts to $P(\text{sufficient context})$ once the task becomes interactive and context must be actively acquired.

Training-free single-turn. DIN-SQL (Pourreza and Rafiei, 2023) decomposes queries into sub-problems with self-correction; DAIL-SQL (Gao et al., 2024) selects few-shot examples by question similarity; MAC-SQL (Wang et al., 2025) coordinates multiple agents for schema linking, decomposition, and refinement; CHESS (Taleai et al., 2024) filters schemas and retrieves evidence hints; CHASE-SQL (Pourreza et al., 2025) combines multi-path reasoning with preference-optimised candidate selection, reaching 73.0% on the BIRD test set. AGENTIQ (Heidari et al., 2025) routes queries between a multi-expert pipeline and a baseline parser for efficiency, but this is query-complexity routing within a single-turn setting, not source-specific acquisition routing across an interactive environment. These pipelines report strong single-turn results but operate in a closed information space: all context comes from the prompt and (optionally) execution feedback. They do not address ambiguity resolution, KB-driven domain knowledge, or phase-carry across follow-up questions. When we adapt these methods to BIRD-Interact’s interactive environment (Section 5), they remain below 20.7% P1 (Table 4).

Interactive and multi-turn querying. CoSQL (Yu et al., 2019a) and SPaC (Yu et al., 2019b) introduce conversational text-to-SQL with context-dependent questions, but assume schema is fully provided and do not model external knowledge sources. BIRD-Interact (Huo et al., 2026) extends the paradigm with a hierarchical knowledge base, multi-type ambiguity, and follow-up questions in a PostgreSQL environment. The model autonomously decides when to call each tool. As shown in Section 3.2, this leads to speculative KB calls that waste budget without closing information gaps (Obs. 1). AgentSM (Biswal et al., 2026) adds semantic memory to improve cross-task consistency in agentic text-to-SQL. Spider 2.0 (Lei et al., 2025) evaluates enterprise workflows with large schemas but does not include KB retrieval or user interaction channels. QueryGPT (Chen et al., 2025a) reports production-scale interactive querying at LinkedIn with workspace-based

Method	Venue	Query decomp.	Exec. feedback	KB access	User clarify	Budget aware	Setting
<i>Training-based, single-turn</i>							
MTIR-SQL (Xu et al., 2025)	arXiv’25	✓	✓	✗	✗	✗	BIRD-dev
SQL-Trail (Hua et al., 2026)	arXiv’26	✓	✓	✗	✗	✗	BIRD-dev
TRUST-SQL (Jian et al., 2026)	arXiv’26	✗	✓	✗	✗	✗	BIRD-dev
MARS-SQL (Yang et al., 2026)	arXiv’25	✓	✓	✗	✗	✗	BIRD-dev, Spider
<i>Training-free, single-turn</i>							
DIN-SQL (Pourreza and Rafiei, 2023)	NeurIPS’23	✓	✗	✗	✗	✗	Spider, BIRD
DAIL-SQL (Gao et al., 2024)	VLDB’24	✗	✗	✗	✗	✗	Spider, BIRD
MAC-SQL (Wang et al., 2025)	COLING’25	✓	✓	✗	✗	✗	Spider, BIRD
CHES (Taleai et al., 2024)	arXiv’24	✓	✗	✗ [†]	✗	✗	BIRD
CHASE-SQL (Pourreza et al., 2025)	ICLR’25	✓	✓	✗	✗	✗	BIRD
AGENTIQL (Heidari et al., 2025)	arXiv’25	✓	✓	✗	✗	✗	Spider
<i>Interactive / multi-turn</i>							
CoSQL (Yu et al., 2019a)	EMNLP’19	✗	✗	✗	✓	✗	CoSQL
ECLAIR (Murzaku et al., 2025)	AAAI’25	✗	✗	✗	✓	✗	Enterprise
a-Interact (Huo et al., 2026)	ICLR’26	✗	✓	✓ [‡]	✓ [‡]	✗	BIRD-Interact
PASCAL (ours)		✗	✓	✓	✓	✓	BIRD-Interact

Table 8: Extended mechanism comparison. [†]CHES uses schema-linked evidence hints from BIRD’s gold annotation, not an agent-accessible KB channel. [‡]a-Interact exposes KB and user tools but provides no routing protocol; the model decides autonomously when to call each tool.

context management, though it does not formalise acquisition as a sequential decision problem.

Ambiguity and clarification. AmbiSQL (Ding et al., 2026b) introduces an ambiguity taxonomy with four classes (column, value, keyword, context) that all map into our Δ_{usr} gap class; it addresses a subset of our five-class taxonomy. PRAC-TIQ (Dong et al., 2025) contributes a dataset of ambiguous and unanswerable queries for conversational text-to-SQL; we apply our gap-taxonomy diagnosis to PRACTIQ and report 86.5% over-all accuracy. AMBROSIA (Saparina and Lapata, 2024) benchmarks ambiguous questions but targets single-turn parsing without agent interaction. ECLAIR (Murzaku et al., 2025) explores user clarification in an enterprise assistant, but treats clarification in isolation from KB access and budget-aware scheduling. MATS (Hoang et al., 2025) applies small-LM multi-agent methods to interactive text-to-SQL and is complementary to our work. Huang et al. (Huang et al., 2025) teach language models to gather information proactively, though in general-purpose QA rather than database interaction.

Cost-aware information acquisition. PASCAL’s cost-normalized priority (Section 4.1) draws on two research threads. The first is active feature acquisition under budget constraints: adaptive submodularity (Golovin and Krause, 2011) provides theoretical foundations for sequential

acquisition with diminishing returns, and Aronsson et al. (Aronsson et al., 2025) survey strategies where features are acquired at non-uniform costs before prediction. PASCAL instantiates this framework for the specific structure of interactive text-to-SQL, where the “features” are information items drawn from five source-specific channels.

The second thread is budget-aware agent exploration. Calibrate-then-Act (Ding et al., 2026a) trains LLM agents to trade off exploration against action costs, showing that cost-calibrated exploration outperforms both undirected exploration and immediate action—a finding that parallels our Obs. 1 (Section 3.2). BATS (Liu et al., 2026) applies budget-aware trial-and-error search to multi-hop web agents. BudgetThinker (Wen et al., 2025) allocates thinking budgets across reasoning steps. Tran and Kiela (Tran and Kiela, 2026) show that single-agent LLMs outperform multi-agent systems under equal token budgets, reinforcing the importance of budget allocation over agent count. PASCAL shares the budget-awareness motivation but targets a domain with structured information sources ($\mathcal{M}, \mathcal{K}, \mathcal{U}, \mathcal{D}, \mathcal{H}$) whose costs are heterogeneous and source-determined.

Positioning. Table 8 summarises mechanism coverage. The training-based $\times$ single-turn quadrant is well populated; the training-free $\times$ single-turn quadrant is equally crowded. PASCAL is the only work in the training-free $\times$ interactive quadrant on BIRD-Interact. The training-based $\times$ interac-

tive quadrant is currently empty: RL methods can in principle port to BIRD-Interact by adding an `ask_user` tool, but none have reported results at submission time. Our gap taxonomy and ISH findings would remain valid diagnostics for such future work.

B PASCAL Pseudocode

PASCAL processes each task in two stages: acquisition then construction. Algorithm 1 covers acquisition: it injects schema metadata, retrieves KB definitions through namespace exposure, and triggers user clarification when cheaper channels are exhausted (Stages 1–3 in Section 4). Algorithm 2 covers construction: it generates SQL, exposes structured value-diff feedback on failure, and uses a retry breaker to redirect the agent to acquisition when stagnation is detected (Stage 4). For multi-phase tasks, Algorithm 3 carries all acquired state from Phase 1 into Phase 2 via CTE materialization. The three algorithms execute sequentially: Alg. 1 $\rightarrow$ Alg. 2 for single-phase tasks, with Alg. 3 wrapping both for follow-up questions.

Implementation notes. (1) `NEEDSCLARIFY` triggers under three conditions (Section 4.3): no KB match for an ambiguity, N_v consecutive identical SQL hashes, or a partial KB definition leaving a parameter unspecified. (2) `THRESHOLDS` returns tighter limits when called from within the retry breaker (acquisition incomplete for that gap) and relaxed limits at Stage 4 entry (Section 4.3). (3) Schema injection and namespace listing are cached per-database across sibling tasks. (4) Each tool call deducts its BIRD-COIN cost (Table 38) from the task budget B ; the agent terminates when the budget is exhausted or the 100-turn ceiling is reached.

C Gap Taxonomy: Benchmark Coverage and Notation

C.1 Gap class coverage across benchmarks

Table 9 maps the five gap classes of Section 2.2 to major text-to-SQL benchmarks. Early benchmarks (WikiSQL, Spider) assume all required information is present in the schema and the query; the only acquisition channel is schema linking. BIRD introduces external knowledge bases, adding Δ_{kb} . Conversational benchmarks (SParC, CoSQL) introduce multi-turn context and user interaction, adding Δ_{usr} and Δ_{ph} . Spider 2.0 targets enterprise work-

Algorithm 1: PASCAL Acquisition (Stages 1–3)

Input: Query q , database $\mathcal{D}$, metadata $\mathcal{M}$, knowledge base $\mathcal{K}$, user $\mathcal{U}$, prior state $\mathcal{H}$

Output: Acquired information $\mathcal{I}$, KB namespace ns

// Stage 1: Structural grounding (Δ_{sch})

```

1  $\mathcal{I} \leftarrow \mathcal{H}$ 
2 Inject full schema  $\mathcal{M}$  into agent context
  // zero cost
3 Validate  $q$  against  $\mathcal{M}$ ; reject invalid
  table/column refs

// Stage 2: Knowledge retrieval ( $\Delta_{kb}$ )
4  $ns \leftarrow \text{LISTKBNames}(\mathcal{K})$  // namespace
5 Expose  $ns$  to agent context
6 foreach term  $a$  in  $q$  identified by the LLM as
  potentially requiring domain knowledge do
7    $e \leftarrow \text{LLMSELECT}(a, ns)$  // match term
  to entry, or NULL
8   if  $e \neq \text{NULL}$  then
9      $def \leftarrow \text{FETCHDEFINITION}(e)$ 
10     $\mathcal{I} \leftarrow \mathcal{I} \cup \{(e, def)\}$ 

// Stage 3: Informed clarification ( $\Delta_{usr}$ )
11 if  $\text{LLMDETECT}(ambiguity\ a, no\ KB\ match\ in\ ns)$  or
12    $KB\ definition\ for\ a\ leaves\ parameter\ unbound$  then
13   Formulate question grounded in  $\mathcal{I}$ 
  // informed
14    $resp \leftarrow \text{ASKUSER}(question)$ 
15    $\mathcal{I} \leftarrow \mathcal{I} \cup \{resp\}$ 
16 return  $\mathcal{I}, ns$ 
```

Algorithm 2: PASCAL Construction with Retry Control (Stage 4)

Input: Query q , acquired state $\mathcal{I}$, namespace ns , database $\mathcal{D}$, user $\mathcal{U}$, budget B

Output: SQL solution σ

```

1  $H \leftarrow \emptyset$ ;  $n_{sub} \leftarrow 0$ 
2  $(N_{cap}, N_v, N_e) \leftarrow \text{THRESHOLDS}(\text{TRUE})$ 
   // acquisition complete
3 while  $n_{sub} < N_{cap}$  and  $B > 0$  do
4    $\sigma \leftarrow \text{GENERATESQL}(q, \mathcal{I})$ 
5    $r, pass \leftarrow \text{EXECUTE}(\sigma, \mathcal{D})$  // pass = no
   // execution error
6    $B \leftarrow B - c(\text{EXECUTE})$ 
7   if  $pass$  then break
   // Structured value-diff feedback
8    $diff \leftarrow (\text{row-count, col-count, sample-value})$  divergence between  $r$ 
   // and the prior submission
9   Expose  $diff$  to agent context
10   $H \leftarrow H \cup \{(diff, \text{HASH}(\sigma))\}$ ;
    $n_{sub} += 1$ 
   // Retry breaker
11  if  $n_{sub} \geq N_{cap}$  then break // stop: hard
   // cap reached
12
13  if last  $N_v$  entries in  $H$  share same hash
   // then
   // stuck: same SQL resubmitted
14    if  $\text{NEEDSCLARIFY}(q, \mathcal{I}, ns, diff)$ 
   // then
15       $\mathcal{I} \leftarrow$ 
16       $\mathcal{I} \cup \{\text{ASKUSER}(q, \mathcal{I}, diff)\}$ 
17       $B \leftarrow B - c(\text{ASKUSER})$ 
18    else if last  $N_e$  entries share same error
   // category then
   // switch: different SQL, same error type
19    Redirect agent to re-examine  $\mathcal{I}$  for
   // missing KB entry
20   $B \leftarrow B - c(\text{SUBMIT})$ 
21 return  $\sigma$ 

```

Algorithm 3: Multi-Phase State Carry (Δ_{ph})

Input: Phase-1 state $\mathcal{I}_T^{(1)}$, Phase-1 SQL σ_1 , follow-up query q_2 , budget B_2

Output: Phase-2 SQL σ_2

```

1  $\mathcal{H}_2 \leftarrow \mathcal{I}_T^{(1)} \cup \{\sigma_1\}$  // inherit all Phase-1 items
2 Materialize  $\sigma_1$  as CTE phase1_result in
   // context
3  $\mathcal{I}_2, ns \leftarrow \text{Alg. 1}(q_2, \mathcal{D}, \mathcal{M}, \mathcal{K}, \mathcal{U}, \mathcal{H}_2)$ 
4  $\sigma_2 \leftarrow \text{Alg. 2}(q_2, \mathcal{I}_2, ns, \mathcal{D}, \mathcal{U}, B_2)$ 
5 return  $\sigma_2$ 

```

flows with execution feedback but does not model user clarification or follow-up composition. BIRD-Interact is the first benchmark that requires all five acquisition channels, making it the natural testbed for studying information acquisition under budget constraints.

