Natural Language Interfaces for Databases: What Changes for SQL-Literate Users?

PANOS IPEIROTIS, Stern School of Business, New York University, USA
HAOTIAN ZHENG, Tandon School of Engineering, New York University, USA

Natural Language Interfaces for Databases (NLIDBs) let users query data in everyday language instead of SQL, and recent systems translate those questions accurately. Accuracy says little about the work of querying: does an NLIDB remove that work or only reallocate it? We report a mixed-method, between-subjects study comparing SQL-LLM, a GPT-4o-backed NLIDB, with Snowflake, a traditional SQL platform. Twenty SQL-literate professionals and graduate students, ten per interface, each completed 12 tasks drawn from the BIRD benchmark. SQL-LLM cut completion time per query by about 31%, but the speedup did not buy accuracy: graded against the BIRD gold answers, SQL-LLM users were correct on 46% of queries versus 64% for Snowflake. Two analysts independently coded the think-aloud sessions to locate the effort. SQL-LLM users left schema navigation to the model and spent it verifying that the generated SQL matched their intent; Snowflake users explored the schema and built syntax by hand. Per-minute rates of the coded behaviors did not differ between interfaces. The interface reallocated the work of querying rather than removing it: users still had to verify the generated SQL, and an NLIDB that hid it would remove the step that let them trust the answer.

Additional Key Words and Phrases: Databases, Human-Computer Interaction, Natural Language Interfaces, Text-to-SQL, User Experience

ACM Reference Format:
Panos Ipeirotis and Haotian Zheng. 2026. Natural Language Interfaces for Databases: What Changes for SQL-Literate Users?. *ACM Trans. Manag. Inform. Syst.* 0, 0, Article 0 (2026), 26 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Natural Language Interfaces for Databases (NLIDBs) promise a simple bargain: type a question, get an answer, skip the SQL. The bargain responds to a long-standing complaint about how hard database systems are to use [6]. The translation NLIDBs perform, from a natural-language question to SQL (Text-to-SQL), is now technically credible, with recent large language models (LLMs) reaching roughly 85% accuracy on benchmarks like Spider [19]. The harder question is what users do with these systems: how they verify generated queries, where they get stuck, and whether the interface earns enough trust to displace a manual SQL workflow [7, 17]. One controlled study found that interactive interfaces for repairing generated SQL gave users no advantage over editing the query directly [17], but whether a modern LLM-based NLIDB beats a hand-written SQL workflow for SQL-literate users has not been tested head-to-head.

Authors' Contact Information: Panos Ipeirotis, panos@nyu.edu, Stern School of Business, New York University, Department of Technology, Operations, and Statistics, New York, NY, USA; Haotian Zheng, hz2687@nyu.edu, Tandon School of Engineering, New York University, Electrical and Computer Engineering, New York, NY, USA.

ACM 2158-6578/2026/0-ART0
https://doi.org/XXXXXXX.XXXXXXX

We run that test with a controlled, mixed-method comparison on the same task set: SQL-LLM, a commercially deployed NLIDB backed by GPT-4o, and Snowflake, a conventional SQL workflow. (Throughout, "Snowflake" names the manual-SQL arm; those participants used Snowflake's Snowsight editor.) Twenty SQL-literate professionals and graduate students each completed 12 tasks from the BIRD Text-to-SQL benchmark [12] on three relational schemas, recorded with screen capture, think-aloud protocols, and timing logs. Both arms ran against the same Snowflake warehouse and engine (Section 3.1.1), ruling out query-engine differences as a confound; what differs is the interface bundle: natural-language prompting, an editable generated query, and re-run.

On speed, the interface delivers a clear advantage: SQL-LLM cut completion time per query by 31% ($p = 0.03$), an estimated 212 s relative to the Snowflake group's mean (Section 4.1.1). On accuracy it shows no detectable gain, and the data cannot rule out a substantial loss: graded against the BIRD gold answers, SQL-LLM users answered 46% of queries correctly versus 64% for Snowflake, an 18-point difference favoring Snowflake ($p = 0.08$ at $N = 20$). Correct answers per unit time land near parity (4.6 versus 4.2 per hour), so what the interface demonstrably changes is where the user's time goes, not how much correct output comes back.

