Evaluating LLMs on Conversational Text-to-SQL under Chain Ambiguity and Intent Drift

Yujia Liu[1*], Jiayan Lin[1*], Zijin Hong[2*†], Zheng Yuan[2], Shengyuan Chen[2], Hao Chen[3] Qinggang Zhang[4], Xiao Huang[2], Feiran Huang[5†]
[1]Jinan University [2]The Hong Kong Polytechnic University
[3]City University of Macau [4]Jilin University [5]Beihang University
yujialiu@stu2023.jnu.edu.cn; ljiayan@stu2023.jnu.edu.cn
zijin.hong@connect.polyu.hk; xiao.huang@polyu.edu.hk; huangfr@buaa.edu.cn

Abstract

Recent advances in large language models (LLMs) have established conversational text-to-SQL as a practical interface between users and databases, often involving multiple turns of clarification and revision. However, existing benchmarks primarily evaluate execution accuracy, leaving the unfolding and shifting of user intent across turns largely uncovered. To address this, we introduce **TIDE-BENCH**, a benchmark for conversa**TI**onal text-to-SQL under chain ambiguity and intent **D**rift **E**valuation, targeting two recurring patterns: **chain ambiguity**, where an underspecified question triggers layered clarification with conditional dependencies, and **intent drift**, where the user retracts and replaces a previously committed request element. Built on 514 anchor SQLs from BIRD, TIDE-BENCH comprises 1,542 samples and introduces dedicated metrics for chain identification and drift recognition-resolution beyond execution accuracy. Evaluating 12 advanced LLMs reveals a persistent chain identification bottleneck unaffected by clarification frequency, a wide drift recognition-resolution gap, and overlap between failure modes when jointly activated. The corresponding code of TIDE-BENCH is released for further research[1].

1 Introduction

The advent of large language models (LLMs) has established text-to-SQL (Hong et al., 2025; Liu et al., 2025a) as a practical interface between users and structured databases. In real deployments, this interaction is rarely single-turn: users and agents converse over multiple rounds, refining requests and seeking clarification before SQL is produced (Yu et al., 2019a; Huo et al., 2026). Even agents that perform well in single-turn evaluation can fail once conversation becomes part of the task (Yu et al., 2018; Li et al., 2023; Lei et al., 2025).

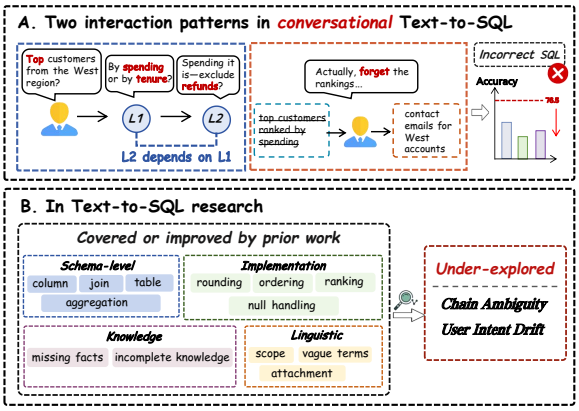

Figure 1: (A) Two conversational patterns that challenge text-to-SQL. (B) Neither is covered by existing text-to-SQL benchmarks or standard evaluation protocols.

However, **how does user intent actually unfold and shift across turns, and are current agents equipped to handle it?** Figure 1 illustrates two recurring interaction patterns in multi-turn dialogue. First, an underspecified question may trigger **chain ambiguity**: a layered clarification process in which a later clarification is conditionally dependent on the resolution of an earlier one, going beyond turn-local ambiguity studied in prior work (Bhaskar et al., 2023; Saparina and Lapata, 2024; Dong et al., 2025). Second, a user may retract and replace a committed request element mid-conversation, a non-monotonic shift we term **user intent drift**. These two interaction patterns raise two critical concerns: **1) Chain dependency is not captured by turn-local ambiguity**, as resolving each clarification in isolation says nothing about whether the agent recognizes the dependency between layers; **2) Commitment dynamics are not captured by end-to-end accuracy**, as a single execution match cannot tell whether the agent registered the retraction or merely happened to land on the right query.

Existing conversational text-to-SQL benchmarks (Yu et al., 2019b,a; Sun et al., 2025; Huo et al., 2026) measure multi-turn SQL accuracy, while a related line of work (Bhaskar et al., 2023;

*All authors contributed equally to this research.
†Corresponding author.
[1]https://github.com/Rcrossmeister/TIDE-Bench

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 55720–55739
October 24-29, 2026 ©2026 Association for Computational Linguistics

Saparina and Lapata, 2024; Dong et al., 2025; Ding et al., 2026) also defines turn-local ambiguity taxonomies for individual utterances. However, these efforts either evaluate dialogue execution as a single end-to-end number or restrict ambiguity to one turn at a time, leaving open how agents handle cross-turn dependency and retraction. In this context, **effectively evaluating how text-to-SQL agents handle these dialogue-level interaction patterns remains a significant challenge**. As a solution, this paper introduces **TIDE-BENCH**, a benchmark for conversaTIonal text-to-SQL under chain ambiguity and intent Drift Evaluation, which operationalizes chain ambiguity through a counterfactual test for conditional dependency and intent drift through a non-monotonicity test over commitment sets derived from SQL abstract syntax trees.

Specifically, TIDE-BENCH is built around a paired factorial design over 514 anchor SQLs from BIRD (Li et al., 2023): each anchor spawns three samples activating chain ambiguity alone and intent drift alone and together, so that performance differences are attributable to the activated interaction patterns rather than others. The resulting 1,542 samples are produced through a four-stage pipeline with automated validation and human verification. Beyond execution accuracy, TIDE-BENCH introduces dedicated metrics for chain-layer identification and drift recognition-resolution. Overall, our main contributions are summarized as follows:

- We identify and formalize **chain ambiguity** and **user intent drift** as two underexamined dialogue-level interaction patterns in conversational text-to-SQL, and operationalize each through a counterfactual test for conditional dependency and a non-monotonicity test over commitment sets for systematic evaluation.
- We introduce **TIDE-BENCH**, a benchmark of 514 anchor SQLs and 1,542 paired samples constructed through a four-stage pipeline with automated validation and human verification, whose design attributes performance differences to the activated interaction patterns.
- We evaluate 12 advanced LLMs on TIDE-BENCH and also diagnose three key distinct failure modes: a persistent chain identification bottleneck unaffected by ask behavior, a drift recognition-resolution gap concentrated on retraction-target inference, and a joint penalty where the two failures occur on overlapping rather than independent samples.

2 Problem Formalization

Real-world text-to-SQL systems often operate in multi-turn interactions, where a user's intent may be underspecified at first and clarified over time. During this process, the intended SQL query may evolve as the user resolves ambiguity or revises a prior choice. We formalize two key interaction patterns: **chain ambiguity** and **user intent drift**.

2.1 Chain Ambiguity

Definition. A sample exhibits *chain ambiguity* when an underspecified user query induces two layers of clarification: **L1 (layer one)** and **L2 (layer two)**, such that the existence, scope, or framing of L2 is conditionally dependent on the resolution of L1. Each layer offers two semantically plausible interpretations of the query. This conditional dependency distinguishes *chain ambiguity* from two stacked independent ambiguities, where simply resolving L1 leaves L2's semantic content unaffected.

Counterfactual Test. To distinguish chain ambiguity from independent stacking, we use the following *counterfactual test* (Pearl, 2009): if L1 were resolved with its alternative choice, would L2 still be posed as the same question? The test yields one of three verdicts: (i) **Eliminated**, where L2's premise vanishes under the L1 alternative and the question becomes vacuous; (ii) **Transformed**, where L2 must be reframed through changes to its referent, vocabulary, or scope; and (iii) **Invariant**, where L2 can be asked verbatim, indicating semantic independence. A sample qualifies as chain-ambiguous only when the verdict is *eliminated* or *transformed*; samples judged *invariant* are rejected at this stage.

Worked Example. For the dialogue in Figure 2, the L1 alternative *tenure* measures duration rather than a monetary quantity, so L2's question about excluding refunds no longer has a valid referent. The verdict for this example is therefore *eliminated*.

2.2 User Intent Drift

Definition. *User intent drift* is a non-monotonic revision in which at least one previously established commitment is withdrawn or replaced. Unlike cooperative continuation, which retains all prior commitments, drift requires revising the dialogue state and discarding withdrawn requirements from the resulting query but may also introduce new commitments. To formalize this definition, the user's committed intent at each dialogue turn is represented

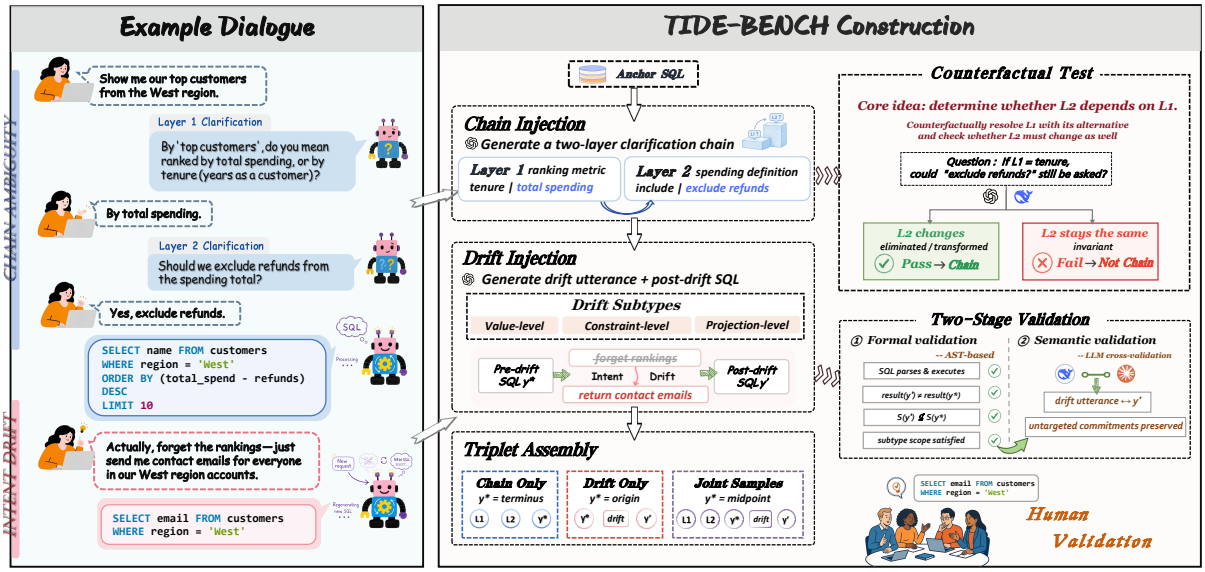

Figure 2: **Left:** motivating dialogue illustrating chain ambiguity (L1/L2 clarification with conditional dependency) and user intent drift (non-monotonic retraction of a prior commitment). **Right:** overview of the TIDE-BENCH four-stage construction pipeline (anchor selection, chain injection, drift injection, and triplet assembly; see §3).

by a SQL query y encoding the intent accumulated up to that turn and a corresponding *commitment set* $S(y)$ directly extracted from its full abstract syntax tree. The set covers filtering predicates, projections, grouping and ordering keys, aggregation filters, and row-limit and offset specifications (Appendix A). Let $S_{\text{pre}} = S(y_{\text{pre}})$ and $S_{\text{post}} = S(y_{\text{post}})$.