C.2 Notation summary

Table 10 collects the symbols used throughout the paper.

D Information Sufficiency Extraction

This appendix details the computation of the sufficiency score $S(q)$ introduced in Section 4.1 (Eq. 2). For each task, we compare the gold-required information set $\mathcal{I}^*(q)$ against the agent’s acquired set $\mathcal{I}_t[T]$, reconstructed from the trajectory without re-executing the agent.

Required set $\mathcal{I}^*(q)$. The required set decomposes into five classes, each derived from the task’s ground-truth annotation: *Tables and columns* ($c \in \{\text{tab}, \text{col}\}$): we parse the gold SQL with sqlglot (Postgres dialect); CTE alias names are filtered so only base tables are counted. *KB IDs* ($c = \text{kb-id}$): the external_knowledge list of integer IDs on each task record. *KB terms* ($c = \text{kb-term}$): the knowledge_ambiguity annotation lists domain-term phrases the agent must retrieve. *Intent terms* ($c = \text{intent}$): the user_query_ambiguity.critical_ambiguity annotation lists terms the agent must disambiguate with the user.

Acquired set $\mathcal{I}_t[T]$. The acquired set is reconstructed from the trajectory’s tool calls. `execute_sql` and `submit_sql` contribute tables and columns parsed from their SQL arguments. `get_knowledge_definition` contributes KB IDs extracted from the response text and the queried term from its argument. `ask_user`

Benchmark	Δ_{sch}	Δ_{kb}	Δ_{usr}	Δ_{exe}	Δ_{ph}	Setting
WikiSQL (Zhong et al., 2017)	✓	✗	✗	✗	✗	single-turn, single-table
Spider (Yu et al., 2018)	✓	✗	✗	✗	✗	single-turn, cross-DB
KaggleDBQA (Lee et al., 2021)	✓	✗	✗	✗	✗	single-turn, real-web DB
SParC (Yu et al., 2019b)	✓	✗	✗	✗	✓	multi-turn, context-dep.
CoSQL (Yu et al., 2019a)	✓	✗	✓	✗	✓	conversational
BIRD (Li et al., 2023)	✓	✓	✗	✗	✗	single-turn, external KB
AmbiQT (Bhaskar et al., 2023)	✓	✗	✓	✗	✗	ambiguous queries
Spider 2.0 (Lei et al., 2025)	✓	✓	✗	✓	✗	enterprise workflows
PRACTIQ (Dong et al., 2025)	✓	✗	✓	✗	✗	ambiguous + unanswerable
BIRD-Interact (Huo et al., 2026)	✓	✓	✓	✓	✓	interactive, multi-phase

Table 9: Gap class coverage across text-to-SQL benchmarks. Δ_{sch} : schema linking required. Δ_{kb} : external knowledge base provided. Δ_{usr} : user clarification channel available. Δ_{exe} : execution feedback used during interaction. Δ_{ph} : follow-up questions referencing prior-phase results. BIRD-Interact is the only benchmark requiring all five acquisition channels.

contributes an intent token bag drawn from the question text and the user’s response.

Matching rules. Each class uses a dedicated matching function $\text{match}_c(r, a)$ between a required item r and an acquired item a :

- **Tables, KB IDs:** exact equality after lower-casing.
- **Columns:** exact match on table.column; a bare-column required entry matches any acquired entry sharing that column suffix.
- **KB terms, intent terms:** token-overlap match $|r_{\text{tok}} \cap a_{\text{tok}}| / |r_{\text{tok}}| \geq 0.5$.

Per-class and aggregate sufficiency. For each gap class c , let $\text{hit}_c(q) = |\{r \in I_c^*(q) : \exists a \in \mathcal{I}_t[T, c], \text{match}_c(r, a)\}|$ denote the number of required items matched by at least one acquired item. The per-class sufficiency score is

$$S_c(q) = \begin{cases} \text{hit}_c(q) / |I_c^*(q)| & \text{if } |I_c^*(q)| > 0, \\ 1 & \text{otherwise.} \end{cases} \quad (3)$$

The task-level score averages the five per-class ratios uniformly:

$$S(q) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} S_c(q), \quad (4)$$

$$\mathcal{C} = \{\text{tab, col, kb-id, kb-term, intent}\}.$$

This is equivalent to Eq. 2 in the main text. Both $S(q)$ and the pass/failure diagnostics use the gold required set, so part of their association follows from construction. We interpret the task-level trend as consistency with ISH. Uniform averaging ensures that each acquisition channel contributes equally regardless of class size; a task with many columns but few KB IDs is not dominated by column coverage alone. The per-class breakdown

$S_c(q)$ is reported separately in the residual analysis (Table 7) to attribute low overall scores to specific gap classes.

Phase-carry items. Phase-carry state (Δ_{ph}) is not scored by $S(q)$ because it is mechanically preserved by CTE materialization (Algorithm 3) rather than acquired through tool calls. The five computed classes cover the acquisition channels actively managed by the protocol.

E Per-Database Breakdown

Tables 11 and 12 report per-database P1 and Normalised Reward for PASCAL 35B under the anchor configuration. Performance varies substantially across databases (P1 range: 20.0%–82.4% on lite_300, 12.5%–71.4% on full_600), reflecting differences in KB complexity, the proportion of hidden knowledge entries, and the number of composite formulas each task requires.

F Failure-Mode Case Studies: Real Baselines vs PASCAL

This appendix complements the per-class sufficiency analysis in Section 5.4 with concrete per-task evidence on why existing text-to-SQL solutions fail on tasks that require information acquisition beyond schema alone.

Selection methodology. We intersect the P1-failure sets of four real baselines (DIN-SQL, MAC-SQL, CHESS, DAIL-SQL) on lite_300 with Qwen3.6-35B as the SQL backbone, and intersect with the P1-success set of PASCAL. The candidate pool is **112 tasks**. We pick five cases that span the gap-taxonomy classes (Δ_{kb} , Δ_{usr} , Δ_{sch} , Δ_{exe}) and their composites. For each case we report the

Symbol	Description
<i>Environment</i>	
$\mathcal{E}$	Interaction environment ($\mathcal{D}, \mathcal{M}, \mathcal{K}, \mathcal{U}, \mathcal{H}_0$)
$\mathcal{D}$	Relational database with execution engine
$\mathcal{M}$	Schema metadata (tables, columns, FKs)
$\mathcal{K}$	External knowledge base
$\mathcal{U}$	User (clarification source)
$\mathcal{H}_0$	Prior-phase state
<i>Interaction</i>	
q	Natural-language query
a_t	Action at turn t
o_t	Observation returned by the environment
h_t	Interaction history up to turn t
$c(a_t)$	Cost of action a_t (one environment interaction)
B	Interaction budget
σ	Submitted SQL solution
$\sigma^*(q)$	Gold SQL solution for task q
<i>Information sets</i>	
$\mathcal{I}$	Universe of information items
$\mathcal{I}^*(q)$	Required information set for task q
$\mathcal{I}_t$	Acquired information set at turn t
$g(a_t, o_t)$	Information items revealed by observation o_t
<i>Gap taxonomy</i>	
$\Delta_t(q)$	Information gap at turn t : $\mathcal{I}^*(q) \setminus \mathcal{I}_t$
Δ_{sch}	Schema gap (source: $\mathcal{M}$)
Δ_{kb}	Knowledge gap (source: $\mathcal{K}$)
Δ_{usr}	User-intent gap (source: $\mathcal{U}$)
Δ_{exe}	Execution gap (source: $\mathcal{D}$)
Δ_{ph}	Phase-carry gap (source: $\mathcal{H}_0$)
$s(i)$	Canonical source map: item $\rightarrow$ environment component
<i>Diagnostics</i>	
$S(q)$	Sufficiency score (Eq. 2): post-hoc coverage diagnostic
$S_c(q)$	Per-class coverage of gap class c
$\bar{S}_c$	Mean per-class coverage (averaged over task group)
<i>Priority and routing</i>	
$\text{need}_t(c)$	Routing signal: whether class c remains unresolved at turn t
$\text{cost}(c)$	Expected interaction steps to close one item from class c
$\text{priority}_t(c)$	$\text{need}_t(c)/\text{cost}(c)$: cost-normalized acquisition priority
<i>Retry breaker</i>	
H_t	Submission history up to turn t
e_t	Entry in H_t : error category + SQL hash
N_v	Stagnation threshold (identical SQL hashes)
N_e	Error-repeat threshold (same error category)
N_{cap}	Hard cap on total submissions
<i>Trajectory metrics</i>	
P1 / P2	Phase-1 / Phase-2 pass rate
Gen%	Fraction of actions on SQL generation + retry
Acq%	Fraction of actions on schema / KB / user tools
KB%	Subset of Acq% on KB retrieval
H	Tool entropy (Shannon, base 2)
InfoFail%	Tasks failing from insufficient information
GenFail%	Tasks failing despite sufficient information

Table 10: Notation used throughout the paper.

Database	n	P1%	P2%	Rwd
alien	17	35.3	35.3	35.29
archeology	17	47.1	35.3	43.53
credit	17	76.5	64.7	72.94
cross_db	17	58.8	41.2	53.53
crypto	17	52.9	23.5	44.12
cybermarket	17	52.9	47.1	51.18
disaster	17	64.7	47.1	59.41
fake	17	47.1	35.3	43.53
gaming	17	58.8	41.2	53.53
insider	17	52.9	41.2	49.41
mental	17	35.3	17.6	30.00
museum	17	41.2	29.4	37.65
news	17	52.9	41.2	49.41
polar	17	58.8	47.1	55.29
robot	15	20.0	20.0	20.00
solar	17	82.4	58.8	75.29
vaccine	15	33.3	20.0	29.33
virtual	15	40.0	13.3	32.00
Overall	300	51.0	37.0	46.80

Table 11: Per-database results on lite_300 (PASCAL 35B). Rwd = Normalised Reward (same scale as Table 4).

gap class established by inspecting PASCAL’s trajectory *up to the first P_1 -passing submit* only; later turns belong to the Phase-2 follow-up and are not credited.

Analytical framework. Each case study is structured around three analysis dimensions that link the per-task evidence to the paper’s diagnostic framework: (i) *required information decomposition*: the gold-derived $\mathcal{I}^*(q)$ broken down by gap class (Section 2.2), establishing which acquisition channels each task demands; (ii) *acquisition trajectory with mechanism attribution*: key turns in PASCAL’s trajectory annotated with the Section 4.3 mechanism that closed each gap, showing how the priority schedule (Section 4.1) sequences acquisition before generation; (iii) *ISH consistency analysis*: the connection between the observed failure modes and the Section 3 observations, showing at the per-task level that baselines commit before acquiring required evidence (Obs. 3).