Switching interfaces reallocated the observable work of querying. Behavioral coding of the think-aloud transcripts, available for 11 of the 20 participants (4 SQL-LLM, 7 Snowflake), shows where the effort went; none of the coded behaviors differed detectably in per-minute rate at this sample size. SQL-LLM users left initial schema navigation to the model (on the task hints' role, see Section 3.2.2) and spent their attention verifying generated SQL against intent, while Snowflake users spent theirs on manual schema exploration and syntax construction.

This reallocation, more than the speedup, is what matters for adoption in an analytics organization. NLIDBs that promise to "democratize data access" still leave users checking whether the generated query matches the question they asked [22]. Our participants were SQL-literate and equipped to do that checking; whether the trade-off holds for users without SQL training is future work. For the users we tested, the verification burden survives the switch to natural language.

Contributions.

- **Effort reallocation.** A behavioral characterization, grounded in transcripts and screen recordings, of how querying effort reallocates between the two interfaces: from schema-and-syntax work under manual SQL to semantic verification of generated SQL under Text-to-SQL.
- **Head-to-head study.** A controlled comparative user study of a deployed GPT-4o NLIDB against a conventional SQL workflow, with both arms on the same Snowflake warehouse (20 participants, 12 tasks each, 11 think-aloud-coded sessions).
- **Time-and-accuracy evidence.** A mixed-effects analysis of completion time and a logistic analysis of accuracy, distinguishing where the interface helps from where the evidence runs out.
- **Adoption implications.** Interface design implications for organizations deploying NLIDBs, with explicit attention to the checking work that natural language input does not remove.

Section 2 positions the study against prior comparisons of NLIDBs with traditional query interfaces. Section 3 describes the setup and study design; Section 4 reports the results, with the qualitative analysis in Section 4.2. Section 5 discusses implications, Section 6 outlines future work, and Section 7 concludes.

2 Related Work

Benchmark accuracy for NLIDBs has climbed high enough that parsing is no longer the bottleneck; whether that accuracy helps real users is. The empirical user studies below converge on one

unresolved question: once an NLIDB is accurate enough to deploy, does it reduce the user's remaining effort or merely relocate it? Prior work measured usability, trust, and efficiency against traditional query interfaces, and one finding sets up the tension this study takes up: interactive repair interfaces gave no advantage over editing the generated SQL by hand [17].

2.1 From Benchmark Accuracy to the Verification Burden

Querying a database in plain language is an old promise, answering a long-standing complaint about how hard database systems are to use [6]; what changed recently is accuracy. On large-scale benchmarks like Spider [28], deep learning models improved Text-to-SQL accuracy [7], and GPT-4-based methods have pushed it further, reporting about 85% execution accuracy on the Spider test set [19].

High accuracy does not settle whether the system helps the person using it; the gap that remains is one of trust [22]. A residual error rate of roughly 15% [17] matters in business-critical domains, where incorrect data drives flawed decisions. Users respond by demanding near-perfect precision [7], yet struggle to detect subtle errors in system-generated SQL [17]. That struggle predates LLMs: from exam analyses [1] to student corpora of over 33,000 queries [21], studies of hand-written SQL show that trained users routinely write queries that are syntactically legal but semantically wrong (contradictory conditions, missing join predicates, wrong grouping), which the DBMS executes without warning.

The problem sharpens when a system behaves as a "black box," returning an answer with no way to check it. The model leaves a residual error rate; opacity makes those errors impossible to inspect, so users cannot calibrate their trust. Checking the generated SQL against the question asked is the one act that lets a user detect and manage the errors, though it does not shrink the error rate itself. That checking work is the verification burden, and it frames the design response the rest of this section traces: make the generated SQL visible so the user can verify it. An early vision proposed the complementary move, rendering a query back into natural language so the user could confirm it captured their intent [20]. Recent research follows this line, moving the user from accepting opaque output to inspecting it.