Non-Monotonicity Test. Drift occurs iff $S_{\text{pre}} \nsubseteq S_{\text{post}}$. The test itself places no restriction on added commitments, $S_{\text{post}} \setminus S_{\text{pre}}$, and allows removal and replacement while excluding no-change updates and pure additions. Each dialogue contains one controlled drift event, which may affect multiple commitments. This design keeps the pre- and post-drift targets identifiable under the paired design.

Drift Subtypes. The benchmark distinguishes three subtypes: (i) **Value-level**, which replaces only one predicate literal while preserving predicate structure; (ii) **Constraint-level**, which removes one or more filtering predicates and makes no other uery changes; and (iii) **Projection-level**, which changes only output-side commitments while preserving all filtering predicates. Algorithm 1 operationalizes this decision process, first applying the non-monotonicity test and then assigning each accepted drift update to one of the three subtypes.

3 TIDE-BENCH Construction

We describe the construction of TIDE-BENCH. The benchmark is built around a paired design that isolates each interaction pattern from the anchor.

3.1 Paired Factorial Design

TIDE-BENCH uses a **paired factorial design**: each BIRD anchor SQL y^* that passes construction filters spawns three samples sharing the anchor but activating different interaction patterns: $\mathcal{C}_{y^*}$ (Chain-only), $\mathcal{D}_{y^*}$ (Drift-only), and $\mathcal{J}_{y^*}$ (Joint). Pairing by anchor controls for schema complexity, domain difficulty, and surface ambiguity, attributing performance differences to the activated patterns rather than any uncontrolled anchor-specific confounds.

3.2 Construction Pipeline

Building on this design, our construction pipeline proceeds in four stages: anchor selection, chain injection, drift injection, and triplet assembly (Figure 2, right). We summarize each stage below; full prompts, judge configurations, and associated validation checks are instead deferred to Appendix B.

Anchor selection. Candidate anchors are drawn from BIRD (Li et al., 2023), retaining only SQLs that (i) execute on their SQLite database, (ii) carry enough structural complexity for two-layer chain injection, and (iii) admit at least one drift subtype from § 2.2. Stratified sampling balances complexity tiers and source splits, totaling **830 anchors**.

Chain injection. For each anchor, an advanced LLM (GPT-5.1) generates the two-layer clarification structure $(L1, L2)$, an underspecified initial question, chosen responses, and unchosen alternatives, conditioned on the anchor SQL as the canonical $L1 + L2$ target. Two independent judges (GPT-

Algorithm 1: User Intent Drift Operationalization

Require: Queries y_{pre} and y_{post}
Ensure: Drift status or subtype
Notation:
$\mathcal{W}(y)$: filtering commitments in y
$\mathcal{O}(y) = S(y) \setminus \mathcal{W}(y)$: output-side commitments in y
$\text{LitReplace}(\mathcal{W}_{\text{pre}}, \mathcal{W}_{\text{post}})$: one predicate literal changes; its structure and all other filters remain fixed

```
% Extract commitments
S_pre ← S(y_pre); S_post ← S(y_post)
Set W_t ← W(y_t) and O_t ← O(y_t) for t ∈ {pre, post}
% Test non-monotonicity
if S_pre ⊆ S_post then
    return NOT DRIFT
end if
% Assign drift subtype
if O_pre = O_post and LitReplace(W_pre, W_post) then
    return VALUE-LEVEL
end if
if O_pre = O_post and W_post ⊊ W_pre then
    return CONSTRAINT-LEVEL
end if
if W_pre = W_post and O_pre ⊈ O_post then
    return PROJECTION-LEVEL
end if
return REJECT
```

5.3-Codex, DeepSeek-V4-Pro) apply the counterfactual test (§ 2.1) and discard *invariant* candidates. This stage retains 695/830 candidates (83.7%), with 94.7% overall inter-judge agreement. A final LLM rewrite of BIRD's evidence field removes clauses pre-resolving L1 or L2 while faithfully preserving schema-essential mappings (Appendix B).

Drift injection. For each chain-validated anchor, the same generator produces a drift utterance and post-drift target y^*_{post} conditioned on a pre-assigned subtype. Each pair passes two stages. AST-level validation enforces parseability, executability, result distinctness, the non-monotonicity condition $S_{\text{pre}} \nsubseteq S_{\text{post}}$, and subtype-specific scope constraints. LLM cross-validation (Claude-Sonnet-4.6, DeepSeek-V4-Pro) assesses semantic validity beyond formal checks, namely faithfulness to the drift turn and preservation of non-targeted structural elements. The two stages are complementary: AST checks enforce the definition, while cross-validation guards against drifts pass checks but misalign with semantic intent. AST validation retains 554/695 (79.7%); cross-validation retains 514/554 (92.8%), with 96.6% overall inter-judge agreement.

Triplet assembly. Surviving chain and drift constructions are paired by anchor to form **514 triplets (1,542 samples)**, each consisting of matched Chain-only, Drift-only, and Joint instances built over the same anchor SQL and sharing the same schema. This pairing supports controlled performance comparisons across all the three interaction settings.

Split	Majority Acc.	Strict Acc.	Fleiss' κ
CHAIN-ONLY	93.2	84.8	0.82
DRIFT-ONLY	95.6	87.6	0.80
JOINT	88.4	76.8	0.77
Overall	**92.4**	**83.1**	**0.81**

Table 1: Human validation on 250 stratified triplets. Majority requires $\geq 3/5$ annotators per item; strict requires unanimity. Fleiss' κ uses binary accept/reject verdicts.

3.3 Human Validation

Beyond the automated filters above, we additionally verify the quality and construct validity of TIDE-BENCH through human annotation. We conduct a stratified audit over 250 triplets (750 samples), balanced across sample type, drift subtype, counterfactual verdict, and SQL complexity tier. Each sample is independently annotated by five annotators with NLP and production SQL backgrounds, blind to automated verdicts produced during construction. Rather than a holistic judgment, we use a per-item checklist that operationalizes the definitions in § 2; the full checklist and annotator details appear in Appendix C for reproducibility.

We report *majority acceptance* ($\geq$3/5 annotators pass every applicable item) and *strict acceptance* (unanimity). Overall majority and strict acceptance are 92.4% and 83.1% (Table 1; all Fleiss' $\kappa > 0.77$). Counterfactual dependency and non-monotonic retraction pass on 93.4% and 94.8% respectively, while joint transition coherence is slightly lower at 91.6%, as it requires both interaction patterns to be validated jointly in the dialogue.

3.4 Dataset Statistics

TIDE-BENCH comprises 514 triplets and 1,542 benchmark samples (514 each of Chain-only, Drift-only, Joint). Anchors are partitioned into basic, intermediate, and advanced tiers by a rule-based score combining anchor-SQL structure with chain- and drift-induced factors. Drift subtypes are evenly covered: value level 34.0%, constraint-level 31.3%, and projection-level 34.6%. Counterfactual verdicts are balanced between *eliminated* (49.4%) and *transformed* (50.6%), while the overall complexity is mostly intermediate (61.3%), with basic and advanced tiers at 26.5% and 12.3%, respectively.

Metric	Full Name	Definition	Set
DEA	Dialogue Execution Accuracy	$\frac{1}{\lvert\mathcal{S}\rvert}\sum_{\tau\in\mathcal{S}}\prod_{i=1}^{n_\tau}\mathrm{EX}(y_i, y_i^*)$	all
AR	Ask Rate	$\frac{1}{\lvert\mathcal{S}\rvert}\sum_{\tau\in\mathcal{S}}\mathbb{1}[\tau \text{ contains a clarification}]$	all
L1-IR	L1 Identification Rate	$\frac{1}{\lvert\mathcal{S}\rvert}\sum_{\tau\in\mathcal{S}}\mathbb{1}[\exists\, t : \rho_t = \mathrm{L1}]$	$\mathcal{C}, \mathcal{J}$
L2-IR^p	L2 Identification Rate	$\frac{1}{\lvert\mathcal{S}\rvert}\sum_{\tau\in\mathcal{S}}\mathbb{1}[\exists\, t : \rho_t = \mathrm{L2}]$	$\mathcal{C}, \mathcal{J}$
JIR	Joint Identification Rate	$\frac{1}{\lvert\mathcal{S}\rvert}\sum_{\tau\in\mathcal{S}}\mathbb{1}[\exists\, t_1, t_2 : \rho_{t_1} = \mathrm{L1} \wedge \rho_{t_2} = \mathrm{L2}]$	$\mathcal{C}, \mathcal{J}$
DRR	Drift Recognition Rate	$\frac{1}{\lvert\mathcal{S}\rvert}\sum_{\tau\in\mathcal{S}}\mathbb{1}[\mathrm{Rec}(\tau)]$	$\mathcal{D}, \mathcal{J}$
DRA	Drift Resolution Accuracy	$\frac{1}{\lvert\mathcal{S}\rvert}\sum_{\tau\in\mathcal{S}}\mathbb{1}\left[\mathrm{Rec}(\tau) \wedge \mathrm{EX}(y_{\text{post}}, y_{\text{post}}^*)\right]$	$\mathcal{D}, \mathcal{J}$

Table 2: TIDE-BENCH metrics spanning four behavioral facets: execution accuracy (DEA), clarification behavior (AR), chain-layer identification (L1-IR, L2-IR^p, JIR), and drift handling (DRR, DRA). $\mathcal{C}, \mathcal{D}, \mathcal{J}$ denote Chain-only, Drift-only, and Joint sample sets. L2-IR^p uses the p-protocol (§ 4.1); all other metrics use the f-protocol instead.