Database	<i>n</i>	P1%	P2%	Rwd
archeology_scan	13	23.1	15.4	20.77
cold_chain_pharma	28	28.6	23.8	27.14
cross_border	29	24.1	6.9	18.97
crypto_exchange	30	56.7	46.7	53.67
cybermarket_pattern	30	66.7	56.7	63.67
disaster_relief	15	40.0	33.3	38.00
exchange_traded_funds	28	32.1	25.0	30.00
fake_account	30	41.4	24.1	36.21
households	29	34.5	27.6	32.41
hulushows	30	36.7	23.3	32.67
insider_trading	30	50.0	30.0	44.00
labor_certification	29	20.7	6.9	16.55
mental_health	26	23.1	11.5	19.62
museum_artifact	30	20.0	13.3	18.00
organ_transplant	32	12.5	9.4	11.56
planets_data	29	41.4	24.1	36.21
polar_equipment	30	33.3	20.0	29.33
reverse_logistics	28	71.4	60.7	68.21
robot_fault	17	35.3	35.3	35.29
solar_panel	30	53.3	30.0	46.33
sports_events	29	51.7	37.9	47.59
virtual_idol	28	35.7	28.6	33.57
Overall	600	38.3	26.8	34.84

Table 12: Per-database results on full_600 (PASCAL 35B). one task with unknown DB assignment is excluded from per-DB rows but included in the overall denominator.

F.1 Case 1: mental_2

This task operates on the mental database (8 tables, 595–1000 rows each), which models a mental health facility network. The query requires joining four tables (facilities, encounters, treatmentbasics, treatmentoutcomes) and accessing a JSONB column (support_and_resources) whose structure is not exposed by standard schema inspection. Three KB definitions are required (Table 13); one is intentionally hidden by the benchmark, forcing the agent to discover the formula empirically. Table 15 lists all retrieved KB entries; Table 16 traces the acquisition trajectory with mechanism attribution. This case primarily validates **Obs. 2** (KB channel produces the largest gain) and **Obs. 3** (the bottleneck is premature commitment, not SQL quality), demonstrating the typed KB retrieval mechanism (Section 4.3) and the $\Delta_{kb} \rightarrow \Delta_{exe}$ empirical fallback path (Section 4.3).

Gap class: Δ_{kb} (component KB) + Δ_{exe} (work-around on hidden KB).

Task. Database mental. User query:

For all facilities, I want to explore the connection between each site’s support level and its compliance rate. Include the overall support index as a reference and the correlation coefficient between each facility’s resource adequacy score and treatment adherence rate.

Three gap classes are active: Δ_{kb} dominates (3 KB IDs, 1 hidden), Δ_{usr} requires resolving ambiguous terms, and Δ_{sch} is non-trivial because FRAI accesses a JSONB column.

Gap class	Required items
Δ_{sch} (tab)	facilities, encounters, treatmentoutcomes, treatmentbasics
Δ_{kb} (id)	60 (CRAA), 5 (FRAI, <i>hidden</i>), 2 (TAR)
Δ_{kb} (term)	“Facility Resource Adequacy Index”
Δ_{usr} (intent)	“support level”, “compliance rate”

Table 13: Case 1 required information $\mathcal{I}^*(q)$.

Metric	Val.	Metric	Val.
P1-pass turn	51	KB found	6
Tool calls	52	KB hidden	1
submit attempts	4	ask_user	0
failed	3	exec_sql probes	30

Table 14: Case 1 PASCAL 35B acquisition summary (pre-P1).

T	ID	KB entry	Stat.
3	60	CRAA (correlation formula)	✓
4	5	FRAI (resource adequacy)	✗hid.
5	2	TAR (adherence rate)	✓
6	24	Community Resources (ordinal)	ctx.
7	28	Social Support Level (ordinal)	ctx.
28	9	MAR (missed appts)	ctx.
43	18	Non-Compliant Patient	ctx.

Table 15: Case 1 KB definitions retrieved. Required IDs marked ✓; id=5 is hidden and recovered via JSONB probing (turns 8–17).

Acquisition trajectory. Table 16 traces key turns, mapping each to a Section 4.3 mechanism. The trajectory follows the priority schedule: schema (turns 0–1) → KB (turns 2–7) → empirical probing (turns 8–44) → retry (turns 45–51). No ask_user is called.

Turn	Tool	Gap closed	Mechanism
0	explore_tables	Δ_{sch} (8 tables)	Schema pre-inj.
1	list_fkeys	Δ_{sch} (FK graph)	Schema pre-inj.
2	list_kb_names	— (namespace)	Namespace exp.
3	get_kb(CRAA)	Δ_{kb} (id=60)	Typed KB retr.
4	get_kb(FRAI)	$\times$ <i>hidden</i>	→ fallback
5	get_kb(TAR)	Δ_{kb} (id=2)	Typed KB retr.
6–7	get_kb $\times 2$	context	Typed KB retr.
8–17	exec_sql $\times 10$	Δ_{exe} (JSONB)	Empirical disc.
35	exec_sql	Δ_{exe} (ordinals)	Value probing
45	submit (fail)	576×6 vs 1×2	Retry: shape
46	submit (fail)	FRAI $1.73 \neq 1.75$	Value-diff fb.
47	exec_sql	Δ_{exe} (FRAI=1.75)	Empirical disc.
49	submit (fail)	Decimal $\neq$ float	Value-diff fb.
51	submit (✓)	P1 pass	—

Table 16: Case 1 (PASCAL 35B) acquisition trajectory. Turns 18–34 and 36–44 omitted (schema sampling / CTE construction).

Gold SQL.

```
WITH FacilityResources AS (
  SELECT fackey,
    CASE (support_and_resources->>'community_resources')
      WHEN 'Comprehensive' THEN 3
      WHEN 'Adequate' THEN 2
      WHEN 'Limited' THEN 1 ELSE 0
    END AS resource_score
  FROM facilities
  WHERE support_and_resources->>'community_resources'
    IS NOT NULL
), FacilityAdherence AS (
  SELECT e.facid,
    COUNT(*) AS total_outcomes,
    COUNT(*) FILTER (
      WHERE tro.txadh IN ('High', 'Medium')
    ) AS adherent_outcomes
  FROM treatmentoutcomes tro
  JOIN treatmentbasics tb ON tro.txref = tb.txkey
  JOIN encounters e ON tb.encref = e.enckey
  WHERE e.facid IS NOT NULL
  GROUP BY e.facid
), FacilityMetrics AS (
  SELECT fr.fackey, fr.resource_score,
    CASE WHEN fa.total_outcomes > 0
      THEN fa.adherent_outcomes::DECIMAL
        / fa.total_outcomes
    ELSE 0
    END AS tar
  FROM FacilityResources fr
  LEFT JOIN FacilityAdherence fa
    ON fr.fackey = fa.facid
), OverallFRAI AS (
```

```

SELECT AVG(resource_score) AS frai
FROM FacilityResources
)
SELECT (SELECT frai FROM OverallFRAI) AS overall_frai,
CORR(resource_score, tar) AS correlation_frai_tar
FROM FacilityMetrics WHERE tar > 0;

```

PASCAL P1-passing SQL (turn 51).

```

WITH facility_tar AS (
  SELECT f.fackey,
    CAST(SUM(CASE WHEN tout.txadh IN ('High',
      'Medium') THEN 1 ELSE 0 END) AS numeric)
    / COUNT(tout.txoutkey) AS tar
  FROM facilities f
  JOIN clinicians c ON c.facconnect = f.fackey
  JOIN patients p ON p.clinleadref = c.clinkey
  JOIN encounters e ON e.patref = p.patkey
  JOIN treatmentbasics tb ON tb.encref = e.enckey
  JOIN treatmentoutcomes tout
    ON tout.txref = tb.txkey
  GROUP BY f.fackey
),
facility_resource AS (
  SELECT f.fackey,
    CASE support_and_resources
      ->>'community_resources'
      WHEN 'Comprehensive' THEN 3
      WHEN 'Adequate' THEN 2
      WHEN 'Limited' THEN 1 ELSE 0
    END AS resource_score
  FROM facilities f
)
SELECT
  ROUND(AVG(CASE support_and_resources
    ->>'community_resources'
    WHEN 'Comprehensive' THEN 3
    WHEN 'Adequate' THEN 2
    WHEN 'Limited' THEN 1 ELSE 0
  END)::numeric, 2) AS overall_frai,
  CORR(fr.resource_score::double precision,
    ft.tar::double precision)
    AS correlation_frai_tar
FROM facilities f
JOIN facility_resource fr
  ON fr.fackey = f.fackey
JOIN facility_tar ft ON ft.fackey = fr.fackey;

```

Baseline SQL and failure analysis. *DIN-SQL* (exec error). Uses PostgreSQL reserved word to as alias for treatmentoutcomes — a Δ_{sch} failure that schema pre-injection would prevent.

```

WITH FacilityMetrics AS (
  SELECT f.fackey,
    CASE
      WHEN f.support_and_resources
        ->>'community_resources'
        = 'Comprehensive' THEN 3
      WHEN f.support_and_resources
        ->>'community_resources'
        = 'Adequate' THEN 2
      WHEN f.support_and_resources
        ->>'community_resources'
        = 'Limited' THEN 1
      ELSE 0
    END AS resource_score,
    COUNT(to.txoutkey) AS total_outcomes,
    COUNT(CASE WHEN to.txadh IN ('High',
      'Medium') THEN 1 END) AS adherent_count
  FROM facilities f
  LEFT JOIN encounters e ON f.fackey = e.facid
  LEFT JOIN treatmentbasics tb
    ON e.enckey = tb.encref
  LEFT JOIN treatmentoutcomes to
    ON tb.txkey = to.txref
  GROUP BY f.fackey, f.support_and_resources
),
FacilityRates AS (
  SELECT fackey, resource_score,
    adherent_count * 1.0
    / NULLIF(total_outcomes, 0) AS tar
  FROM FacilityMetrics
)

```

```

SELECT fackey, resource_score, tar,
  AVG(resource_score) OVER ()
    AS overall_support_index,
  (SELECT CORR(resource_score, tar)
    FROM FacilityRates)
    AS correlation_coefficient
FROM FacilityRates;

```

MAC-SQL (exec OK, P1 fail). Wraps the ordinal CASE inside AVG(), collapsing per-facility scores into a global aggregate. Without the FRAI KB definition, the model defaults to averaging — a Δ_{kb} failure.

```

WITH FacilityMetrics AS (
  SELECT f.fackey,
    AVG(CASE
      WHEN f.support_and_resources
        ->>'community_resources'
        = 'Comprehensive' THEN 3
      WHEN f.support_and_resources
        ->>'community_resources'
        = 'Adequate' THEN 2
      WHEN f.support_and_resources
        ->>'community_resources'
        = 'Limited' THEN 1
      ELSE 0
    END) AS resource_score
  FROM facilities f GROUP BY f.fackey
),
FacilityAdherence AS (
  SELECT f.fackey,
    CAST(SUM(CASE WHEN to1.txadh IN ('High',
      'Medium') THEN 1 ELSE 0 END) AS REAL)
    / COUNT(to1.txoutkey) AS tar
  FROM facilities f
  JOIN encounters e ON f.fackey = e.facid
  JOIN treatmentbasics tb
    ON e.enckey = tb.encref
  JOIN treatmentoutcomes to1
    ON tb.txkey = to1.txref
  GROUP BY f.fackey
),
CombinedMetrics AS (
  SELECT fm.resource_score, fa.tar
  FROM FacilityMetrics fm
  JOIN FacilityAdherence fa
    ON fm.fackey = fa.fackey
),
Stats AS (
  SELECT COUNT(*) AS n,
    SUM(resource_score) AS sum_x,
    SUM(tar) AS sum_y,
    SUM(resource_score * tar) AS sum_xy,
    SUM(resource_score * resource_score)
      AS sum_x2,
    SUM(tar * tar) AS sum_y2,
    AVG(resource_score) AS avg_x
  FROM CombinedMetrics
)
SELECT avg_x AS overall_support_index,
  CASE WHEN (n*sum_x2 - sum_x*sum_x)
    * (n*sum_y2 - sum_y*sum_y) = 0 THEN NULL
  ELSE (n*sum_xy - sum_x*sum_y)
    / SQRT((n*sum_x2 - sum_x*sum_x)
      * (n*sum_y2 - sum_y*sum_y))
  END AS correlation_coefficient
FROM Stats;