2.2 Comparing NLIDBs with Traditional Query Interfaces

Direct head-to-head evidence on natural-language querying versus manual SQL for SQL-literate users is old and sparse. The one controlled comparison found the two equally accurate and natural-language users somewhat faster, though with a restricted-vocabulary system in an idealized setting [24]. What a modern, LLM-based interface does for users who already know SQL has not been tested this way.

The most rigorous recent user study shifts the question from efficiency to verification. Ning et al. [17] tested interactive error-handling interfaces (a step-by-step decomposition view, a graph visualization, and a conversational dialogue) against a no-support baseline in which users directly edited the model-generated SQL, and found no significant difference in task completion time or accuracy: the added interfaces bought nothing over editing the generated query by hand. That study did not compare an NLIDB against a hand-written SQL workflow; ours does, for SQL-literate users.

The benefit prior work does report sits elsewhere, with users who lack SQL. NaLIR let people with minimal database experience formulate complex queries involving joins and aggregations that they could not otherwise construct [10, 11], widening the range of questions a non-technical user can ask. Together, these findings cast Text-to-SQL as a complement, not a replacement, most valuable to users without SQL skills or for queries hard to express otherwise. Our study does not

test non-technical users and makes no claim about them; it asks what a modern NLIDB does for the SQL-literate population, where direct evidence has been thin and dated.

2.3 Lowering the Verification Burden: Interactive and Explainable Systems

Interactive and explainable NLIDBs are the field's answer to the verification burden: each makes its logic transparent and hands the user something to verify. Their evaluations measure error discovery, repair success, confidence, and SQL comprehension, but none asks how a user's effort redistributes between construction and verification when the interface changes.

Decomposition and Step-by-Step Explanations. One approach breaks a complex query into simpler steps, each explained in natural language. The DIY system shows intermediate results for every sub-query, and Narechania et al. found that even non-expert users could debug queries by following its reasoning step by step [16]. STEPS extends this line with editable step-by-step natural-language explanations of the generated query, evaluated in a 24-participant user study by the group that later built SQLucid [23]. The step-by-step view lowers the cost of verification without removing it: the user still reads each step's output and judges whether it is right.

Visual Query Representations. Another strategy visualizes query structure. QueryVis [9] and SQLVis [14] render tables, joins, and filters as diagrams, and Leventidis et al. found that QueryVis diagrams let users interpret queries faster and with fewer errors than raw text [9]. The diagram is a cheaper route to the same verification: it speeds reading of a query the user must still confirm matches their intent.

Conversational Clarification and Direct Manipulation. A third direction opens a dialogue to resolve ambiguity. MISP [27] and DialSQL [3] ask follow-up questions to clarify intent (e.g., "When you say 'sales,' do you mean 'sales amount' or 'number of sales'?"). This handles simple disambiguation well, but users grow frustrated when the conversation repeats or misses the core error [17]. The dialogue moves verification upstream into the user's answers, yet still leaves them to judge whether the resulting query is correct.

2.4 Empirical Evidence on User Performance and Confidence

When these systems are tested on real users, the payoff tracks how well the design supports verification, not how clever the translation is. The Ning et al. study behind the null result above [17] traced that null to a common cause: users could not follow the system's mistakes. Transparency helps only when it makes the error checkable.

Combining modalities helps more. SQLucid pairs natural-language explanations with visual highlights and direct editability, letting users see how their input maps to the resulting SQL [22]. In user studies it raised task-completion accuracy to 85%, against 56% for MISP and 67% for DIY, narrowed the novice-expert gap, and lifted confidence (6.4/7 vs. 3.8/7 for MISP and 5.3/7 for DIY). Users found the system's mistakes easy to correct, the same lesson from the other side: the design has to support the user's verification burden, not just the model's translation.