4 Evaluation Protocol

Building on the construction in § 3, this section specifies how each TIDE-BENCH sample is turned into an evaluation run for the experiments below.

4.1 Interaction Setup

Each TIDE-BENCH sample defines a dialogue task where an agent interacts with a user simulator (§ 4.2) over a database $\mathcal{D}$ with schema $\mathcal{S}$. The dialogue begins with the user's initial question u_1 and proceeds in alternating turns $(u_1, a_1, u_2, a_2, \ldots)$. Each agent turn a_i is either a clarification request or candidate SQL query against $\mathcal{D}$; the simulator returns u_{i+1}. For Drift-only and Joint samples, the harness injects the pre-annotated drift turn verbatim after the agent's first SQL query, ensuring deterministic drift content across evaluation runs.

***f*-protocol and *p*-protocol.** Each sample is evaluated under two protocols. The ***f*-protocol** (*full*) starts from u_1, requiring the agent to elicit and resolve both clarification layers for Chain-only and Joint samples. The ***p*-protocol** (L1-*primed*) pre-populates the history with a synthetic three-turn L1 resolution: the initial question, annotated L1 clarification, and canonical L1 answer. It marks the simulator's L1 state as resolved before the first live turn. The p-protocol isolates L2, testing whether agents miss it once L1 is no longer a confound. Full harness parameter settings appear in Appendix D.

4.2 User Simulator

The simulator follows the two-stage function-driven design of recent advance (Huo et al., 2026), with three minor adaptations for TIDE-BENCH's two-layer chain structure (Appendix E). The router decisions ρ_t used by the metrics below are produced at each user turn responding to an agent clarification. We validate the simulator on USERSIM-GUARD and on agent–simulator traces independently labeled by five SQL experts (Fleiss' $\kappa = 0.802$); full configuration provided in Appendix E.

4.3 Metrics

Seven metrics span four behavioral facets: execution accuracy, clarification behavior, chain-layer identification and drift handling. All are computed as proportions over a sample set $\mathcal{S}$ of a given type.

Notation. Let τ denote a dialogue trajectory containing $n_\tau \in \{1, 2\}$ SQL queries $y_1, \ldots, y_{n_\tau}$ paired with $y_1^*, \ldots, y_{n_\tau}^*$. Chain-only samples have $n_\tau = 1$, while Drift-only and Joint samples have $n_\tau = 2$, with $y_1 = y_{\text{pre}}$ and $y_2 = y_{\text{post}}$. Let $\mathrm{EX}(y, y^*) \in \{0, 1\}$ denote execution match, i.e., equivalent result sets modulo row order; $\rho_t \in \{\mathrm{L1}, \mathrm{L2}, \mathrm{Other}\}$ denote the router decision (§ 4.2) for the agent's clarification at turn t; and $S(y)$ denote the commitment set of y (§ 2.2). Define $\mathrm{Rec}(\tau) := \left[S(y_{\text{pre}}) \not\subseteq S(y_{\text{post}})\right]$ as the non-monotonicity condition used for drift recognition.

Metric definitions. The seven metrics are defined in Table 2. **DEA** requires every SQL query in a trajectory to pass execution match. **AR** measures whether the agent asks, regardless of layer, providing a baseline clarification rate. **L1-IR**, **L2-IR^p**, and **JIR** measure whether the agent surfaces clarification requests routed to the corresponding layer within a trajectory, with L2-IR^p evaluated under the p-protocol to isolate L2 from L1 confounds. **DRR** captures whether the agent's commitment sets exhibit the required non-monotonic change after the

#	Model	Chain-only	Drift-only				Joint			
			Value	Constraint	Projection	All	Value	Constraint	Projection	All
1	Claude-Sonnet-4.6	**35.9**	**47.7**	**44.2**	43.4	**45.1**	**31.2**	20.2	**31.4**	**27.8**
2	Qwen2.5-72B	32.4	44.3	41.1	44.0	43.2	30.1	21.5	28.6	26.8
3	o4-mini	32.9	42.6	41.1	40.0	41.2	**31.2**	21.5	27.4	26.8
4	GPT-4o	30.9	46.0	38.7	**44.6**	43.2	27.8	**22.1**	29.7	26.7
5	GPT-5.4	32.5	43.2	36.8	42.3	40.9	27.8	20.9	29.7	26.3
6	Llama-3.3-70B	32.7	40.3	39.3	38.9	39.5	29.5	**22.1**	26.9	26.3
7	o3-mini	32.6	40.9	37.4	38.9	39.1	29.5	18.4	28.6	25.7
8	Gemini-2.5-Flash	31.6	39.2	42.9	33.1	38.3	27.3	20.2	27.4	25.1
9	Grok-4.1-Fast	30.4	41.5	36.2	41.7	39.9	29.0	18.4	25.1	24.3
10	Qwen-Max	30.5	40.3	43.6	40.6	41.4	28.4	20.9	21.7	23.7
11	DeepSeek-V4-Flash	27.3	40.9	39.9	39.4	40.1	21.6	17.8	26.9	22.2
12	Grok-4-Fast	25.9	36.9	33.7	38.9	36.6	19.9	14.7	22.9	19.3

Table 3: DEA (%) under the paired factorial design. The best and second-best results per column are in bold and underlined, respectively. Chain-only reports strict execution accuracy; Drift-only and Joint report end-to-end accuracy broken down by drift subtype; All denotes aggregate performance, and higher metric values are better.

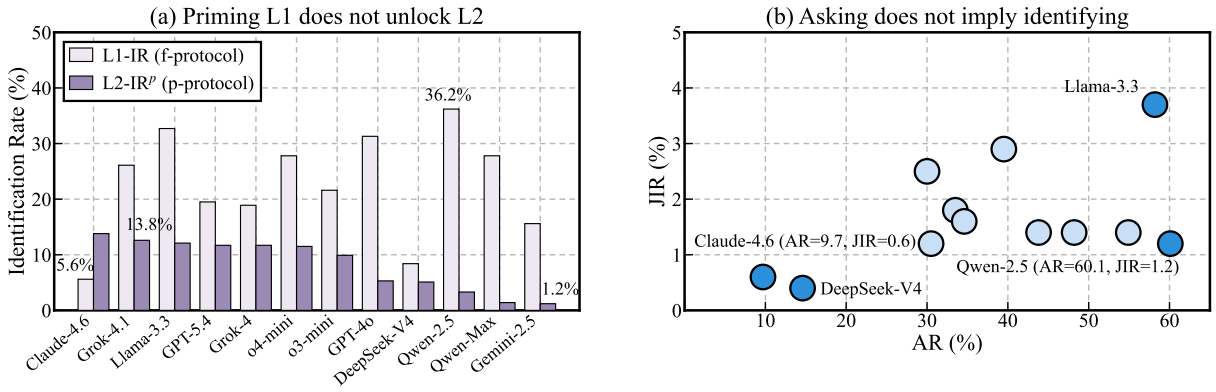

Figure 3: Chain-layer identification on Chain-only samples. (a) L1-IR (f-protocol) vs. L2-IR^p (p-protocol) per model: even with L1 pre-resolved, L2 identification stays low. (b) JIR against ask rate: identification remains low regardless of how much the agent asks. Model names are abbreviated; full names appear in Table 3 for reference.

drift utterance, and **DRA** additionally requires the resulting post-drift SQL to be executionally correct.

5 Experimental Results

We evaluate 12 advanced LLMs on TIDE-BENCH. We report model accuracy across the three sample types, diagnose where chain ambiguity and intent drift individually fail, and examine their interaction when both patterns co-occur in a single dialogue. This analysis distinguishes recognition from resolution and reveals how errors propagate across dialogue phases into their final SQL predictions.

5.1 Overall Performance

Table 3 reports DEA across the three sample types. Performance is low throughout: even the strongest evaluated model, Claude-Sonnet-4.6, reaches only 45.1% on Drift-only and drops to 27.8% on Joint, and no model exceeds 46% overall on any sample type. DEA is consistently highest on Drift-only, lowest on Joint, and intermediate on Chain-only.

This consistently low performance is not concentrated in any particular model class. Larger evaluated models outperform smaller by only 2–3 percentage points on average across three conditions, and the spread between top and bottom models stays within roughly 10 percentage points everywhere. The overall low performance therefore appears to come less from overall model scale than from how agents handle chain ambiguity and user intent drift. The following analyses separate these errors into chain identification, drift recognition, and drift resolution across models and subtypes, and examine how they compound in Joint samples.

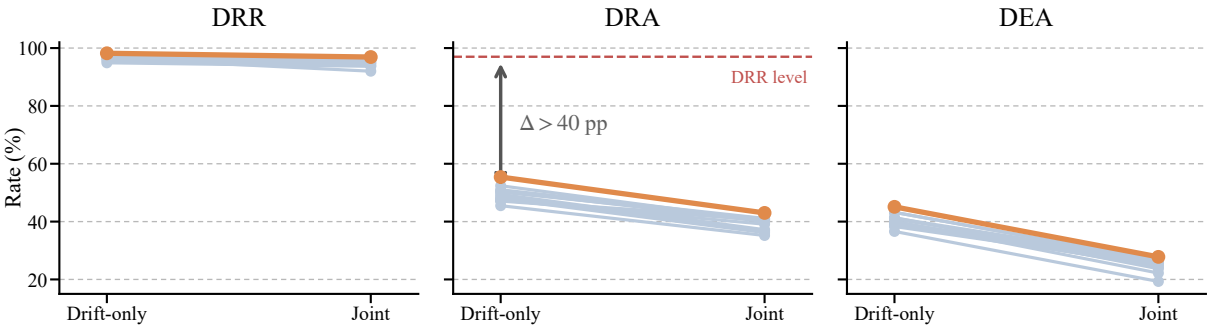

Figure 4: Drift handling from Drift-only to Joint across DRR, DRA, and DEA, with one line per model. DRR remains high and flat, while DRA and DEA start lower and decline further in Joint. The dashed DRA reference marks DRR, highlighting the recognition–resolution gap that persists for all models across both interaction settings. The orange line highlights Claude-Sonnet-4.6, which achieves the highest DEA in both Drift-only and Joint.