```

CHESS (exec OK, P1 fail). Filters WHERE fm.tar IS NOT NULL, excluding zero-adherence facilities. The gold uses ELSE 0 to include them — a Δ_{kb} failure.

```

WITH FacilityMetrics AS (
  SELECT f.fackey,
    CASE
      WHEN f.support_and_resources
        ->>'community_resources'
        = 'Comprehensive' THEN 3
      WHEN f.support_and_resources

```

```

-->'community_resources'
= 'Adequate' THEN 2
WHEN f.support_and_resources
-->'community_resources'
= 'Limited' THEN 1
ELSE 0
END AS resource_score,
COUNT(CASE WHEN to1.txadh IN ('High',
'Medium') THEN 1 END) * 1.0
/ NULLIF(COUNT(to1.txoutkey), 0) AS tar
FROM facilities f
LEFT JOIN encounters e ON f.fackey = e.facid
LEFT JOIN treatmentbasics tb
ON e.enckey = tb.enckey
LEFT JOIN treatmentoutcomes to1
ON tb.txkey = to1.txref
GROUP BY f.fackey,
f.support_and_resources
-->'community_resources'
),
OverallSupport AS (
SELECT AVG(CASE
WHEN support_and_resources
-->'community_resources'
= 'Comprehensive' THEN 3
WHEN support_and_resources
-->'community_resources'
= 'Adequate' THEN 2
WHEN support_and_resources
-->'community_resources'
= 'Limited' THEN 1
ELSE 0 END) AS overall_support_index
FROM facilities
)
SELECT
MAX(os.overall_support_index)
AS overall_support_index,
(COUNT(fm.tar)
* SUM(fm.resource_score * fm.tar)
- SUM(fm.resource_score) * SUM(fm.tar))
* 1.0
/ SQRT(
(COUNT(fm.tar)
* SUM(fm.resource_score
* fm.resource_score)
- SUM(fm.resource_score)
* SUM(fm.resource_score))
* (COUNT(fm.tar)
* SUM(fm.tar * fm.tar)
- SUM(fm.tar) * SUM(fm.tar)))
AS correlation_coefficient
FROM FacilityMetrics fm
CROSS JOIN OverallSupport os
WHERE fm.tar IS NOT NULL;

```

DAIL-SQL (exec error: empty submission — generation brittleness, outside acquisition scope).

Gap diagnosis. PASCAL acquires six KB definitions in the first 45 turns. The seventh lookup (FRAI) returns “not found”; PASCAL falls through to empirical JSONB probing rather than calling `ask_user`, recovering the formula by turn 47. The three failed submits (turns 45, 46, 49) each receive structured value-diff feedback: shape mismatch → output projection fix, value mismatch → FRAI denominator fix, type mismatch → explicit cast. The baselines lack KB access entirely: DIN-SQL hits a reserved-word parse error (Δ_{sch}); MAC-SQL and CHESS produce executable but semantically wrong SQL because they guess the FRAI/TAR encoding without the KB definitions (Δ_{kb}).

Consistency with ISH. This case validates Obs. 2 and Obs. 3. All four baselines commit to

SQL with kb-id coverage 0/3; failures trace to missing domain knowledge, not SQL quality (Obs. 3). PASCAL acquires 2/3 KB IDs directly and recovers the hidden third through the $\Delta_{kb} \rightarrow \Delta_{exe}$ fallback. Opening the KB channel is the decisive lever (Obs. 2): the same backbone that scores 0/4 under baselines passes P1 once KB definitions ground the CASE encoding and CORR aggregation.

F.2 Case 2: cross_db_4

This task operates on the `cross_db` database (2 key tables: `datapofile`, `securityprofile`). The query requires computing a Data Sensitivity Index (DSI) and a Security Robustness Score (SRS) as a JSON object, but neither formula is documented in the schema or retrievable from the KB before P1. Tables 17 and 19 show the required information; Table 20 traces the acquisition trajectory. This case validates **Obs. 3** (premature commitment) and demonstrates the informed clarification mechanism (Section 4.3): `ask_user` at turns 83 and 85 unlocks the SRS formula after 31 failed submissions.

Gap class: Δ_{usr} dominant (formula clarification after repeated execution failures).

Gap class	Required items
Δ_{sch} (tab)	<code>datapofile</code> , <code>securityprofile</code>
Δ_{kb} (id)	4 (DSI), 5 (SRS, <i>hidden</i>), 14
Δ_{kb} (term)	“Security Robustness Score”
Δ_{usr} (intent)	“sensitive data issues”, “JSON object”

Table 17: Case 2 required information $\mathcal{I}^*(q)$.

Metric	Val.	Metric	Val.
P1-pass turn	86	KB found (pre-P1)	0
Tool calls	106	KB hidden	0
submit attempts	34	<code>ask_user</code>	2
failed (pre-P1)	31	<code>exec_sql</code> probes	63

Table 18: Case 2 PASCAL 35B acquisition summary (pre-P1). No KB retrieval occurs before P1; all 7 KB calls happen during Phase-2 follow-up.

Acquisition trajectory. Table 20 traces PASCAL 35B’s key turns. Unlike Case 1, no KB retrieval precedes P1. The agent spends 80+ turns probing DSI/SRS formula variants via `execute_sql` and `submit_sql`, cycling through plausible but incorrect encodings. At turns 83 and 85, the retry breaker triggers `ask_user`, which returns the canonical SRS conditional structure. PASCAL passes P1 immediately at turn 86.

T	ID	KB entry	Stat.
<i>No KB calls before P1 (turns 0–86)</i>			
88	–	CBCE (cross-border compliance)	ctx.
89	–	CDVR (data volume risk)	ctx.
90	–	GDPR compliance	ctx.
91	–	CBRF (cross-border risk)	ctx.
92	–	RES (risk exposure)	ctx.
103	–	DRRS (data retention risk)	ctx.

Table 19: Case 2 KB definitions retrieved. All KB calls serve the Phase-2 follow-up; P1 is solved entirely through schema probing and user clarification.

Turn	Tool	Gap closed	Mechanism
0–10	exec_sql × 11	Δ_{sch} (schema)	Schema probing
11	submit (fail)	shape 5 × 1 vs 5 × 1	Value mismatch
12–82	exec/submit × 71	DSI/SRS variants	Retry cycle
83	ask_user	Δ_{usr} (DSI keys)	Inf. clarif.
85	ask_user	Δ_{usr} (SRS formula)	Inf. clarif.
86	submit (✓)	P1 pass	—

Table 20: Case 2 (PASCAL 35B) acquisition trajectory. Turns 12–82 omitted (DSI/SRS formula iteration). ask_user at turns 83/85 is the decisive mechanism.

Gold SQL.

```

SELECT
  JSON_BUILD_OBJECT(
    'ProfileTrace', dp.ProfileTrace,
    'DataSensitivityIndex', ROUND((dp.VolGB *
      CASE
        WHEN dp.DataSense = 'High' THEN 3
        WHEN dp.DataSense = 'Medium' THEN 2
        ELSE 1
      END)::numeric, 2),
    'SecurityRobustnessScore',
      CASE
        WHEN sp.EncState = 'Full'
          AND sp.AclState = 'Strong' THEN 3
        WHEN sp.EncState = 'Full'
          OR sp.AclState = 'Strong' THEN 2
        ELSE 1
      END
  ) AS exposure_details
FROM DataProfile dp
LEFT JOIN SecurityProfile sp
  ON dp.ProfileTrace = sp.ProfileKey
WHERE dp.VolGB *
  CASE WHEN dp.DataSense = 'High' THEN 3
        WHEN dp.DataSense = 'Medium' THEN 2
        ELSE 1
  END > 100
AND CASE
  WHEN sp.EncState = 'Full'
    AND sp.AclState = 'Strong' THEN 3
  WHEN sp.EncState = 'Full'
    OR sp.AclState = 'Strong' THEN 2
  ELSE 1
END < 2
ORDER BY dp.VolGB *
  CASE WHEN dp.DataSense = 'High' THEN 3
        WHEN dp.DataSense = 'Medium' THEN 2
        ELSE 1
  END DESC
LIMIT 5;

```

PASCAL P1-passing SQL (turn 86).

```

WITH profile_scores AS (
  SELECT dp.profiletrace,
    ROUND(CAST(dp.volgb *
      CASE WHEN dp.datasense = 'High' THEN 3
            WHEN dp.datasense = 'Medium' THEN 2
            WHEN dp.datasense = 'Low' THEN 1
            ELSE 0 END AS numeric), 2) AS dsi,

```

```

CASE
  WHEN sp.encstate = 'Full'
    AND sp.aclstate = 'Strong' THEN 3
  WHEN sp.encstate = 'Full'
    OR sp.aclstate = 'Strong' THEN 2
  ELSE 1
END AS srs
FROM dataprofile dp
JOIN securityprofile sp
  ON sp.profilekey = dp.profiletrace
WHERE dp.volgb *
  CASE WHEN dp.datasense = 'High' THEN 3
        WHEN dp.datasense = 'Medium' THEN 2
        WHEN dp.datasense = 'Low' THEN 1
        ELSE 0 END > 100
AND (CASE
  WHEN sp.encstate = 'Full'
    AND sp.aclstate = 'Strong' THEN 3
  WHEN sp.encstate = 'Full'
    OR sp.aclstate = 'Strong' THEN 2
  ELSE 1 END) < 2
)
SELECT json_build_object(
  'ProfileTrace', profiletrace,
  'DataSensitivityIndex', dsi,
  'SecurityRobustnessScore', srs
) AS exposure_details
FROM profile_scores
ORDER BY dsi DESC LIMIT 5;

```

Pre-P1 user-simulator dialogue.

PASCAL (turn 83): *The expected output shows DSI values like 2904 and 2903 for rows 1–2, but my calculation gives 29551 and 29259. Can you clarify: Should the DSI formula use volgb × multiplier where multiplier is based on datasense?*

User: *The JSON object should include ‘ProfileTrace’, ‘DataSensitivityIndex’, and ‘SecurityRobustnessScore’. The Data Sensitivity Index is calculated using ‘VolGB’ multiplied by a factor based on ‘DataSense’. The sorting is by Data Sensitivity Index from highest to lowest.*

PASCAL (turn 85): *Can you clarify the exact Security Robustness Score (SRS) formula? What values should EncState and AclState have for each SRS level (1, 2, 3)?*

User: *SRS of 3 is assigned when both EncState and AclState are ‘Full’ and ‘Strong’, respectively. SRS of 2 is assigned if either EncState is ‘Full’ or AclState is ‘Strong’.*

Baseline SQL and failure analysis. *DIN-SQL* (exec error). Uses json_object() which does not exist in PostgreSQL with this signature; the correct function is json_build_object(). Also uses ‘Adequate’ for SRS level 2 instead of ‘Strong’ — a Δ_{usr} failure.

```

SELECT
  json_object(
    'id', T1.profiletrace,
    'sensitivity_metric', T1.volgb * CASE
      WHEN T1.datasense = 'High' THEN 3
      WHEN T1.datasense = 'Medium' THEN 2
      ELSE 1 END,
    'security_robustness_score', CASE
      WHEN T2.encstate = 'Full'
        AND T2.aclstate = 'Strong' THEN 3
      WHEN T2.encstate = 'Full'
        OR T2.aclstate = 'Adequate' THEN 2
      ELSE 1 END
  ) AS result
FROM dataprofile AS T1
INNER JOIN securityprofile AS T2
  ON T1.profiletrace = T2.profilekey
WHERE (T1.volgb * CASE

```

```

    WHEN T1.datasense = 'High' THEN 3
    WHEN T1.datasense = 'Medium' THEN 2
    ELSE 1 END) > 100
AND (CASE
    WHEN T2.encstate = 'Full'
        AND T2.aclstate = 'Strong' THEN 3
    WHEN T2.encstate = 'Full'
        OR T2.aclstate = 'Adequate' THEN 2
    ELSE 1 END) < 2
ORDER BY T1.volgb * CASE
    WHEN T1.datasense = 'High' THEN 3
    WHEN T1.datasense = 'Medium' THEN 2
    ELSE 1 END DESC
LIMIT 5;