Our study takes up that open question for SQL-literate users directly: a deployed GPT-4o-backed NLIDB against a conventional SQL editor on the same task set, both arms backed by the same Snowflake warehouse. Section 5 returns to where our results echo Ning et al. [17] and where they diverge. The behavioral pattern we document complements the SQLucid [22] finding that interactive explainability raises confidence: the observable work of querying shifts from schema and syntax to semantic verification rather than disappearing, and both results locate NLIDB usability in the user's verification burden, not the model's translation alone.

3 Methodology

We compare SQL-LLM and Snowflake in a between-subjects study. Twenty SQL-literate participants were randomly assigned to one of the two interface arms, ten per arm, and every participant attempted the same 12 tasks, drawn from the BIRD benchmark, on three relational schemas. The outcome measures are per-query completion time and answer accuracy graded against the BIRD gold answers, both collected for all 20 participants; behavioral coding of the think-aloud commentary covers the 11 sessions with codable audio. We first describe what both arms hold constant and what the SQL-LLM arm adds (Experimental Setup), then who took part, what they were asked to do, and what we measured (Study Design).

3.1 Experimental Setup

The two arms are identical except for the interface, so any difference in completion time or accuracy reflects the interface *bundle*, not the data, the schemas, or the execution engine. The estimand is the effect of that complete bundle against a manual editor: natural-language prompting, generation, validation, visible and editable SQL, conversational refinement. The design cannot isolate any single component, so we flag component-level statements later in the paper as design hypotheses rather than causal findings. This subsection describes the constants (the databases, the task set, and its difficulty levels), then the SQL-LLM system that supplies the manipulated interface.

3.1.1 Databases Used. The study used three relational databases from the BIRD dataset, each from a distinct domain. **Books** is a bibliographic schema of authors, books, publishers, and publication years, supporting queries on authorship, title filtering, and publication metrics. **Mondial Geo** is a geographic schema of countries, cities, populations, and regions that invites multi-table joins and aggregations, such as comparing populations across regions or filtering cities by country. **Legislator** covers U.S. legislator names, terms served, state affiliations, and demographics, supporting queries on service history and party affiliation.

All three were ingested as native tables in a single Snowflake database and queried through one consistently configured virtual warehouse (Snowflake's compute unit): the Snowflake arm through the standard Snowsight web UI, and the SQL-LLM arm through the SQL-LLM editor, which generated SQL against the same warehouse. Both arms therefore queried the same data over the same execution engine and compute configuration, which removes query-engine performance as a confound.

3.1.2 Task Selection. The study tasks come from BIRD (BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation) [12], a dataset of realistic natural-language questions mapped to SQL across multiple domains. We selected 12 tasks (4 per database) spanning common business-intelligence patterns (entity retrieval, aggregated reporting, nested-condition queries) and varying across three complexity levels (Easy, Medium, and Hard; Section 3.1.3).

3.1.3 Difficulty Categorization. To relate behavior to query difficulty, we categorized the 12 tasks into three levels:

- **Easy Level**: single-table selection with basic filters and projections.
- **Medium Level**: multi-table joins with aggregations or group-by operations.
- **Hard Level**: nested subqueries, multiple joins, and conditional filtering logic.

The set is balanced four tasks to a level; the per-database Easy/Medium/Hard split is Books 1/2/1, Mondial Geo 1/1/2, and Legislator 2/1/1. Levels reflect schema-comprehension demands and structural complexity, our own structural heuristics rather than BIRD's published difficulty labels.