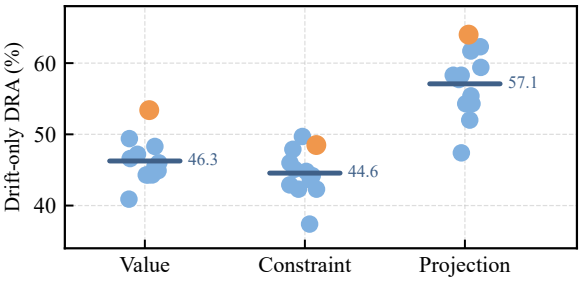

Figure 5: Drift-only DRA by drift subtype. Points denote models, and the bar marks the corresponding subtype mean. Orange points mark per-subtype maxima.

5.2 What Makes Chain Challenging

Layer-level identification. Resolving a chain-only sample requires recognizing dependent clarification layers from the initial question. We measure this directly through per-layer identification rates, with per-model details in Table 13 (Appendix F). JIR stays below 5% for every model and below 2% for the majority, ***showing that no evaluated model reliably surfaces both clarification layers***. L1-IR ranges from 5.6% (Claude-Sonnet-4.6) to 36.2% (Qwen2.5-72B), with most models clustered in the 20–30% range, despite L1 being directly available in the initial question. One plausible explanation for the even lower JIR is that L1 difficulty cascades downward: when L1 is missed, the agent may never reach L2 at all. The p-protocol (§ 4.1) is designed to remove exactly this confound, pre-populating L1 with its canonical resolution so that L2 is isolated as the sole downstream identification target. Figure 3(a) contrasts L1-IR under the f-protocol with L2-IR^p under the p-protocol for each model. Even with L1 externally resolved, L2-IR^p ranges only from 1.2% (Gemini-2.5-Flash) to 13.8% (Claude-Sonnet-4.6), with half the models still below 10%, ***showing that L2 identification fails on its own terms rather than as a side effect of L1 difficulty***.

Ask behavior. Could this low identification simply reflect how agents ask? An agent that rarely asks for clarification at all, or one that asks freely without targeting genuine ambiguity, would both depress identification rates. Figure 3(b) plots JIR against ask rate (AR) for all 12 models. Ask rates vary widely, ranging from below 10% to over 60%, yet JIR stays under 4% across the entire range. ***Models that ask more do not surface the chain more reliably***. Claude-Sonnet-4.6 and Qwen2.5-72B illustrate the two extremes: Claude asks on only 9.7% of Chain-only samples but Qwen2.5-72B asks on 60.1%, yet their JIR are at most 1.2% (0.6% and 1.2% respectively). To further isolate whether high asking is selective or indiscriminate, we measure the ask rate on Drift-only samples, where the initial user question carries no chain ambiguity, so any clarification reflects asking that is not ambiguity-driven. Claude's Drift-only ask rate (Table 13) drops to 0.4%, while Qwen2.5-72B's remains 20.0%, confirming opposite ask styles. ***Whether an agent asks selectively or indiscriminately, the chain remains unidentified, pointing to identification capacity itself as the bottleneck***.

5.3 Where Drift Handling Fails

Recognition–resolution gap. While chain failure is primarily an identification problem, drift exhibits a different failure pattern. We measure whether agents register the drift turn (DRR) and whether they recover an executionally correct post-drift SQL afterward (DRA). Figure 4 presents both metrics, with per-model values in Appendix F. DRR is at least 94.9% across models, while DRA remains roughly within the 45–55% band, ***leaving every evaluated model with a gap exceeding 40 percentage points between recognizing the drift and successfully resolving user's revised request***.

[Context]

Schema: `coaches(coachID, year, tmID, won, lost); teams(year, tmID, won, lost).`
Question: "Which coaches had disappointing seasons in 2000–2010? Show their coach ID, team, and year."

[Chain Ambiguity]

Gold path: $L1$: overall losing record → $L2$: team's full-season record.
Alternatives: $L1$: away-game record; $L2$: coach-stint record.

Gold SQL: `SELECT DISTINCT c.coachID, t.tmID, c.year FROM coaches c JOIN teams t ON c.tmID=t.tmID AND c.year=t.year WHERE c.year BETWEEN 2000 AND 2010 AND t.lost>t.won`

Claude-Sonnet-4.6: Clarifies that "disappointing" means a losing record, but uses `coaches.lost>coaches.won` without asking whose record should define the season.

$L1$ recognized $L2$ missed preEX=False ✗

Llama-3.3-70B: Also resolves the losing-record interpretation, but applies `coaches.lost > coaches.won` despite joining `teams`.

$L1$ recognized $L2$ missed preEX=False ✗

[User Intent Drift] (constraint-remove)

Drift utterance: "Do not limit it to 2000–2010; show those coaches for all years."

Gold post-drift SQL: `SELECT DISTINCT c.coachID, t.tmID, c.year FROM coaches c JOIN teams t ON c.tmID=t.tmID AND c.year=t.year WHERE t.lost>t.won`

Claude-Sonnet-4.6: Removes the year predicate but retains the coach-stint criterion.

DRR=True ✓ chain error inherited postEX=False ✗

Llama-3.3-70B: Makes the same drift update while retaining the same chain error.

DRR=True ✓ chain error inherited postEX=False ✗

Table 4: A TIDE-BENCH Joint example. Both models resolve $L1$ but miss the dependent record scope at $L2$; the resulting error persists after a correctly handled drift. SQL and model turns are abridged for space and clarity.

Resolution across subtypes. To localize this gap, we decompose Drift-only DRA by drift subtype. Figure 5 shows that resolution difficulty is not uniform across the three subtypes defined in § 2.2: a consistent ordering emerges across models where projection-level drift is the easiest to resolve for 11 of 12 models and constraint-level drift is the hardest for 10 of 12, indicating that the resolution gap is structured rather than evenly distributed. The spread is also substantial: within a single model, the subtype-level DRA can differ by as much as 19.4 percentage points (GPT-5.4: projection 61.7%, constraint 42.3%). This ordering runs against what SQL complexity alone would predict. Constraint-level drift yields the *simplest* post-drift SQL among the three subtypes, removing only WHERE predicates while leaving join, aggregation, and subquery structure unchanged (Appendix G). ***The subtype that yields the simplest SQL is the hardest to resolve, indicating that what makes constraint-level drift difficult is not the SQL itself but inferring which prior commitment is now being retracted***.

5.4 How Chain and Drift Interact

What happens when chain and drift co-occur? Comparing Drift-only to Joint isolates the cost of placing the same drift after a chain phase, with the post-drift ground-truth SQL held identical across the two conditions by construction (§ 3.1). Figure 4 traces the shift from Drift-only to Joint across metrics. ***The drop is asymmetric across all 12 models: DRR shifts by at most 5 percentage points, DRA falls by 6–12, and DEA by 13–18***. Recognition depends only on the drift turn and is largely unaffected by upstream events; resolution and execution depend on the agent state when the drift arrives. If chain and drift failed independently, Joint DEA would equal the product of Chain-only and Drift-only accuracies, approximately 13%. Across models, Joint DEA exceeds this prediction by 10–13 percentage points, ***indicating that the samples on which the two failures occur largely overlap***.

Across all three analyses, chain ambiguity, user intent drift, and their interaction emerge as three distinct dialogue-level failure modes that current models do not yet handle consistently and reliably.

5.5 Case Study

To make the interaction between chain ambiguity and user intent drift concrete, Table 4 traces a Joint anchor through its chain and drift phases. In the chain phase, the models must determine whether a *disappointing* season means an overall losing record or poor away-game performance, and then decide whether the record should reflect the team's full season or the coach's stint. Both models identify the first layer but fail to surface the dependent record-scope ambiguity, applying `coaches.won/lost` instead of the gold `teams.won/lost`. In the drift phase, the user removes the 2000–2010 restriction. Both models correctly delete the year predicate (DRR=True), yet retain the unresolved record-scope error, yielding preEX=False and postEX=False. Because this constraint-removal drift leaves the record definition unchanged, the post-drift failure can be cleanly traced to the chain phase. This example shows that resolving $L1$ and correctly applying the drift are not sufficient for end-to-end correctness: the unresolved $L2$ ambiguity invalidates both queries.

6 Related Work

Static benchmarks. Text-to-SQL progress has been driven by single-turn benchmarks. Spider (Yu et al., 2018) established cross-domain generalization; BIRD (Li et al., 2023) scaled evaluation to messy, knowledge-grounded databases across professional domains; and Spider 2.0 (Lei et al., 2025) pushes into enterprise workflows with multiple SQL dialects. These benchmarks advance query generation under the assumption of a well-formed question, leaving multi-turn interaction untested.

Conversational and multi-turn benchmarks. Related work brings dialogue into text-to-SQL evaluation. SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) introduce context-dependent question sequences and Wizard-of-Oz dialogues over Spider databases. More recent benchmarks raise interactive realism: BIRD-Interact (Huo et al., 2026) couples each database with a function-driven user simulator and a CRUD-spanning task suite, and DySQL-Bench (Sun et al., 2025) evaluates dynamic refinement under evolving user intent. These benchmarks mainly measure end-to-end multi-turn execution, but none specifically targets the *conditional dependency* between clarification layers or the *non-monotonic retraction* of committed elements as evaluation dimensions. TIDE-BENCH addresses these two phenomena directly, with a paired factorial design that attributes performance differences to chain ambiguity and user intent drift.

Ambiguity and clarification. Prior work treats ambiguity as central. AmbiQT (Bhaskar et al., 2023) characterizes lexical and structural ambiguity from overlapping schema names and join paths; AMBROSIA (Saparina and Lapata, 2024) grounds three linguistic ambiguity types in controlled databases; PRACTIQ (Dong et al., 2025) pairs ambiguous and unanswerable questions with single-turn clarifications; and AmbiSQL (Ding et al., 2026) couples a fine-grained taxonomy with interactive multiple-choice resolution. HiL-Bench (Trinh et al., 2026) studies help-seeking across progressively discovered but independent blockers. These ambiguity-focused benchmarks define rich turn-local taxonomies for individual user utterances. TIDE-BENCH is complementary: it studies how ambiguities depend on one another across turns (chain) and how committed intent can be retracted across turns (drift), two dialogue-level phenomena beyond these turn-local taxonomies.