```

MAC-SQL (exec OK, P1 fail). Same 'Adequate' error in SRS encoding; also uses non-standard JSON key names ('id' instead of 'ProfileTrace') — a Δ_{usr} failure.

```

SELECT
  json_build_object(
    'id', T1.profiletrace,
    'sensitivity_metric', T1.dsi_calc,
    'security_robustness_score', T1.srs_calc
  ) AS result
FROM (
  SELECT T1.profiletrace,
    CASE WHEN T1.datasense = 'High'
      THEN T1.volgb * 3
    WHEN T1.datasense = 'Medium'
      THEN T1.volgb * 2
    ELSE T1.volgb * 1
    END AS dsi_calc,
    CASE
      WHEN T2.encstate = 'Full'
        AND T2.aclstate = 'Strong' THEN 3
      WHEN T2.encstate = 'Full'
        OR T2.aclstate = 'Adequate' THEN 2
      ELSE 1
    END AS srs_calc
  FROM dataprofile AS T1
  INNER JOIN securityprofile AS T2
    ON T1.profiletrace = T2.profilekey
  WHERE T1.datasense IS NOT NULL
    AND T2.encstate IS NOT NULL
    AND T2.aclstate IS NOT NULL
) AS T1
WHERE T1.dsi_calc > 100 AND T1.srs_calc < 2
ORDER BY T1.dsi_calc DESC, T1.srs_calc ASC
LIMIT 5;

```

CHESS (exec OK, P1 fail). Identical 'Adequate' error; all three non-trivial baselines independently guess the same wrong SRS branch, confirming this is a systematic Δ_{usr} failure.

```

SELECT json_build_object(
  'profile_id', T1.profiletrace,
  'sensitivity_metric', T1.DSI,
  'security_robustness_score', T1.SRS)
FROM (
  SELECT T1.profiletrace,
    T1.volgb * CASE T1.datasense
      WHEN 'High' THEN 3
      WHEN 'Medium' THEN 2
      ELSE 1 END AS DSI,
    CASE
      WHEN T2.encstate = 'Full'
        AND T2.aclstate = 'Strong' THEN 3
      WHEN T2.encstate = 'Full'
        OR T2.aclstate = 'Adequate' THEN 2
      ELSE 1 END AS SRS
  FROM dataprofile AS T1
  INNER JOIN securityprofile AS T2
    ON T1.profiletrace = T2.profilekey
  WHERE (T1.volgb * CASE T1.datasense
    WHEN 'High' THEN 3 WHEN 'Medium' THEN 2
    ELSE 1 END) > 100
  AND (CASE
    WHEN T2.encstate = 'Full'
      AND T2.aclstate = 'Strong' THEN 3

```

```

    WHEN T2.encstate = 'Full'
      OR T2.aclstate = 'Adequate' THEN 2
    ELSE 1 END) < 2
) AS T1
ORDER BY T1.DSI DESC LIMIT 5;

```

DAIL-SQL (exec error). Joins with the wrong table (riskmanagement) and uses compscore as SRS — a generation failure.

Gap diagnosis. PASCAL's first 80+ turns cycle through DSI/SRS formula variants via `execute_sql` without converging. At turns 83 and 85, the informed clarification mechanism triggers: PASCAL asks the user for (i) the JSON key names and DSI rounding, and (ii) the exact SRS conditional structure. The user confirms that SRS = 2 requires `AclState = 'Strong'`, not 'Adequate'. PASCAL submits at turn 86 and passes immediately. All three non-trivial baselines independently produce the same 'Adequate' branch — a systematic Δ_{usr} failure that no amount of SQL retry can resolve without user clarification.

Consistency with ISH. This case validates Obs. 3 in its purest form: all four baselines commit to SQL without clarifying the SRS formula; their SQLs are structurally correct but semantically wrong due to a single-value ambiguity ('Adequate' vs 'Strong'). PASCAL's 31 failed submits before `ask_user` show that the retry breaker alone is insufficient for Δ_{usr} gaps — the informed clarification mechanism (Section 4.3) is the decisive lever. This is the only case where no KB retrieval occurs before P1, isolating the clarification channel's contribution.

F.3 Case 3: museum_3

This task operates on the museum database (12 tables). The query asks to “rank all artifacts to identify which need urgent care” using a Conservation Priority Index (CPI) whose formula is hidden in the KB. Tables 21–24 detail the required information, KB retrieval, and trajectory. This case validates **Obs. 1** (undirected tool use wastes budget) and **Obs. 2** (KB channel is the highest-leverage source), demonstrating the namespace exposure mechanism (Section 4.3).

Gap class: Δ_{kb} (component KB) + Δ_{sch} (schema sampling).

Acquisition trajectory. Table 24 shows PASCAL 35B's path. When CPI returns “not found” (turn 2), PASCAL retrieves three component entries (turns 3–5), samples column values

Gap class	Required items
Δ_{sch} (tab)	artifactscore, artifactratings
Δ_{kb} (id)	0 (CPI, <i>hidden</i>)
Δ_{kb} (term)	“Conservation Priority Index”
Δ_{usr} (intent)	“rank all artifacts”, “urgent care”

Table 21: Case 3 required information $\mathcal{I}^*(q)$.

Metric	Val.	Metric	Val.
P1-pass turn	16	KB found	3
Tool calls	109	KB hidden	1
submit attempts	31	ask_user (pre-P1)	0
failed (pre-P1)	0	exec_sql probes	5

Table 22: Case 3 PASCAL 35B acquisition summary. P1 passes on the first submit at turn 16.

(turns 6–10), probes the formula via `execute_sql` (turns 11–15), and submits at turn 16 — passing on the first attempt.

Gold SQL.

```
WITH status_mapping AS (
  SELECT ArtRegistry, ArtName, ArtDynasty,
  CASE ConserveStatus
    WHEN 'Excellent' THEN 1
    WHEN 'Good' THEN 3
    WHEN 'Fair' THEN 5
    WHEN 'Poor' THEN 7
    WHEN 'Critical' THEN 10
  END AS status_score
FROM ArtifactsCore
)
SELECT a.ArtRegistry, a.ArtName, a.ArtDynasty,
((r.HistSignRating + r.ResearchValRating
+ r.CultScore) * (10 - s.status_score))
/ 30.0 AS CPI
FROM status_mapping s
JOIN ArtifactRatings r
ON s.ArtRegistry = r.ArtRef
JOIN ArtifactsCore a
ON s.ArtRegistry = a.ArtRegistry
ORDER BY CPI DESC;
```

PASCAL P1-passing SQL (turn 16).

```
SELECT
a.artregistry AS "Artifact ID",
a.artname AS "name",
a.artdynasty AS "historical era",
ROUND(CAST(
(r.histsignrating + r.researchvalrating
+ r.cultscore) * (10 -
CASE a.conservestatus
  WHEN 'Excellent' THEN 1
  WHEN 'Good' THEN 3
  WHEN 'Fair' THEN 5
  WHEN 'Poor' THEN 7
  WHEN 'Critical' THEN 10
END) AS numeric) / 30, 2) AS "score"
FROM artifactscore a
JOIN artifactratings r
ON a.artregistry = r.artref
ORDER BY score DESC;
```

Baseline SQL and failure analysis. *DIN-SQL* (exec OK, P1 fail). Multiplies CPI by a sensitivity-weight factor from `sensitivitydata` — a component that is *not* part of the gold CPI formula. Without the CPI KB definition, the model hallucinates additional factors — a Δ_{kb} failure.

```
WITH ArtifactData AS (
  SELECT a.artregistry, a.artname, a.artdynasty,
  r.histsignrating, r.researchvalrating,
  r.cultscore, a.conservestatus,
```

T	ID	KB entry	Stat.
2	0	CPI (composite formula)	$\times$ hid.
3	–	ConserveStatus (ordinal)	✓
4	–	HistSignRating	✓
5	–	Conservation Priority Level	✓

Table 23: Case 3 KB definitions retrieved (pre-P1). CPI is hidden; PASCAL retrieves its three component definitions and reconstructs the formula.

Turn	Tool	Gap closed	Mechanism
0	explore_tables	Δ_{sch}	Schema pre-inj.
1	list_kb_names	— (namespace)	Namespace exp.
2	get_kb(CPI)	$\times$ <i>hidden</i>	→ fallback
3–5	get_kb $\times 3$	Δ_{kb} (components)	Typed KB retr.
6–10	sample_values $\times 5$	Δ_{sch} (ordinals)	Schema probing
11–15	exec_sql $\times 5$	Δ_{exe} (formula)	Empirical disc.
16	submit (✓)	P1 pass	—

Table 24: Case 3 (PASCAL 35B) acquisition trajectory. First-attempt P1 pass at turn 16.

```
s.envsensitivity, s.lightsensitivity,
s.tempsensitivity, s.humiditysensitivity,
CASE a.conservestatus
  WHEN 'Excellent' THEN 1 WHEN 'Good' THEN 3
  WHEN 'Fair' THEN 5 WHEN 'Poor' THEN 7
  WHEN 'Critical' THEN 10 ELSE 0
END AS conserve_status_val,
CASE s.envsensitivity WHEN 'Low' THEN 1
  WHEN 'Medium' THEN 5 WHEN 'High' THEN 10
  ELSE 0 END AS env_weight,
CASE s.lightsensitivity WHEN 'Low' THEN 1
  WHEN 'Medium' THEN 5 WHEN 'High' THEN 10
  ELSE 0 END AS light_weight,
CASE s.tempsensitivity WHEN 'Low' THEN 1
  WHEN 'Medium' THEN 5 WHEN 'High' THEN 10
  ELSE 0 END AS temp_weight,
CASE s.humiditysensitivity WHEN 'Low' THEN 1
  WHEN 'Medium' THEN 5 WHEN 'High' THEN 10
  ELSE 0 END AS humid_weight
FROM artifactscore a
JOIN artifactratings r ON a.artregistry = r.artref
JOIN sensitivitydata s ON a.artregistry = s.artref
)
SELECT artregistry, artname, artdynasty,
(((histsignrating + researchvalrating
+ cultscore) * (10 - conserve_status_val))
/ 30.0)
* ((env_weight + light_weight
+ temp_weight + humid_weight) / 4.0)
AS score
FROM ArtifactData ORDER BY score DESC;
```

MAC-SQL (exec OK, P1 fail). Same sensitivity-weight multiplication; also uses MAX instead of the raw CPI and adds a RANK column not in the gold output — Δ_{kb} .

```
SELECT A.artregistry, A.artname, A.artdynasty,
((R.histsignrating + R.researchvalrating
+ R.cultscore) * (10.0 - CASE A.conservestatus
  WHEN 'Excellent' THEN 1 WHEN 'Good' THEN 3
  WHEN 'Fair' THEN 5 WHEN 'Poor' THEN 7
  WHEN 'Critical' THEN 10 ELSE 0 END) / 30.0)
* ((CASE S.envsensitivity
  WHEN 'High' THEN 10 WHEN 'Medium' THEN 5
  WHEN 'Low' THEN 1 ELSE 0 END
+ ... + humid) / 4.0) AS score
FROM artifactscore AS A
JOIN artifactratings AS R ON A.artregistry = R.artref
JOIN sensitivitydata AS S ON A.artregistry = S.artref
ORDER BY score DESC;
```

CHESS (exec OK, P1 fail). Identical sensitivity-factor error in single-line SQL — Δ_{kb} failure.

DAIL-SQL (exec error). Uses `condassessscore / 5`

as CPI — completely wrong formula.

Gap diagnosis. When the composite CPI entry returns “not found,” PASCAL retrieves three documented component entries (ConserveStatus ordinal, HistSignRating, Conservation Priority Level), samples column values to confirm scales, and reconstructs the formula. All three non-trivial baselines hallucinate an extra sensitivity-weight factor — a systematic Δ_{kb} failure where the model fills in plausible-but-wrong components without KB grounding.

Consistency with ISH. This case validates Obs. 1 and Obs. 2. The baselines have the correct CPI *skeleton* (ratings $\times$ status) but multiply by an extraneous factor, producing executable but semantically wrong SQL. PASCAL avoids this by grounding the formula in KB-retrieved component definitions rather than parametric guesses. The first-attempt P1 pass at turn 16 demonstrates that targeted KB acquisition can resolve a task in fewer turns than any baseline’s single SQL attempt.

F.4 Case 4: archeology_1

This task operates on the archeology database (3 key tables: scans, scanpointcloud, sites). It requires computing a Scan Quality Score (SQS) whose composite formula is hidden in the KB and must be clarified via ask_user. Tables 25–28 detail the analysis. This case validates **Obs. 2** (KB as highest-leverage source) and **Obs. 3** (pre-mature commitment), demonstrating both typed KB retrieval (Section 4.3) and informed clarification (Section 4.3).