Difficulty enters the analysis as a covariate in the linear mixed-effects model (Section 4.1.1), where it absorbs task heterogeneity across the 12 tasks, so the categorization has to be one the analysis can trust. We cannot cross-check it against BIRD's published labels: the study tasks are drawn from BIRD's *train* split, whose public release carries no per-question difficulty labels; BIRD provides difficulty only for its dev and test splits. Instead we validate the categorization against an objective structural-complexity score computed from each task's gold SQL. The score sums the tables referenced and aggregation calls, adds one point each for `GROUP BY`, `HAVING`, `ORDER BY`, and `DISTINCT` clauses, and weights nested subqueries and set operations double. Mean complexity rises monotonically across our levels (2.00 for Easy, 3.25 for Medium, 5.50 for Hard), and the rank correlation between our ordinal difficulty and the gold-SQL complexity is positive and moderately strong (Spearman $\rho = 0.69$), with the residual overlap concentrated at adjacent-level boundaries rather than spanning them. The scoring script `difficulty_structural_check.py` and its output `difficulty_structural_complexity.csv` are in the replication archive.

3.1.4 SQL-LLM System Overview. SQL-LLM is a commercially deployed NLIDB backed by OpenAI's GPT-4o (via the OpenAI API in spring 2025); we evaluate it under a pseudonym at the vendor's request. On top of the shared warehouse (Section 3.1.1), it adds the interface bundle of natural-language prompting, an editable generated query, and re-run.

Between the user and the model sits a layer that turns each natural-language question into validated SQL: it serializes the active database schema, assembles a few-shot text-to-SQL prompt with curated exemplars for that database, forwards the question to GPT-4o, and checks that the generated SQL runs against the active schema; queries that fail are rejected rather than displayed. The exemplars are fixed per database, not retrieved per question: each study database carries a curated set of 5–10 question/SQL pairs drawn from the BIRD and Spider training splits, attached at deployment, and the full per-database set accompanies every question on that database. Validated SQL appears in an editable editor (Figure 1) and executes against that same Snowflake warehouse when the user presses Run.

None of the 12 study tasks appears in any database's exemplar set: curation screened the exemplars from both benchmarks against the study tasks and excluded any pair matching a study task, so no question/SQL pair a participant attempted was ever shown to the model as an in-context example. This guarantee is deliberately scoped to the prompt layer: BIRD and Spider are public benchmarks, so GPT-4o may have encountered their questions, schemas, or near-duplicate templates during its own training, a channel no prompt-side screen can close. A stronger test would use privately authored tasks over schemas absent from public corpora; we flag that design in Future Work.

No human corrections or manual query rewrites were performed during the study: the validation layer rejects unrunnable queries but never repairs model output, and what the user submits via Run is executed verbatim against Snowflake. The behavior we report is therefore that of the deployed pipeline. Two instrumentation gaps remain. First, rejection events at the validation layer were not separately logged, so we cannot report how often the validator intervened, and any time a participant spent re-prompting after a rejection folds into that task's completion time rather than being isolated. Second, finer-grained interaction traces (initial generated SQL before edits, edit and regeneration sequences, chat turns) were not captured, which limits how directly we can measure the path from generated to submitted query. The qualitative analysis offsets this limitation only partially.

Interface affordances. Figure 1 shows the SQL-LLM editor as participants saw it, with the natural-language question and the model's generated SQL side by side. The generated SQL was always visible in the editor pane, so participants could read and confirm it against their intent before