7 Conclusion

We introduce TIDE-BENCH, a benchmark for evaluating how text-to-SQL agents handle two under-examined patterns in conversational interaction: chain ambiguity and user intent drift. Built around a paired factorial design over 514 anchor SQLs from BIRD, TIDE-BENCH contains 1,542 samples constructed through a four-stage automated pipeline with multi-stage validation and human verification. Our experiments on 12 LLMs reveal three failure modes. Chain ambiguity exposes an identification bottleneck unaffected by ask behavior, while user intent drift reveals a wide gap between recognition and resolution, and the two failures overlap when they co-occur. The paired design attributes these gaps to interaction structure rather than uncontrolled differences in schemas or query difficulty. These findings suggest that future work on conversational text-to-SQL should look beyond surface clarification behaviors toward modeling identification capacity and tracking dialogue-level commitments. This requires agents to surface dependent ambiguities and revise prior commitments without inheriting earlier errors. We hope TIDE-BENCH provides a diagnostic tool for both research and development in conversational text-to-SQL.

Limitations

This study has limitations. First, although BIRD offers cross-domain coverage, generalization to other benchmarks remains untested. Second, chain ambiguity is restricted to two clarification layers; extending the counterfactual test to deeper chains remains future work. Third, Joint samples follow only a chain-then-drift order, leaving drift-then-chain unexplored. Finally, our evaluation relies on an LLM-based user simulator. Although the simulator is validated on USERSIM-GUARD and on expert-labeled traces (Fleiss' $\kappa = 0.802$), it remains an imperfect proxy; human-in-the-loop evaluation would better reflect real user behavior across practical settings and more clearly reveal simulator-specific biases.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 62272200), and in part by the National Natural Science Foundation of China (No. 62502008), and in part by the Funding Scheme for Research and Innovation of FDCT (No. 0021/2025/ITP1), China and Macao S.A.R.

References

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

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ – a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology (TIST)*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Herbert H. Clark. 1996. *Using Language*. Cambridge University Press.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

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

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 *North American Chapter of the Association for Computational Linguistics (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. In *International Conference on Very Large Data Bases (VLDB)*.

Jonathan Ginzburg. 2012. *The Interactive Stance: Meaning for Conversation*. Oxford University Press.

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

Zijin Hong, Hao Wu, Su Dong, Junnan Dong, Yilin Xiao, Yujing Zhang, Zhu Wang, Feiran Huang, Linyi Li, Hongxia Yang, and Xiao Huang. 2026b. Benchmarking llms' mathematical reasoning with unseen random variables questions. In *Conference on Artificial Intelligence (AAAI)*.

Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. 2024. Knowledge-to-SQL: Enhancing SQL generation with data expert LLM. In *Findings of Association for Computational Linguistics (ACL)*.

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

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 *International Conference on Learning Representations (ICLR)*.

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

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The dawn of natural language to sql: Are we fully ready? In *International Conference on Very Large Data Bases (VLDB)*.

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

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025a. A survey of text-to-SQL in the era of LLMs: Where are we, and where are we

going? *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. 2025b. Nl2sql-bugs: A benchmark for detecting semantic errors in nl2sql translation. In *Conference on Knowledge Discovery and Data Mining (KDD)*.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ali Mohammadjafari, Anthony S Maida, and Raju Gottumukkala. 2024. From natural language to sql: Review of llm-based text-to-sql systems. *arXiv preprint arXiv:2410.01066*.

Judea Pearl. 2009. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

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

Yiwei Qin, Xuefeng Li, Haoyang Zou, Yixiu Liu, Shijie Xia, Zhen Huang, Yixin Ye, Weizhe Yuan, Hector Liu, Yuanzhi Li, and Pengfei Liu. 2024. O1 replication journey: A strategic progress report–part 1. *arXiv preprint arXiv:2410.18982*.

Ge Qu, Jinyang Li, Bowen Qin, Xiaolong Li, Nan Huo, Chenhao Ma, and Reynold Cheng. 2025. SHARE: An SLM-based hierarchical action CorREction assistant for text-to-SQL. In *Association for Computational Linguistics (ACL)*.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Conference on Artificial Intelligence (AAAI)*.

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

Linzhuang Sun, Tianyu Guo, Hao Liang, Yuying Li, Qifeng Cai, Jingxuan Wei, Bihui Yu, Wentao Zhang, and Bin Cui. 2025. Rethinking text-to-SQL: Dynamic multi-turn SQL interaction for real-world database exploration. *arXiv preprint arXiv:2510.26495*.

Qwen Team. 2024. Qwq: Reflect deeply on the boundaries of the unknown. *Hugging Face*.

Tu Trinh, Mohamed Elfeki, Guangze Luo, Kelvin Luu, Nathan Hunt, Ernesto Hernandez, Nandan Marwaha, Yannis Yiming He, Charles Wang, Fernando Carabedo, Alessa Castillo, and Bing Liu. 2026. HiL-Bench (Human-in-Loop Benchmark): Do agents know when to ask for help? *arXiv preprint arXiv:2604.09408*.

Pengfei Wang, Baolin Sun, Xuemei Dong, Yaxun Dai, Hongwei Yuan, Mengdie Chu, Yingqi Gao, Xiang Qi, Peng Zhang, and Ying Yan. 2025. Agentar-scale-sql: Advancing text-to-sql through orchestrated test-time scaling. *arXiv preprint arXiv:2509.24403*.

Cheng Xu, Shuhao Guan, Derek Greene, and M-Tahar Kechadi. 2024. Benchmark data contamination of large language models: A survey. *arXiv preprint arXiv:2406.04244*.

Hongyu Yao, Zijin Hong, Hao Chen, Zhiqing Li, Qijie Shen, Zuobin Ying, Qihua Feng, Huan Gong, and Feiran Huang. 2026. Macro graph of experts for billion-scale multi-task recommendation. In *Conference on Knowledge Discovery and Data Mining (KDD)*.

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 *Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

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 *Empirical Methods in Natural Language Processing (EMNLP)*.

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. SParC: Cross-domain semantic parsing in context. In *Association for Computational Linguistics (ACL)*.

Zheng Yuan, Hao Chen, Zijin Hong, Qinggang Zhang, Feiran Huang, Qing Li, and Xiao Huang. 2026. Knapsack optimization-based schema linking for llm-based text-to-sql generation. In *International Conference on Data Engineering (ICDE)*.

Qinggang Zhang, Junnan Dong, Hao Chen, Wentao Li, Feiran Huang, and Xiao Huang. 2024. Structure guided large language model for sql generation. In *International Conference on Machine Learning (ICML)*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Association for Computational Linguistics (ACL): System Demonstrations*.

A Commitment Extraction Rules

Overview. For each candidate query y, we define its commitment set $S(y)$ as a finite set of clause-scoped atomic units extracted from the parsed SQL abstract syntax tree. Each unit records a localized semantic decision expressed by the query, covering what is projected, which predicates are imposed, which grouping keys are used, and how the results are ordered. We obtain these units by traversing the abstract syntax tree produced by `sqlglot` and applying the per-scope emission rules in Table 5.

Extraction procedure. Algorithm 2 constructs $S(y)$ using a scope-aware traversal. Each worklist entry pairs a query block with its scope path. Clause-level units are emitted only from the current scope, while immediate subqueries and set-operation branches are assigned distinct child paths and processed independently through the worklist.

Value and structural forms. Each emitted unit is stored in two forms. The value form preserves literal values, such as `region = 'West'`, while the structural form replaces literals with placeholders, such as `region = ?`. The non-monotonicity test in §2.2 compares value forms: a follow-up is marked as drift when $S(y_{\text{pre}}) \nsubseteq S(y_{\text{post}})$. We use the structural form only to identify value-level edits: a retracted unit and a newly introduced unit share the same structure but differ only in their literal values.

Algorithm 2: Commitment Extraction

Require: Candidate SQL query y
Ensure: Commitment set $S(y)$
Notation:
W: worklist of query-scope pairs (q, π)
ϵ: root scope
$\pi \oplus \ell$: extend path π with location ℓ
$\textsc{Branch}(q, j)$: path label for branch b_j

% Parse and initialize
$T \leftarrow \textsc{ParseSQL}(y)$; $S \leftarrow \emptyset$; $W \leftarrow [(T, \epsilon)]$
% Traverse query scopes
while $W \neq \emptyset$ **do**
  $(q, \pi) \leftarrow \textsc{Pop}(W)$
  if q is a set operation **then**
    for each ordered branch b_j of q **do**
      $\pi_j \leftarrow \pi \oplus \textsc{Branch}(q, j)$; $\textsc{Push}(W, (b_j, \pi_j))$
    end for
    continue
  end if
% Emit current-scope commitments
  for each current-scope clause node v of q **do**
    for each unit u emitted from v by Table 5 **do**
      $u^{\text{val}} \leftarrow \textsc{CanonValue}(u)$; $u^{\text{str}} \leftarrow \textsc{CanonStruct}(u)$
      $S \leftarrow S \cup \{(\pi, u^{\text{val}}, u^{\text{str}})\}$
    end for
  end for
% Enqueue nested subqueries
  for each immediate subquery q_j at ℓ_j in q **do**
    $\pi_j \leftarrow \pi \oplus \ell_j$; $\textsc{Push}(W, (q_j, \pi_j))$
  end for
end while
return S

B Construction Pipeline Details

This appendix expands the construction pipeline summarized in §3. For added clarity, We detail anchor selection, chain injection, drift injection, and evidence rewriting in order, specifying how candidates are selected, generated, filtered, and verified before their inclusion in TIDE-BENCH.

B.1 Chain Injection

Generation. We use GPT-5.1 to generate the chain candidates. For each BIRD entry, the model produces a JSON object with the initial question and the L1/L2 clarification pairs. The full prompt template and output schema are both shown in Figure 6.