Gap class: Δ_{kb} (hidden composite formula) + Δ_{usr} (formula clarification).

Gap class	Required items
Δ_{sch} (tab)	scans, scanpointcloud, sites
Δ_{kb} (id)	3 (SQS, <i>hidden</i>)
Δ_{kb} (term)	“Scan Quality Score”
Δ_{usr} (intent)	“scan quality”

Table 25: Case 4 required information $\mathcal{I}^*(q)$.

Metric	Val.	Metric	Val.
P1-pass turn	80	KB found	6
Tool calls	114	KB hidden	1
submit attempts	7	ask_user	2
failed (pre-P1)	2	exec_sql probes	85

Table 26: Case 4 PASCAL 35B acquisition summary (pre-P1).

T	ID	KB entry	Stat.
11	–	Scan Resolution Index (SRI)	✓
12	–	Scan Coverage Effectiveness (SCE)	✓
13	–	NoiseDb (noise level)	ctx.
14	–	PointDense (point density)	ctx.
15	–	CoverPct (coverage %)	ctx.
16	–	ScanResolMm (resolution)	ctx.
76	3	Scan Quality Score (SQS)	✗hid.

Table 27: Case 4 KB definitions retrieved. Six component entries found; the composite SQS is hidden and obtained via ask_user at turn 77.

Turn	Tool	Gap closed	Mechanism
0–10	schema + exec $\times 11$	Δ_{sch}	Schema probing
11–16	get_kb $\times 6$	Δ_{kb} (components)	Typed KB retr.
26	submit (fail)	wrong output shape	Retry: shape
27	ask_user	Δ_{usr} (column names)	Inf. clarif.
29	submit (fail)	wrong SQS values	Value-diff fb.
76	get_kb(SQS)	$\times$ <i>hidden</i>	$\rightarrow$ fallback
77	ask_user	Δ_{usr} (SQS formula)	Inf. clarif.
80	submit (✓)	P1 pass	—

Table 28: Case 4 (PASCAL 35B) acquisition trajectory. Turns 17–25 and 30–75 omitted (SQS formula probing).

Acquisition trajectory.

Pre-P1 user-simulator dialogue.

PASCAL (turn 27): *The expected result has 3 columns. Could you clarify what the 3 columns should be?*

User: *The three columns should be site_code, site_name, and avg_sqs (average Scan Quality Score). The ranking is applied to the avg_sqs column.*

PASCAL (turn 77): *The expected avg_sqs values are around 1388, but my calculations using the documented SQS formula give different results. Can you confirm the exact SQS aggregation?*

User: *The formula is: ROUND(AVG(POWER(10.0 / (LOG(10, spc.scanresolmm * 1000.0) / LOG(10, spc.pointdense))) * 5.0, 1.5) * (spc.coverpct / 100.0) * POWER(1.0 - spc.noisedb / 30.0, 2)), 2).*

Gold SQL.

```
SELECT s.zoneref AS site_code,
       si.zonelabel AS site_name,
       ROUND(AVG(
         POWER(10.0 / (LOG(10, spc.scanresolmm
           * 1000.0) / LOG(10, spc.pointdense))
           * 5.0, 1.5)
         * (spc.coverpct / 100.0)
         * POWER(1.0 - spc.noisedb / 30.0, 2)
       ), 2) AS avg_sqs
FROM scans s
JOIN scanpointcloud spc ON s.arcref = spc.arcref
JOIN sites si ON s.zoneref = si.zoneregistry
GROUP BY s.zoneref, si.zonelabel
ORDER BY avg_sqs DESC;
```

PASCAL P1-passing SQL (turn 80).

```
SELECT sc.zoneref AS site_code,
       si.zonelabel AS site_name,
       ROUND(CAST(AVG(
         POWER(10.0 / (LOG(10, spc.scanresolmm
```

```

    * 1000.0) / LOG(10, sp.pointdense))
    * 5.0, 1.5)
    * (sp.coverpct / 100.0)
    * POWER(1.0 - sp.noisedb / 30.0, 2))
  AS numeric), 2) AS avg_sqs
FROM scans sc
JOIN scanpointcloud sp ON sc.arceref = sp.arceref
JOIN sites si ON sc.zoneref = si.zoneregistry
GROUP BY sc.zoneref, si.zoneref
ORDER BY avg_sqs DESC;

```

Baseline SQL and failure analysis. *DIN-SQL* (exec error). Joins with non-existent table scanspatial and references non-existent column T2.zoneref. Uses LOG10 (not valid in PostgreSQL) instead of LOG(10, ...) — a Δ_{sch} failure.

```

SELECT T3.zoneregistry, T3.zoneref,
(Power(10.0 / ((LOG10(T1.scanresolmm
* 1000.0) / LOG10(T1.pointdense))
* 5.0), 1.5))
* ((T1.coverpct * (1.0 + (T1.lappct / 100.0)
* (1.0 - (T1.coverpct / 100.0)))) / 100.0)
* POWER(1.0 - (T1.noisedb / 30.0), 2)
AS scan_quality
FROM scanpointcloud T1
JOIN scanspatial T2 ON T1.arceref = T2.arceref
JOIN sites T3 ON T2.zoneref = T3.zoneregistry
ORDER BY scan_quality DESC;

```

MAC-SQL (exec OK, P1 fail). Includes lappct (overlap) in the SCE sub-formula and uses MAX instead of AVG for aggregation — Δ_{kb} failure (wrong SQS sub-components).

CHESS (exec OK, P1 fail). Uses LN (natural log) instead of LOG(10, ...) (log base 10) and includes the lappct overlap factor — Δ_{kb} failure.

DAIL-SQL (exec error). Uses uvmapqual from scanmesh as a categorical proxy for “scan quality” — completely wrong approach.

Gap diagnosis. PASCAL retrieves six component KB entries (SRI, SCE, NoiseDb, PointDense, CoverPct, ScanResolMm). The composite SQS is hidden. After extensive probing and two failed submits, ask_user at turn 77 returns the exact aggregation formula; PASCAL passes at turn 80. The baselines compute SQS variants that include documented sub-components but diverge on the SCE / overlap encoding, the log base (LN vs LOG(10,...)), and the aggregation function.

Consistency with ISH. This case validates Obs. 2 and Obs. 3. PASCAL follows the $\Delta_{kb} \rightarrow \Delta_{usr}$ escalation path: component KB entries first, then ask_user when the composite formula remains unresolved. The two ask_user calls (turns 27 and 77) demonstrate the informed clarification mechanism: each question references previously acquired KB components rather than asking from scratch.

F.5 Case 5: cybermarket_1

This task operates on the cybermarket database (2 key tables: riskanalysis, transactions). The query asks to track “suspicious chains” using a Transaction Chain Risk (TCR) formula hidden in the KB. Unlike Cases 1–4, PASCAL solves this task through schema exploration alone (no KB retrieval, no ask_user), passing P1 at turn 10. This case validates **Obs. 1** (how budget is spent matters more than how much) and demonstrates that schema pre-injection (Section 4.3) can sometimes close Δ_{kb} gaps when column names are sufficiently descriptive.

Gap class: $\Delta_{exe}/\Delta_{sch}$ (schema-inferred formula on hidden KB).

Gap class	Required items
Δ_{sch} (tab)	riskanalysis, transactions
Δ_{kb} (id)	16 (TCR, <i>hidden</i>), 23
Δ_{kb} (term)	“Transaction Chain Risk”
Δ_{usr} (intent)	“suspicious chains”, “chain risk value”, “top risky ones”

Table 29: Case 5 required information $\mathcal{I}^*(q)$.

Metric	Val.	Metric	Val.
P1-pass turn	10	KB found (pre-P1)	0
Tool calls	19	KB hidden	1
submit attempts	3	ask_user	0
failed (pre-P1)	1	exec_sql probes	6

Table 30: Case 5 PASCAL 35B acquisition summary. The shortest case: 10 turns to P1 with only 1 failed submit.

T	ID	KB entry	Stat.
3	16	Transaction Chain Risk	χ_{hid} .
<i>Turns 11+ (Phase-2 only):</i>			
11	–	Communication Security Risk	χ_{hid} .
14–16	–	CSR, langpattern, alertsev	ctx.

Table 31: Case 5 KB definitions. TCR is hidden; PASCAL infers the formula from schema column names. KB calls at turns 11+ serve Phase-2.

Acquisition trajectory.

Gold SQL.

```

WITH risky_chains AS (
  SELECT r.riskregistry,
    r.txchainlen, r.linkedtxcount,
    r.wallage, r.wallturnrt,
    (r.txchainlen * 10)
    + (r.linkedtxcount * 5)
    + (r.fraudprob * 100)
    - (r.profilecomplete * 0.5)
    - (r.idverifyscore * 0.5)
  AS transaction_chain_risk

```

Turn	Tool	Gap closed	Mechanism
0	explore_tables	Δ_{sch} (tables)	Schema pre-inj.
1	list_kb_names	— (namespace)	Namespace exp.
2	list_fkeys	Δ_{sch} (FK graph)	Schema pre-inj.
3	get_kb(TCR)	$\times$ <i>hidden</i>	→ fallback
4–6	exec_sql $\times 3$	Δ_{exe} (column values)	Schema probing
7	submit (fail)	wrong row count	Retry: shape
8–9	exec_sql $\times 2$	Δ_{exe} (filter values)	Empirical disc.
10	submit (✓)	P1 pass	—

Table 32: Case 5 (PASCAL 35B) acquisition trajectory. The agent infers the TCR formula and laundering filters from schema column names in 10 turns.

```
FROM riskanalysis r
)
SELECT rc.riskregistry,
       rc.transaction_chain_risk,
       t.txregistry, t.payamtusd
FROM risky_chains rc
JOIN riskanalysis ra
  ON rc.riskregistry = ra.riskregistry
JOIN transactions t ON ra.txref = t.txregistry
WHERE rc.transaction_chain_risk > 150
      AND rc.wallage < 30
      AND rc.wallturnrt > 5
      AND rc.linkedtxcount >= 3
ORDER BY rc.transaction_chain_risk DESC;
```

PASCAL P1-passing SQL (turn 10).

```
SELECT ra.riskregistry,
       ROUND(CAST(
         (ra.txchainlen * 10)
         + (ra.linkedtxcount * 5)
         + (ra.fraudprob * 100)
         - (ra.profilecomplete * 0.5)
         - (ra.idverifyscore * 0.5)
       AS numeric), 2) AS chain_risk_value,
       t.txregistry, t.payamtusd
FROM riskanalysis ra
JOIN transactions t
  ON ra.txref = t.txregistry
WHERE (ra.txchainlen * 10)
      + (ra.linkedtxcount * 5)
      + (ra.fraudprob * 100)
      - (ra.profilecomplete * 0.5)
      - (ra.idverifyscore * 0.5) > 150
      AND ra.wallage < 30
      AND ra.wallturnrt > 5
      AND ra.linkedtxcount >= 3
ORDER BY chain_risk_value DESC;
```

Baseline SQL and failure analysis. *DIN-SQL* (exec OK, P1 fail). Computes the correct TCR weighted formula but omits all four laundering filters (wallage, wallturnrt, linkedtxcount, TCR > 150) — a Δ_{kb} failure (KB#23 Money Laundering Indicator unknown).

```
SELECT T1.riskregistry,
       (T1.txchainlen * 10)
       + (T1.linkedtxcount * 5)
       + (T1.fraudprob * 100)
       - (T1.profilecomplete * 0.5)
       - (T1.idverifyscore * 0.5)
       AS chain_risk_value,
       T2.txregistry, T2.payamtusd
FROM riskanalysis AS T1
INNER JOIN transactions AS T2
  ON T1.txref = T2.txregistry
ORDER BY chain_risk_value DESC;
-- missing: WHERE TCR > 150
-- AND wallage < 30 AND wallturnrt > 5
-- AND linkedtxcount >= 3
```

MAC-SQL (exec OK, P1 fail). Same missing filters; also adds LIMIT 10 (gold has no limit) — Δ_{kb} failure.

CHESS (exec OK, P1 fail). Identical: correct formula, no laundering filters — Δ_{kb} failure.

DAIL-SQL (exec error). Uses txchainlen directly as “chain risk value” and fraudprob for ordering — wrong formula.