Judging. We score each candidate with two LLM judges, GPT-5.3-Codex and DeepSeek-V4-Pro. Each judge applies the three criteria in Table 6, in priority order. A candidate is accepted only if both judges pass all three criteria. The judges agree on 94.7%, and 695/830 candidates pass (83.7%).

B.2 Drift Injection

Generation. We use GPT-5.1 to generate drift candidates. For each chain-validated anchor, the model receives a drift subtype: value-, constraint-, or projection-level. It returns a JSON object with the drift utterance and post-drift SQL. The prompt template and output schema are shown in Figure 7.

AST-level validation. Each candidate is checked by a rule-based validator before LLM judging. The validator tests the operational definition of §2.2 and the scope constraints of the assigned subtype, as summarized in Table 7. The validation reuses the AST procedure from Appendix A; 554 of 695 validated chain candidates pass this stage (79.7%).

Cross-validation. AST-passing candidates are scored by two LLM judges, Claude-Sonnet-4.6 and DeepSeek-V4-Pro. The judges receive the anchor SQL, drift utterance, and post-drift SQL. They apply two criteria in priority order: (i) **Drift fit**, requiring the post-drift SQL to match the assigned subtype and user drift; and (ii) **Stability**, requiring

Scope	Emission rule	Example
WHERE	Emit each conjunct after top-level AND decomposition; keep predicates under OR or NOT intact.	WHERE a=1 AND b=2 → {a=1, b=2}
SELECT	Emit each SELECT item; SELECT * is one marker.	SELECT c, AVG(v) → {c, AVG(v)}
GROUP BY	Emit each grouping expression.	GROUP BY c, d → {c, d}
ORDER BY	Emit each ordering expression with its direction.	ORDER BY v DESC → {v DESC}
HAVING	Emit each conjunct after top-level AND decomposition.	HAVING COUNT(*)>5 → {COUNT(*)>5}
LIMIT/OFFSET	Emit the clause with its numeric argument.	LIMIT 10 → {LIMIT 10}

Table 5: Scope-specific emission rules for constructing the commitment set $S(y)$. Each emitted unit is paired with its query-scope path and canonicalized into value-preserving and structural forms before insertion for drift testing.

Criterion	Requirement
Plausibility	L1 and L2 options are plausible and distinct, with no strawmen or rewordings.
Re-ask	Each layer adds a dimension absent from the initial question.
Counterfactuality	L1 must eliminate or transform the core of L2, rather than leave it invariant.

Table 6: LLM judging rubric for chain injection. Both judges must pass three criteria for benchmark inclusion.

untouched units to remain unchanged. A candidate is accepted only if both judges pass both criteria, requiring agreement on validity. The judges agree on 96.6%, and 514/554 candidates pass (92.8%).

B.3 Evidence Rewriting

Motivation. BIRD's evidence field can reveal terms that should stay ambiguous in TIDE-BENCH. For example, it may map "top clients" to "clients ranked by spend," fixing the L1 choice ahead of dialogue. This early cue weakens the chain ambiguity we aim to test. The rewrite removes such cues while preserving schema-essential mappings.

Protocol. For each sample, GPT-5.1 rewrites the evidence using the source evidence, anchor SQL, and generated L1/L2 clarifications. The rewrite removes cues that resolve the ambiguity while retaining mappings needed for SQL interpretation.

Reconstructability. We test whether the rewritten evidence preserves schema-essential information without itself revealing the L1/L2 resolution. For 695 candidates, we evaluate Gemini-3-Flash under three settings in Table 8: the original question and evidence (A), the ambiguous question and rewritten evidence with dialogue (B), and the same inputs without dialogue (C). The small A–B gap of 0.8 pp shows that the rewritten evidence preserves what is needed for reconstruction when paired with dialogue. The large B–C gap of 19.2 pp shows that the rewritten evidence alone does not leak resolution.

C Human Validation Details

This appendix supplements §3.3 with the annotator protocol, the full checklist, the adjudication rules, and the raw counts behind the item-level statistics.

C.1 Annotator Recruitment and Protocol

The audit is performed by five annotators: three NLP PhD students with text-to-SQL research experience, and two recruited annotators with production SQL backgrounds. The two external annotators are compensated at hourly rates commensurate with local expectations for technical annotation work. Before annotating the audit set, all five annotators completed a one-hour calibration round on a held-out set of ten triplets, drawn independently from the audit sample. The calibration round aligned checklist-item interpretation; calibration triplets are excluded from all reported statistics.

Each of the 250 audit triplets is assigned to all five annotators, and each annotator works independently. Annotators see only the sample contents—initial question, clarification structure, drift utterance (if applicable), and ground-truth SQL—and are blind to all automated verdicts produced during construction, as well as to any model evaluation results. The annotation interface presents one sample at a time, with the checklist items relevant to that sample's type. Items that are not applicable to a given sample type (e.g., joint-boundary items on a CHAIN-ONLY sample) are simply not shown there.

C.2 Full Checklist

Each annotator answers yes/no on the following items. The four chain criteria apply to CHAIN-ONLY and JOINT samples; the three drift criteria apply to DRIFT-ONLY and JOINT samples; the joint criterion applies only to JOINT samples, where both patterns co-occur within the same dialogue.

Step	Check	Pass condition	Failure signal
1	Output schema	Required JSON fields are present.	`json_format`
2	SQL parsing	`post_drift_sql` parses and yields extractable commitments.	`sql_parse`
3	SQL execution	`post_drift_sql` executes on the target database.	`sql_execute`
4	Non-empty result	The post-drift query returns at least one row.	`empty_result`
5	Observable effect	Pre- and post-drift result sets differ.	`identical_result`
6	Non-monotonicity	The commitment diff satisfies $S_{\text{pre}} \nsubseteq S_{\text{post}}$.	`no_change`; `monotonic`
7	Subtype match	The realized drift subtype matches the assigned subtype.	`subtype_mismatch`
8	Subtype detail	For projection-level samples, the fine-grained subtype matches.	`fine_subtype`

Table 7: Sequential AST-level checks for validating drift candidates. Checks are applied in order; the final column reports the rejection signal emitted upon failure, and only candidates passing all checks undergo LLM judging.

Cond.	Evidence	Question	Dialogue	Acc.	Δ
A	orig.	orig.	—	51.2	—
B	rew.	ambiguous	✓	50.4	-0.8
C	rew.	ambiguous	—	31.2	-20.0

Table 8: Reconstructability of rewritten evidence under three input settings ($n = 695$). "orig." / "rew." denote original / rewritten BIRD evidence; ✓marks dialogue input; Δ is the overall accuracy change (pp) from A.

Chain items.

1. Is the initial user utterance genuinely underspecified with respect to the business concept clarified at L_1?
2. Are both L_1 options plausible interpretations of that concept that a real user might intend?
3. Are both L_2 options plausible interpretations of the concept clarified at L_2?
4. Under the L_1 alternative, is the core of L_2 eliminated or transformed, rather than simply remaining unchanged in that context?

Drift items.

1. Does the drift utterance retract or replace at least one commitment from the pre-drift turn, rather than only extending it?
2. Does the post-drift SQL correctly implement what the drift utterance asks for?
3. Does the post-drift SQL leave every commitment not targeted by the drift unchanged?

Joint item.

1. Does the dialogue read coherently across the chain-to-drift boundary, with no abrupt topic shift and no contradiction between turns?

C.3 Adjudication Rules

A sample is accepted under *majority acceptance* iff at least three of the five annotators answer yes on every applicable item; it is accepted under *strict acceptance* iff all five annotators answer yes on every applicable item. Items that are not applicable to a sample's type are excluded from both the numerator and the denominator for that sample only.

Fleiss' κ in Table 1 is computed over binary per-annotator accept/reject verdict at the sample level, where a verdict is yes iff that annotator answered yes on every applicable item. The item-level κ values below are computed analogously over binary per-annotator verdicts for every checklist item.

C.4 Item-Level Statistics

Table 1 reports sample-level acceptance after aggregating all applicable checklist criteria. To identify where rejection and disagreement arise, Table 9 reports majority pass rates and Fleiss' κ for three core criteria: counterfactual dependency, non-monotonic retraction, and joint transition coherence. A criterion passes for a given sample iff at least three of the five annotators mark it as satisfied.

All three item-level κ values lie between 0.71 and 0.83, indicating substantial to almost-perfect agreement. The lowest value ($\kappa = 0.71$, on joint transition coherence) is consistent with the more pragmatic nature of that judgment, while the two formal criteria show higher agreement overall.

C.5 Dominant Failure Modes

We summarize the dominant reasons for rejection under majority acceptance for each sample type:

Chain-only. Of the 17 rejected samples, the most common failure concerns counterfactual depen-

Item	Pass Rate	Fleiss' κ
Counterfactual dependency	93.4%	0.79
Non-monotonic retraction	94.8%	0.83
Joint transition coherence	91.6%	0.71

Table 9: Majority pass rates and annotator agreement for three core validation criteria. A criterion passes when at least three of five annotators mark it as satisfied; Fleiss' κ uses binary criterion-level verdicts across annotators.

dency. The remaining rejections involve the plausibility of the L_1 or L_2 options, while initial underspecification rarely emerges as a rejection reason.

Drift-only. Of the 11 rejected samples, the most common failure concerns post-drift SQL fidelity to the drift utterance. Fewer failures involve the preservation of non-targeted commitments, while non-monotonic retraction itself is rarely flagged.

Joint. The 29 rejected samples most often fail on counterfactual dependency, non-monotonic retraction, or chain-to-drift boundary coherence. The remaining rejections reflect less frequent failures on other checklist items across the full audit set.

D Evaluation Setup

Models. We evaluate 12 frontier LLMs, all accessed via official APIs: Claude-Sonnet-4.6, GPT-4o, GPT-5.4, o3-mini, o4-mini, Gemini-2.5-Flash, Grok-4-Fast, Grok-4.1-Fast, Qwen2.5-72B, Qwen-Max, Llama-3.3-70B, and DeepSeek-V4-Flash. Both reasoning and non-reasoning models are included to span the current capability frontier; exact model configurations are also listed in Table 10.

Dialogue harness. Each evaluation runs an agent–simulator dialogue loop bounded by a maximum of 20 turns. Each agent turn yields either a clarification request or a candidate SQL query, parsed from a JSON response under the schema described below. Malformed responses trigger a single retry; persistent failures count as an abstention and the trajectory terminates. Candidate SQL is executed against the corresponding BIRD SQLite database under read-only mode; execution results are not surfaced back to the agent at any subsequent turn.