Gap diagnosis. The TCR KB entry (id=16) is hidden, but the riskanalysis table’s column names (txchainlen, linkedtxcount, fraudprob, profilecomplete, idverifyscore, wallage, wallturnrt) are descriptive enough for PASCAL to infer the weighted composite formula via explore_tables. After one failed submit (wrong row count), PASCAL discovers the four laundering filters (wallage < 30, wallturnrt > 5, linkedtxcount >= 3, TCR > 150) through value-diff feedback and passes at turn 10. The baselines compute the formula correctly but uniformly miss the laundering filters because KB#23 (Money Laundering Indicator) was never retrieved.

Consistency with ISH. This case validates Obs. 1. All baselines produce the correct TCR formula from schema column names — the formula itself is inferable — but none applies the domain-specific laundering filters that determine the output cardinality. PASCAL discovers these filters in 2 turns of execute_sql probing (turns 8–9) after one failed submit, demonstrating that structured retry feedback (the shape-mismatch signal) can close remaining Δ_{exe} gaps even when KB retrieval returns nothing.

F.6 Case 6: museum_6 (Protocol comparison)

This is the motivating example from Section 1 (Fig. 1) and Section 4.3. Unlike Cases 1–5 (where baselines fail and PASCAL passes), both PASCAL and the official ReACT baseline pass P1 and P2 on this task. The analytical value lies in the *acquisition path*: PASCAL closes the KB gap in 2 turns via namespace exposure, while ReACT discovers the same threshold through 15 submits of trial-and-error SQL. Table 35 compares the two trajectories head-to-head. This case validates **Obs. 1** (how the budget is spent matters more than how much is spent) and demonstrates the end-to-end priority schedule from schema grounding through KB retrieval to phase-carry (Section 4.1).

Gap class: Δ_{sch} (schema mapping) + Δ_{kb} (KB threshold) + Δ_{ph} (phase-carry).

Task. Database museum (12 tables). Phase-1 query:

Could you find all the different IDs of showcases experiencing Environmental Instability Events?

Phase-2 follow-up:

For each of those showcases with an Environmental Instability Event, list the showcase ID together with a comma-separated list of the artifact IDs currently on display in it.

Gap class	Required items
Δ_{sch} (tab)	EnvironmentalReadingsCore, UsageRecords (Phase-2)
Δ_{kb} (id)	13 (Environmental Instability Event)
Δ_{kb} (term)	“Environmental Instability”
Δ_{ph}	Phase-1 result set (“those showcases”)

Table 33: Case 6 required information $\mathcal{I}^*(q)$. This is the only case with all three acquisition classes (Δ_{sch} , Δ_{kb} , Δ_{ph}) active simultaneously.

T	KB entry
1	Environmental Instability Event (id=13): “TempVar24h > 1 OR HumVar24h > 3”

Table 34: Case 6 KB definition retrieved. A single lookup at turn 1 provides the exact threshold.

Head-to-head comparison. Table 35 contrasts the two protocols on the same task and backbone (Qwen3.6-35B).

Metric	PASCAL	ReACT
Total tool calls	31	34
submit_sql calls	5	15
execute_sql calls	21	13
KB tool calls	2	0
ask_user calls	0	1
P1 pass at turn	6	29
P2 pass at turn	30	33
P1 / P2	✓ / ✓	✓ / ✓
Reward	1.0	1.0
Elapsed (s)	350	503

Table 35: Case 6 head-to-head: PASCAL vs official ReACT (same 35B backbone). Both achieve full reward, but PASCAL reaches P1 in 6 turns vs ReACT’s 29.

PASCAL acquisition trajectory.

ReACT acquisition path. The official ReACT baseline does not expose KB tools. The agent starts with get_schema (turn 0), then spends turns 1–5

Turn	Tool	Gap closed	Mechanism
0	list_kb_names	— (namespace)	Namespace exp.
1	get_kb(Instability)	Δ_{kb} (id=13)	Typed KB retr.
2	explore_tables	Δ_{sch} (12 tables)	Schema pre-inj.
3	exec_sql (LIMIT 5)	Δ_{sch} (columns)	Schema probing
4	exec_sql (query)	verify result	Empirical disc.
5	exec_sql (COUNT)	529 rows	Sanity check
6	submit (✓)	P1 pass	—
7–13	exec/sample $\times 7$	Δ_{sch} (Phase-2 tables)	Schema probing
14–29	exec/submit $\times 16$	Δ_{ph} (join path)	Phase carry
30	submit (✓)	P2 pass	—

Table 36: Case 6 (PASCAL 35B) acquisition trajectory. KB retrieval at turn 1 provides the exact threshold *before any SQL is written*.

probing column meanings. Without the KB threshold, it submits candidate SQL with guessed conditions (WHERE tempc > 30, WHERE tempvar24h > 5, etc.) and iterates via execution feedback until the correct threshold emerges at turn 29. The 15 submit_sql calls (vs PASCAL’s 5) directly reflect the cost of discovering domain knowledge through trial-and-error rather than targeted retrieval.

Gold SQL (Phase 1).

```
SELECT DISTINCT ShowcaseRef
FROM EnvironmentalReadingsCore
WHERE TempVar24h > 1 OR HumVar24h > 3;
```

PASCAL P1-passing SQL (turn 6).

```
SELECT DISTINCT showcaseref
FROM environmentalreadingscore
WHERE tempvar24h > 1 OR humvar24h > 3;
```

PASCAL P2-passing SQL (turn 30).

```
SELECT e.showcaseref,
STRING_AGG(u.artrefused, ', '
ORDER BY u.artrefused)
AS artifacts_on_display
FROM environmentalreadingscore e
JOIN usagerecords u
ON e.showcaseref = u.showcaserefused
WHERE e.tempvar24h > 1 OR e.humvar24h > 3
GROUP BY e.showcaseref
ORDER BY e.showcaseref;
```

Consistency with ISH. This case validates Obs. 1 at the trajectory level. Both agents use comparable total budget (31 vs 34 calls), but PASCAL front-loads 2 KB calls that resolve the threshold *before* any SQL is generated, while ReACT disperses 15 submits to discover the same threshold empirically. The result is identical (both achieve full reward), but the acquisition path is qualitatively different: PASCAL’s KB-first strategy produces a $4.8\times$ shorter path to P1 (6 turns vs 29). This is the per-task instantiation of the aggregate finding in Table 2: “how the budget is spent matters more than how much is spent.”

The Phase-2 segment demonstrates Δ_{ph} : the follow-up references “those showcases” which requires the Phase-1 result set. PASCAL preserves the Phase-1 SQL as a subquery in the Phase-2

WHERE clause, closing Δ_{ph} mechanically via the state-carry mechanism (Algorithm 3).

F.7 Cross-case summary

Recurring baseline failure modes. Across the 112-task pool we observe four patterns in baseline submitted SQL: (i) missing component KB (hallucinated thresholds or wrong aggregation, common in Δ_{kb} cases); (ii) reasonable-but-wrong interpretation (multiple baselines disagree on Δ_{usr} tasks); (iii) empty submissions (DAIL-SQL zero-shot on hard tasks); (iv) syntactic or alias errors when schema context is incomplete. PASCAL addresses (i) via typed KB acquisition (component lookup + ask_user fallback when hidden), (ii) via ask_user before commit, and (iv) via explore_tables + get_column_meaning. (iii) is a generation-baseline brittleness that no acquisition protocol can rescue. These four patterns map directly to the gap taxonomy (Section 2.2): (i) is Δ_{kb} , (ii) is Δ_{usr} , and (iv) is Δ_{sch} ; only (iii) lies outside the acquisition protocol’s scope.

G Experimental Setting Details

G.1 Benchmarks and splits

We evaluate on two benchmarks.

BIRD-Interact. The interactive text-to-SQL benchmark of Huo et al. (2026) with two difficulty splits: lite_300 (18 databases, 300 tasks) and full_600 (22 databases, 600 tasks). Each task consists of a Phase-1 query and a Phase-2 follow-up; success is measured by execution-match at each phase. All reported numbers use the NVD (No Value-Diff) audit setting, where cell-value comparison in submit_sql feedback is disabled (see Section G.2).

PRACTIQ. The conversational text-to-SQL benchmark of Dong et al. (2025), built on Spider databases with eight question categories: four *ambiguous* (Ambiguous SELECT Column, Ambiguous WHERE Column, Ambiguous Values Within Column, Ambiguous Filter Criteria) and four *unanswerable* (Nonexistent SELECT Column, Nonexistent WHERE Column, Nonexistent Filter Value, Unsupported Join). We evaluate on the medium_n20 split (1,069 tasks; 20 questions per category per database). Details in Appendix I.

G.2 Models and infrastructure

SQL backbone. The default backbone is Qwen3.6-35B-A3B-FP8⁴ (MoE, 3B active parameters per token, FP8 weights). The ablation backbone is Qwen3.6-27B-FP8⁵ (dense, FP8 weights). Both are served via vLLM 0.19.1 with FlashInfer attention, tensor-parallel TP=2 across $2 \times$ NVIDIA RTX 4090 (24 GB each), 80K context window, max-num-seqs=16. Multi-token prediction (MTP) is disabled for the 35B anchor due to a memory constraint in the 2-GPU coexistence configuration; this removes speculative-decoding speedup but does not affect task outcomes.

User simulator. Gemini-2.5-Flash-Lite⁶ via the Google AISTudio API. This is the same alias the BIRD-Interact leaderboard uses for grading; our setup is intentionally identical for leaderboard comparability.

Database backends. BIRD-Interact: a single PostgreSQL 14 instance hosting all 22 full_600 and 18 lite_300 database templates; per-task working databases are cloned at task initialisation. PRACTIQ: Spider SQLite databases with schema modifications applied at initialisation.

Concurrency. The orchestrator runs at $c \in \{24, 48\}$ concurrent tasks with a 100-turn ceiling per task.

G.3 BIRD-COIN budget and tool costs

Each task receives an initial budget $B = 30$ in BIRD-COIN:

$$B = 6 + 2m_{amb} + 2p, \quad (5)$$

where m_{amb} is the per-task count of annotated ambiguities ($|\text{critical_ambiguity}| + |\text{knowledge_ambiguity}|$) and $p = 3$ is the patience parameter (fixed across all experiments). PRACTIQ tasks use similar formula: $B = 12$ for the 506 unanswerable tasks ($m_{amb} = 0$; agent expected to abstain) and $B = 14$ for the 563 ambiguous tasks ($m_{amb} = 1$; one ask_user clarification in scope).

Per-tool costs follow the BIRD-Interact specification (Huo et al., 2026) (Table 38).

⁴<https://huggingface.co/Qwen/Qwen3.6-35B-A3B-FP8>

⁵<https://huggingface.co/Qwen/Qwen3.6-27B-FP8>

⁶Google GA alias gemini-2.5-flash-lite; we did not pin a snapshot.

#	Task	Dominant gap class	P1 turn	KB found/hid.	Usr calls	Fail sub	Obs. supported	Primary mechanism
1	mental_2	$\Delta_{kb} + \Delta_{exe}$	51	6/1	0	3	2+3	Typed KB + emp. fallback
2	cross_db_4	Δ_{usr} dominant	86	0/0	2	31	3	Informed clarification
3	museum_3	$\Delta_{kb} + \Delta_{sch}$	16	3/1	0	0	1+2	Namespace + schema
4	archeology_1	$\Delta_{kb} + \Delta_{usr}$	80	6/1	2	2	2+3	KB retr. + ask_user
5	cybermarket_1	$\Delta_{exe}/\Delta_{sch}$	10	0/1	0	1	1	Schema exploration

Table 37: Per-case pre-P1 acquisition profile. Cases drawn from the 112-task intersection {DIN-SQL, MAC-SQL, CHESS, DAIL-SQL} P1-failed $\cap$ PASCAL 35B P1-passed on lite_300. “Obs. supported” indicates which Section 3 observations are illustrated by the case.

Tool	Cost	Tool	Cost
explore_tables	0.0	execute_sql	1.0
list_fkeys	0.0	get_schema	1.0
sample_values	0.5	get_all_col_mean.	1.0
check_val_exists	0.25	get_all_kb_defs	1.0
get_col_meaning	0.5	ask_user	2.0
list_kb_names	0.5	abstain	2.0
get_kb_def	0.5	submit_sql	3.0

Table 38: Per-tool BIRD-COIN cost. Phase-0 schema exploration is free; semantic acquisition (column meaning, KB lookups, value sampling) is half-price; SQL execution, user clarification, and submission carry the full cost.