Inputs to the agent. At each turn, the agent receives the database DDL, the rewritten evidence (§3.2, with clauses pre-resolving L1 or L2 removed), and the dialogue history through that turn. The agent prompt is fixed across all 12 models.

Display name	Model configuration
Claude Sonnet 4.6	`claude-sonnet-4.6`
Qwen2.5-72B	`qwen2.5-72b-instruct`
o4-mini	`o4-mini`
GPT-4o	`gpt-4o-2024-11-20`
GPT-5.4	`gpt-5.4`
Llama-3.3-70B	`llama-3.3-70b-instruct`
o3-mini	`o3-mini`
Gemini-2.5-Flash	`gemini-2.5-flash-nothinking`
Grok-4.1-Fast	`grok-4-1-fast-reasoning`
Qwen-Max	`qwen-max`
DeepSeek-V4-Flash	`deepseek-chat`
Grok-4-Fast	`grok-4-fast-non-reasoning`

Table 10: Display names and API model configurations for the 12 evaluated LLMs. The configuration strings identify exact model variants and, where applicable, their reasoning modes used in all reported evaluations.

Agent prompt. The system prompt template is shown in Figure 8. It defines the required JSON response schema and the permitted agent actions.

E User Simulator Details

Two-stage design. The simulator follows the function-driven design of Huo et al. (2026). At each user turn responding to an agent clarification, a Stage 1 router classifies the request into one of three actions: AMB (resolve a pre-annotated ambiguity from the sample's chain structure), LOC (answer a localizable SQL-level detail by inspecting the ground-truth SQL), or UNA (refuse a request that the user cannot reasonably be expected to answer, such as questions about database-internal identifiers). Stage 2 then generates the natural-language response conditioned on the routed action and the relevant annotation field for that sample.

Adaptations for TIDE-BENCH. Three changes accommodate the two-layer chain structure (§2.1):

- L2-targeted clarifications issued while L1 is still unresolved are routed as LOC rather than AMB, since the L2 annotation only becomes a valid AMB target after L1 has been explicitly chosen.
- AST traversal during Stage 2 generation is restricted to a single atomic node per response, preventing the simulator from volunteering information beyond what the agent asked for.
- A post-generation validator screens responses for out-of-scope SQL fragments or schema identifiers; offending responses are regenerated.

Split	Accuracy (%)
UNA (refusal)	97.4
AMB (labeled)	94.3
LOC (unlabeled)	93.0

Table 11: Stage 1 router accuracy on the three USERSIM-GUARD splits. UNA tests refusal decisions, AMB tests labeled ambiguity routing, and LOC tests unlabeled SQL-detail localization during interaction.

Configuration. Both Stage 1 routing and Stage 2 generation use GPT-5.1 via API, with the same temperature and decoding settings as the agent harness (Appendix D) to ensure configuration consistency.

Validation. We evaluate the simulator on the USERSIM-GUARD diagnostic suite, which probes the three router actions on held-out queries. Per-split accuracy is reported in Table 11. To verify behavior in full dialogues, the five annotators from §3.3 additionally labeled a stratified random sample of agent–simulator traces (Fleiss' $\kappa = 0.802$).

F Full Per-Model Results

Tables 13 and 14 provide the complete per-model breakdowns behind the summary in §5.1. Table 13 reports chain-layer identification on Chain-only and Joint samples, together with ask rates across Chain-only, Joint, and Drift-only settings. Table 14 reports drift recognition and resolution on Drift-only and Joint samples, broken down by drift subtype. Two patterns hold across all twelve models.

G Ground-Truth SQL Complexity

This appendix substantiates two claims made in §5: (a) value-level drift produces post-drift SQL that is structurally identical to the pre-drift query (§5.4), and (b) constraint-level drift produces the simplest post-drift SQL among the three subtypes (§5.3). We characterize each drift subtype by comparing the pre-drift ground-truth SQL y^*_{pre} against the post-drift ground-truth SQL y^*_{post} over all 514 triplets. All anchors parse successfully under our analyzer.

G.1 Per-Subtype Structural Effects

Table 12 reports the mean change (Δ) in five key structural complexity measures from pre- to post-drift, separately for each drift subtype. Values are means over the corresponding subtype subset; negative values indicate that the post-drift SQL is simpler than the pre-drift SQL along that dimension.

Value-level. Every measure shows zero mean change. This is by design: value-level drift only replaces a literal inside a WHERE predicate, with predicate structure and all other commitments preserved (§2.2). The AST validator explicitly enforces this invariant for every candidate during construction, and the tier distribution before and after drift is identical (44/107/24 in basic/intermediate/advanced). For value-level samples, y^*_{pre} and y^*_{post} are therefore structurally indistinguishable at the level of these measures, isolating dialogue state itself as the only variable that still differs between the paired Drift-only and Joint samples (§5.4).

	Value (n=175)	Constraint (n=161)	Projection (n=178)
SQL Tokens	+0.00	−7.47	−5.87
WHERE Predicates	+0.00	−1.11	−0.03
JOIN Count	+0.00	+0.00	−0.03
Aggregation Count	+0.00	+0.00	−0.26
Subquery Depth	+0.00	+0.00	−0.01

Table 12: Mean change in structural complexity from y^*_{pre} to y^*_{post}, by drift subtype. Values are means over the corresponding subtype subset. Negative values indicate reduced post-drift complexity along the corresponding dimension, with zero denoting no change on average.

Constraint-level. WHERE predicates drop by 1.11 on average across constraint-level samples, consistent with the subtype definition (retraction of one or more WHERE predicates without introducing new constraints, §2.2). Token count drops by 7.47, the largest absolute reduction among the three subtypes. Critically, JOIN, aggregation, and subquery structures are all unchanged on average. The tier distribution shifts substantially toward lower complexity (44/104/13 → 76/79/6 in basic/intermediate/advanced), with roughly 25 samples moving from intermediate to basic. By every measure reported, constraint-level drift yields the simplest post-drift SQL of the three subtypes (used in §5.3).

Projection-level. Aggregation count drops by 0.26 on average, while WHERE predicates (-0.03) and JOIN count (-0.03) are effectively unchanged. This matches the subtype definition: changes are confined to SELECT, GROUP BY, ORDER BY, HAVING, LIMIT, or OFFSET, with WHERE predicates preserved (§2.2). The aggregation reduction reflects projection changes that move from aggregate to non-aggregate queries (e.g., from `SELECT AVG(profit)` to a top-k row listing query), while largely preserving the filtering and join structure.

#	Model	Chain-only				Joint				Drift-only
		L1-IR	L2-IR^p	JIR	AR	L1-IR	L2-IR^p	JIR	AR	AR
1	Claude-Sonnet-4.6	5.6	13.8	0.6	9.7	6.0	11.1	1.4	10.5	0.4
2	Qwen2.5-72B	36.2	3.3	1.2	60.1	35.8	3.9	1.8	60.7	20.0
3	o4-mini	27.8	11.5	1.4	54.9	29.2	12.6	2.1	55.4	17.5
4	GPT-4o	31.3	5.3	1.4	48.2	30.0	6.2	4.1	47.3	8.9
5	GPT-5.4	19.5	11.7	2.5	30.0	20.0	10.9	3.1	31.7	2.5
6	Llama-3.3-70B	32.7	12.1	3.7	58.2	34.0	12.1	3.3	58.0	11.1
7	o3-mini	21.6	9.9	1.8	33.5	23.7	9.7	2.1	34.8	4.3
8	Gemini-2.5-Flash	15.6	1.2	1.2	30.5	12.6	0.4	0.4	28.8	4.1
9	Grok-4.1-Fast	26.1	12.6	2.9	39.5	25.7	11.5	2.1	42.2	5.4
10	Qwen-Max	27.8	1.4	1.4	43.8	26.5	2.1	2.1	44.9	4.3
11	DeepSeek-V4-Flash	8.4	5.1	0.4	14.6	7.8	5.1	0.6	12.8	0.8
12	Grok-4-Fast	18.9	11.7	1.6	34.6	18.7	11.1	1.8	36.4	5.4

Table 13: Chain-layer identification and ask rates (%) on Chain-only, Joint, and Drift-only samples. L1-IR and L2-IR^p measure whether the agent identifies the corresponding clarification layer; JIR requires both layers to be identified within one trajectory, while AR records whether the agent issues any clarification request. L2-IR^p is measured under the p-protocol with L1 pre-injected; all other metrics use the f-protocol throughout all evaluations.

#	Model	Drift-only DRA				Joint DRA				DRR	
		Value	Constraint	Projection	All	Value	Constraint	Projection	All	Drift	Joint
1	Claude-Sonnet-4.6	**53.4**	48.5	**64.0**	**55.4**	38.6	34.4	**55.4**	**43.0**	**98.2**	96.9
2	Qwen2.5-72B	47.2	46.0	58.3	50.6	38.3	35.6	45.7	40.0	**98.2**	95.8
3	o4-mini	46.6	44.2	59.4	50.2	**39.7**	32.5	49.7	40.8	94.9	93.7
4	GPT-4o	49.4	45.1	62.3	52.4	34.1	35.0	51.4	40.3	97.3	96.9
5	GPT-5.4	48.3	42.3	61.7	51.0	35.8	34.4	52.6	41.1	97.3	94.7
6	Llama-3.3-70B	44.9	44.8	52.0	47.3	38.1	**39.9**	45.1	41.1	97.4	**97.1**
7	o3-mini	44.3	42.3	55.4	47.5	35.2	30.1	51.4	39.1	95.7	94.9
8	Gemini-2.5-Flash	44.3	**49.7**	47.4	47.1	36.4	39.3	45.7	40.5	97.4	95.7
9	Grok-4.1-Fast	44.3	42.9	58.3	48.6	33.5	30.1	46.3	36.8	96.5	92.0
10	Qwen-Max	46.0	47.9	54.3	49.4	34.7	35.6	41.1	37.2	97.9	96.1
11	DeepSeek-V4-Flash	45.5	43.6	54.3	47.9	29.0	32.5	46.9	36.2	97.3	96.7
12	Grok-4-Fast	40.9	37.4	57.7	45.5	29.0	28.8	47.4	35.2	96.4	93.9

Table 14: Drift handling metrics (%) under the paired factorial design. The best and second-best results in each column are highlighted in bold and underlined, respectively. Drift-only DRA and Joint DRA are reported overall (All) and by drift subtype. The DRR columns report aggregate Drift Recognition Rate on Drift-only and Joint samples. Higher values indicate more reliable recognition and resolution of injected drift across interaction settings.

System prompt.

You are an expert data annotator for a conversational Text-to-SQL benchmark thatstudies chained ambiguity. Your task is to construct CHAIN AMBIGUITY samples on top of the BIRD benchmark.

A chained ambiguity sample is a dialogue where:

- A user issues an initially ambiguous question.
- The system asks a first clarification (L1), whose options represent two genuinely plausible interpretations of an abstract business concept.
- The user selects one option, `L1_chosen`.
- ONLY AFTER L1 is resolved does a second ambiguity (L2) become meaningful. L2 must be activated by the specific L1 answer, not be a standalone ambiguity.
- The user answers L2, and the system outputs the final SQL, which must match the provided BIRD ground-truth SQL exactly.

Hard constraints.

C1. L1 must be a binary choice. `L1_options` contains exactly 2 items. Both options must represent plausible user intents: a real user could reasonably have meant either one.

C2. L2 must be a binary choice. `L2_options` contains exactly 2 items. Both options must represent plausible user intents: a real user could reasonably have meant either one.

C3. Ground-truth path. `L1_chosen` must be the option that, together with `L2_chosen`, leads to the BIRD ground-truth SQL. The other L1 option must lead to a different but valid SQL.

C4. Conditional dependency. L2 must become meaningful only after `L1_chosen` is selected. Strip any opening reference to `L1_chosen` from `L2_question` and check the remaining core. If the same core question could be asked under the L1 alternative with the same words, then L2 is independent, not a chain. Redesign.

C5. Content novelty. Each clarification layer must introduce information not already present in the initial question. If a layer only re-asks or narrows concrete concepts, entities, filters, named values, or constraints already stated by the user, redesign it to cover a genuinely new dimension.

C6. Literal placement. Category A literals are ambiguity-relevant literals that operationalize an abstract business term; they must surface through L1 or L2 clarification. Category B literals are context parameters, including LIMIT values, dates, named subjects, places, products, and arbitrary numeric query parameters; they must appear in the initial question. Category C literals are explicitly mapped by the BIRD evidence field and need not appear in the dialogue.

C7. Initial question. The initial question must contain: (a) all Category B literals from the BIRD SQL; (b) the abstract business term whose operationalization is ambiguous; and (c) no Category A literals. It must be ambiguous with respect to L1 only and should not pre-expose L2.

C8. Correspondence to BIRD SQL. The resolved dialogue must fully specify the BIRD ground-truth SQL. Every filter, LIMIT, aggregation, ordering, and join must trace back to either a Category B literal in the initial question, a Category A literal grounded in L1/L2, or a Category C evidence mapping.

C9. Naturalness. User utterances must use natural business language. Users never reference SQL column names, table names, or schema identifiers. Clarification must be at the business-concept level, not the schema-detail level.

C10. Alternative SQL feasibility. Both the L1 alternative and L2 alternative must correspond to SQL actually writable against the provided schema. Do not invent columns, tables, or relationships. For each alternative, verify that the referenced columns, tables, and joins exist in the schema.

User prompt template.

Construct a CHAIN AMBIGUITY sample from a BIRD entry containing `db_id`, `schema_ddl`, `sample_rows`, `bird_question`, `bird_evidence`, `bird_sql`, and `complexity_tier`. Generate the JSON object as specified in the system prompt. Remember: L2 must genuinely depend on `L1_chosen`; Category B literals go in the initial question; Category A literals go in L1/L2; both L1 and L2 must introduce new information; every part of the SQL must be grounded; and both alternatives must use only the provided schema.

Output protocol.

Output only a JSON object containing `initial_ambiguous_question`, `L1_question`, `L1_options`, `L1_chosen`, `L1_alternative_sql_impact`, `L2_question`, `L2_options`, `L2_chosen`, `L2_alternative_sql_impact`, `L1_user_answer_utterance`, `L2_user_answer_utterance`, and `chain_rationale`; no markdown fences, reasoning, prose, or commentary.

Figure 6: Compressed version of the prompt used to generate chain ambiguity samples. The full prompt will be released with the code upon publication.

System prompt.

You are a data annotation specialist constructing INTENT DRIFT samples for a multi-turn conversational Text-to-SQL benchmark.

Core concept: Intent drift is a non-monotonic change in a user's query intent during conversation. Formally, let S_{pre} be the set of atomic commitments encoded in the pre-drift SQL, and S_{post} be the set of atomic commitments encoded in the post-drift SQL. A valid drift requires $S_{\text{pre}} \nsubseteq S_{\text{post}}$, i.e., at least one prior commitment must be retracted, replaced, or removed. This distinguishes drift from cooperative follow-up, which is monotonic.

Hard rules.

R1. Non-monotonicity is mandatory. The post-drift SQL must retract, replace, or remove at least one commitment from the pre-drift SQL. Pure additions, such as appending new predicates while keeping all old ones, are not drift.

R2. Drift subtype constraints are strict. The assigned drift subtype is one of three. For value-level, replace exactly one literal value in a WHERE predicate; SELECT, predicate count, and predicate structure must remain identical. For constraint-level, delete one or more top-level WHERE predicates; SELECT must remain identical, and no new predicates or value changes are allowed. For projection-level, change the SELECT projection, including columns, aggregations, ORDER BY, or LIMIT; this SELECT shift must be the dominant change.

R3. Executability. The post-drift SQL must be executable on the given schema, use only existing tables and columns, and follow SQLite dialect.

R4. Non-empty result. The post-drift SQL must return a non-empty result set.

R5. Different result. The post-drift SQL must produce different results from the pre-drift SQL. If both return the same rows, the sample fails.

R6. Natural drift utterance. `drift_utterance` must be natural first-person speech, not a mechanical instruction. It should use colloquial transition words such as "wait", "actually", "forget that", "on second thought", "scratch that", or "let me change that". Length: 1–3 sentences.

R7. Projection-level fine-grained subtype. For projection-level only, declare `intent_subtype` as either `aspect_change` or `target_change`. In `aspect_change`, the core entities remain the same, while the presentation viewpoint shifts. In `target_change`, the core entity, grouping key, FROM tables, or WHERE column set changes.

Drift-subtype-specific instructions.

Value-level: replace exactly one literal value in a WHERE predicate, such as changing `year = 2021` to `year = 2022`. Do not change SELECT, GROUP BY, ORDER BY, HAVING, LIMIT, JOIN structure, predicate count, operator, or filtered column. The new value must be plausible and must produce a different, non-empty result set. The utterance should explicitly signal which value is being changed.

Constraint-level: delete one or more top-level AND-joined WHERE predicates. Do not change SELECT, GROUP BY, ORDER BY, HAVING, LIMIT, JOIN structure, or retained predicate values. Do not remove sub-components of OR expressions, and do not remove predicates inside nested subqueries. Avoid peeling off predicates that act as single-row structural anchors; the removal should broaden an otherwise stable query.

Projection-level: change the SELECT projection in a substantive way, such as changing the returned columns, aggregation, GROUP BY, ORDER BY, or LIMIT. Declare `intent_subtype`. Use `aspect_change` when FROM tables, WHERE column set, and GROUP BY keys remain identical and only the presentation viewpoint shifts. Use `target_change` when at least one of FROM tables, GROUP BY keys, or WHERE column set differs.

User prompt template.

Construct a DRIFT INJECTION sample from a BIRD entry containing `db_id`, `schema_ddl`, `sample_rows`, `bird_question`, `bird_evidence`, `bird_sql`, `complexity_tier`, and `assigned_drift_subtype`. The original question is the fixed pre-drift user utterance, and `bird_sql` is the pre-drift SQL. Generate only the user's next turn, `drift_utterance`, and the corresponding `post_drift_sql`. The output must satisfy the assigned drift subtype, executability, non-empty result, different result, and non-monotonic commitment change.

Output protocol.

Output only a JSON object containing `drift_utterance`, `post_drift_sql`, `drift_subtype_claimed`, `intent_subtype`, `pre_commitments_narrative`, `post_commitments_narrative`, and `drift_rationale`. If `drift_subtype_claimed` is not projection-level, set `intent_subtype` to null. Return no markdown fences, prose, or commentary outside the JSON.

Figure 7: Compressed version of the prompt used to generate intent drift samples. The full prompt will be released with the code upon publication.

System prompt.

You are a text-to-SQL agent. Your task is to understand a user's natural-language data question and produce a correct SQL query for the database described below.

Database schema.

`{schema_ddl}`

Evidence / domain context.

`{evidence}`. Evidence may be empty for some tasks; if so, rely on schema and dialogue alone.

When to ask vs. when to answer.

Carefully read the user's request. If anything is ambiguous or under-specified, ask for clarification rather than making assumptions. When sufficient information is available, produce the SQL query.

Clarification rules.

R1. Ask at most one clarification question per turn. Focus each question on a single ambiguity.

R2. When asking, offer 2–3 discrete options where appropriate to make it easy for the user to choose.

R3. Do not ask about information already determinable from the schema or evidence above.

R4. Do not ask the user to confirm specific table names, column names, or stored values: these are database-internal facts the user does not know.

Handling intent change.

If the user changes their request mid-dialogue (e.g., "actually, instead of X, give me Y"), respect the new request and produce the updated SQL based on the latest user intent. Do not revert to the original request.

SQL output rules.

S1. When producing SQL, the content must contain only the raw SQL query: no explanation, no markdown code fences, no markdown link syntax such as `[col](url)`.

S2. Produce valid SQLite-compatible SQL unless the schema implies another dialect.

S3. Use the exact column and table names as they appear in the schema, preserving backticks, quotes, and case as shown.

Output protocol.

Respond with a single JSON object and nothing else: `{"type": "ask", "content": <clarification question>}` or `{"type": "sql", "content": <complete SQL query>}`.

Figure 8: Agent system prompt template used uniformly across all 12 evaluated models. `{schema_ddl}` and `{evidence}` are populated per sample.