G.4 Retry breaker thresholds

The retry breaker (Section 4.3) monitors the submission history and fires on three conditions. The design defaults are $N_v=N_e=3$ and $N_{cap}=10$ (the values reported in Section 4.3). After empirical testing on long-horizon tasks (50+ turns), the production configuration loosens two thresholds to reduce false positives on hard tasks that legitimately require many submissions, as listed in Table 39.

G.5 Metric definitions

Per-class sufficiency $S_c(q)$. Defined in Eq. 3 (Appendix D). Tables and KB-IDs match by exact equality; columns by table.column or bare column name; KB-terms and intent by token overlap $\geq 50\%$.

Mean sufficiency $\bar{S}(q)$. The item-weighted aggregate $\bar{S}(q) = \frac{1}{5} \sum_c S_c(q)$ followed by a uniform mean across tasks.

InfoFail and GenFail attribution. For a failed task ($P_1 = 0$): *InfoFail* iff either the required KB-ID set is not contained in the acquired KB-ID set, or any required intent term is not covered by ask_user tokens; *GenFail* iff the sufficiency condition holds but P1 still fails (a semantic SQL error in the presence of all required information).

1st-KB. The turn number at which the first get_kb_definition call occurs, normalised by total

trajectory length. Lower values indicate earlier KB engagement in the priority schedule.

Rpt-SQL. The maximum count of any identical submitted SQL (case-folded, whitespace-normalised), divided by total submit_sql calls. Values above 1.0 indicate blind re-submission without modification.

Overthink. $OT(q) = (n_{turns} - t^*)/n_{turns}$, where t^* is the last turn acquiring a new gold-required information unit. High values indicate trajectory continuation after information saturation.

Tool entropy H . Shannon entropy of the per-task tool-call distribution: $H(q) = -\sum_a p(a | q) \log_2 p(a | q)$.

Normalised Reward. The BIRD-Interact leaderboard’s primary metric: a P1/P2 weighted aggregate rescaled to $[0, 100]$, identical to Table 4 in the main text.

H Additional Experiments

The schema-input control and failure-case analysis use all 300 lite_300 tasks. The remaining experiments use the same difficulty-stratified 100-task subset as Section 3.1, unless stated otherwise.

H.1 Full lite_300 Analysis

Schema-input control.

To isolate the effect of the initial schema input, we give the official ReAct scaffold the same initial schema as PASCAL while keeping its interaction policy unchanged. P1 changes from 22.7% to 23.0%, compared with 51.0% for PASCAL.

Configuration	P1 (%)
Official ReAct scaffold	22.7
+ same initial schema as PASCAL	23.0
PASCAL	51.0

Table 40: Schema-input control on all 300 lite_300 tasks using Qwen3.6-35B.

Failure case analysis.

Symbol	Condition	Design	Prod.	Loosening rationale
N_{cap} (STOP)	Total submissions $ H_t $	10	30	Hard tasks (50+ turns) legitimately submit 10+ times before passing
N_v (STUCK)	Consecutive identical SQL hashes	3	3	0% false-positive on passing traces; kept unchanged
N_e (SWITCH)	Consecutive identical error categories	3	5	Threshold of 3 caused $\sim 60\%$ false-positive on hard tasks

Table 39: Retry breaker thresholds (Section 4.3). Design defaults are the originally tested coarse values; production defaults are loosened after empirical validation on long-horizon tasks.

Across all 300 tasks, PASCAL passes 86 tasks that the official ReAct scaffold fails, while seven tasks show the reverse outcome. Among these seven regressions, the model either repeatedly re-submits SQL while a semantic detail remains unresolved, attributes a self-induced SQL error to the environment, or requests user clarification only after repeated failed submissions. On easy Management tasks, PASCAL uses 3.45 acquisition actions before its first SQL, compared with 5.10 for the official ReAct scaffold.

H.2 Robustness on the 100-task Subset

User simulator and incomplete replies.

We test the user channel with two controls. First, we replace the default Gemini-2.5-Flash-Lite user simulator with the stronger GLM-5.2 while keeping the agent configuration unchanged. Second, we keep Gemini-2.5-Flash-Lite and make its replies incomplete: with probabilities 0.3 and 0.5, the simulator withholds exact formulas and thresholds and returns only qualitative information, while grading remains unchanged.

Simulator	Reply condition	PASCAL P1	Official ReAct P1
Gemini-2.5-Flash-Lite	Original	54.0	41.0
GLM-5.2	Original	53.0	44.0
Gemini-2.5-Flash-Lite	Vague ($p=0.3$)	57.0	31.0
Gemini-2.5-Flash-Lite	Vague ($p=0.5$)	51.0	37.0

Table 41: P1 (%) under user-simulator and incomplete-reply controls on the 100-task subset. In the vague conditions, exact formulas and thresholds are withheld with probability p .

The timing of user clarification differs between the two methods: 98.3% of PASCAL’s ask_user calls occur after database probing begins, whereas 90.1% of the official ReAct scaffold’s clarification calls occur before probing.

Unavailable and misleading clarification.

Within the same 100-task control set, we further evaluate clarification-dependent tasks using Qwen3.6-Plus. In the *does-not-know* condition, the user cannot provide the requested definition; in the *misleading* condition, the user provides a plausible but incorrect formula or threshold. P1

is 65.0% with the original replies, 57.0% with does-not-know replies, and 59.0% with misleading replies.

Backbone robustness.

To test robustness to backbone capability, we replace the default Qwen3.6-35B with Qwen3.6-Plus and rerun the same 100-task subset under the same PASCAL protocol. P1 increases from 54.0% to 65.0%, with additional successes concentrated on Query tasks that require evidence from multiple sources.

PASCAL backbone	P1 (%)
Qwen3.6-35B	54.0
Qwen3.6-Plus	65.0

Table 42: Backbone comparison under the same PASCAL protocol on the 100-task subset.

Repeated runs.

To assess run-to-run variation, we repeat the Qwen3.6-35B/Gemini-2.5-Flash-Lite configuration three times on the same 100-task subset, obtaining 54.0%, 54.0%, and 53.0% P1. The corresponding 95% Student- t confidence interval is [52.2%, 55.1%].

H.3 Interaction-control Sensitivity

Tool-cost profile.

To test whether source allocation responds to relative tool costs, we use the inverted cost profile, in which user clarification becomes cheaper and SQL execution becomes more expensive than in the original setting. P1 changes from 54.0% to 57.0%, accompanied by more ask_user calls and fewer SQL executions.

Interaction budget.

To test sensitivity to the total interaction budget, we rerun the 100-task subset with $B \in \{15, 21, 30, 45\}$ while keeping the remaining configuration fixed.

Total budget B	15	21	30	45
P1 (%)	47.0	51.0	54.0	55.0

Table 43: Budget sensitivity on the 100-task subset.

Across these four budgets, nearly 70% of tasks retain the same pass/fail outcome, while fewer than 10% consistently improve as the budget increases.

Retry thresholds.

We vary the two retry-breaker thresholds that differ from their design values in the main configuration: N_{cap} , the cumulative-failure threshold, and N_e , the consecutive-failure threshold. These thresholds determine when repeated SQL failures stop further retries and return the agent to information acquisition. The main experiments use $(N_{cap}, N_e) = (30, 5)$.

(N_{cap}, N_e)	(10,3)	(20,5)	(30,5)	(30,7)	(10,7)
P1 (%)	49.0	51.0	54.0	47.0	49.0

Table 44: Retry-threshold sensitivity on the 100-task subset.

I Cross-Benchmark Analysis on PRACTIQ

This appendix complements the brief PRACTIQ reference in Section 5.4 with a per-category breakdown, a quantitative comparison against the PRACTIQ paper’s baselines.

I.1 Benchmark and budget

PRACTIQ (Dong et al., 2025) is a conversational text-to-SQL benchmark over Spider databases with eight question categories: four *ambiguous* and four *unanswerable* (Table 45). We evaluate on the medium_n20 split (1,069 tasks).

Budget. PRACTIQ tasks use the standard formula (Eq. 5 in Appendix G.3) with no override. The 506 unanswerable tasks have $m_{amb} = 0$ ($B = 12$); the 563 ambiguous tasks have $m_{amb} = 1$ ($B = 14$).

I.2 Per-category results

PASCAL with Qwen3.6-35B-A3B-FP8 reaches P1 = **86.5%** (925 / 1069) on medium_n20. Table 45 reports per-category P1 accuracy.

I.3 Comparison with PRACTIQ paper baselines

The PRACTIQ paper benchmarks five LLMs within the DIN-SQL framework: Claude 3.5 Sonnet, Claude 3 Sonnet (both Anthropic), Mixtral-large-v2 (Mistral AI), Llama-3.1 70B, and Llama-3.1 8B (both Meta). The two best-performing baselines (Claude 3.5 Sonnet at 71.27% and Mixtral-large-v2 at 71.95%) both receive full oracle clarifi-

cation context as static input. PASCAL’s +14.6 pp advantage over the best baseline comes from the protocol (multi-turn iteration, typed KB access, structured abstain path) rather than a stronger backbone or user simulator. Both protocols access PRACTIQ-curated clarification content (see oracle-disclosure below).

The largest gains are on unanswerable categories (+21 to +34 pp): PASCAL’s schema exploration pipeline correctly identifies missing columns, values, or join paths and routes to abstain in 82% of cases (65 / 79 unanswerable tasks passing via abstain). The smallest gains are on the ambiguous SELECT / WHERE categories (+4 pp), where the disambiguation requires interpreting subtle column-name differences that the protocol alone cannot fully resolve.

Channel attribution. The 1,069 tasks decompose into 563 ambiguous (52.7%, Δ_{usr} channel), 506 unanswerable split between schema-missing (39.4%, Δ_{sch} channel) and KB-missing (8.0%, Δ_{kb} channel). Each acquisition channel addresses a distinct corpus slice; no single channel suffices.

I.4 Efficiency

Average per task on PRACTIQ medium_n20: 155 s wall-time, 40.8 tool calls, 5.7 submit_sql calls, 0.8 ask_user calls, and 12.6 of 13.1 BIRD-COIN consumed (96.8% utilisation). The low per-task cost reflects the shorter SQL surface area (Spider-scale schemas) and the deterministic abstention path for the four unanswerable categories.

<i>PRACTIQ baselines (DIN-SQL, Table 5 of Dong et al. 2025)</i>								
Category	<i>n</i>	Claude 3.5 Sonnet	Claude 3 Sonnet	Mixtral- large-v2	Llama-3.1 70B	Llama-3.1 8B	PASCAL 35B	Δ
<i>Ambiguous (resolve via ask_user, Δ_{usr})</i>								
Ambig. Filter Criteria	140	77.23	61.72	75.91	68.65	48.84	86.4	+9.2
Ambig. SELECT Column	273	67.25	58.48	74.27	71.35	55.56	78.4	+4.1
Ambig. Values Within Col.	31	68.03	53.28	69.67	63.11	45.90	83.9	+14.2
Ambig. WHERE Column	119	77.14	59.05	75.24	71.43	59.05	79.0	+1.9
<i>Unanswerable (resolve via schema exploration $\Delta_{\text{sch}} / KB \Delta_{\text{kb}} + \text{abstain}$)</i>								
Nonexist. Filter Value	85	74.12	64.71	71.76	65.88	54.71	92.9	+18.8
Nonexist. SELECT Column	80	64.73	55.19	66.18	67.01	48.76	100.0	+33.0
Nonexist. WHERE Column	110	65.11	51.06	65.53	69.36	46.81	95.5	+26.1
Unsupported Join	231	76.53	77.46	77.00	63.85	56.34	89.2	+11.7
Avg (excl. answerable)	1069	71.27	60.12	71.95	67.58	52.00	86.5	+14.6

Table 45: Per-category execution accuracy (%) on PRACTIQ medium_n20. All PRACTIQ baselines are under official setting (Pourreza and Rafiei, 2023; Dong et al., 2025) with oracle clarification context; PASCAL uses Qwen3.6-35B under the same oracle-clarification convention. Δ is computed against the best PRACTIQ baseline per category. The largest gains are on unanswerable categories (+18 to +33 pp), where PASCAL’s schema exploration pipeline correctly routes to abstain.