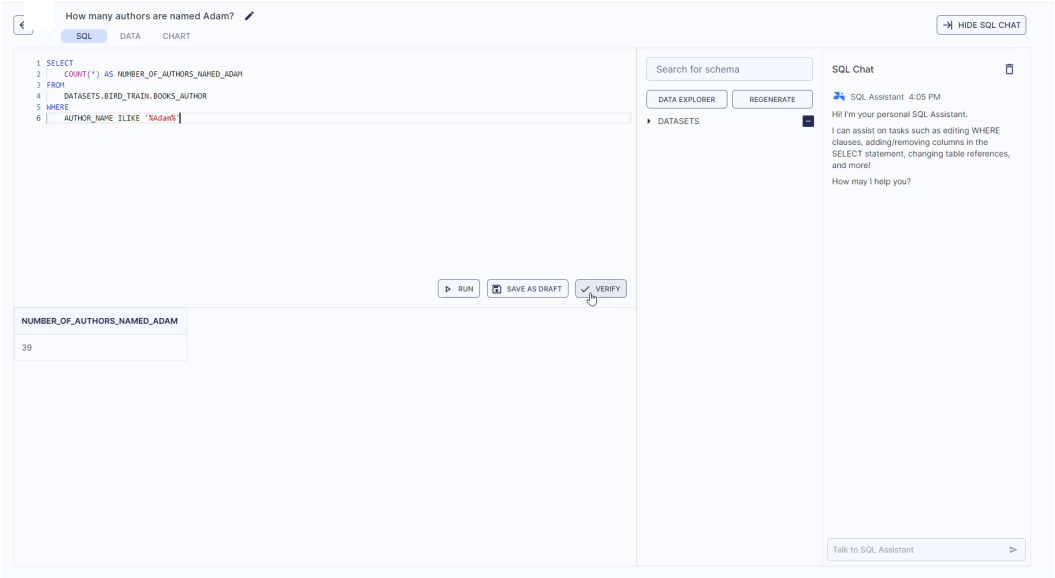

Fig. 1. The SQL-LLM editor. The natural-language question (top), the generated SQL (centre, editable), the result table (bottom), and the conversational SQL Chat pane (right) were all visible at once. Run, Regenerate, and Verify were the three primary actions; participants could also edit the SQL directly or issue follow-up instructions in the chat pane.

running. The editor was a free-form SQL textarea: participants could modify the model's output (rename columns, add filters, fix joins) and re-execute, or write SQL from scratch. A Regenerate button re-issued the original question to the model for an alternate query, and a conversational *SQL Chat* pane let participants issue follow-up natural-language instructions (e.g., "add a filter for year > 2000") that the model applied incrementally to the current query. Generation did not auto-execute: participants pressed Run to send the SQL to the warehouse, and results appeared as a table beneath the editor. A collapsible Datasets panel exposed the database, schema, table, and column hierarchy with type-to-filter search, and a Verify toggle let participants mark a result as accepted, a participant-side bookkeeping affordance that did not feed back into the model.

3.2 Study Design

3.2.1 Participants. We recruited 20 participants from Upwork, an online freelancing platform: working data analysts, business-intelligence professionals, and graduate students trained in databases. All reported at least basic SQL familiarity, and none had prior exposure to the SQL-LLM interface. Our findings therefore speak to SQL-literate users; we do not test, and do not claim, benefit for non-technical users who cannot read or write SQL.

All participants consented to be recorded. The protocol was approved by the New York University Institutional Review Board (IRB-FY2023-7595); in quotations and tables, participants appear only by neutral codes (P1–P10 for the SQL-LLM arm, P11–P20 for Snowflake), matching the public replication package. Each was paid a fixed $100 via Upwork and could take breaks or withdraw without penalty. Payment was not contingent on accuracy; Section 5.3 revisits the resulting satisficing risk.

3.2.2 Task Design. Each participant attempted the same 12 tasks on their assigned interface: the tasks selected above, posed as natural-language questions and grouped by database. Two representative example prompts appear below:

Database: `books`

- **B-Q1.** Which customer has made the most orders? Show his/her full name.
  Hint: Most orders refers to `MAX(COUNT(order_id))`; customer refers to `first_name, last_name`.

- **B-Q2.** How many authors are named Adam?
  Hint: Authors named Adam refers to `author_name LIKE 'Adam'`.

Each task carried a textual hint naming the target columns and operators (for example, mapping "most orders" to `MAX(COUNT(order_id))`). These hints clarified the prompt but pre-resolved work the interfaces would otherwise have differentiated: for an SQL-LLM user, near-verbatim prompt material; for a Snowflake user, the columns and operators that manual schema exploration would have had to find. Which arm this favored, on net, the design does not identify. Two hints warrant a flag. The B-Q2 hint's predicate `LIKE 'Adam'` conflicts with the gold answer's prefix match, a prompt/gold mismatch we analyze in Section 4. The B-Q4 hint's `COUNT(…WHERE …)` construction is pseudo-SQL, not a valid aggregate, so a Snowflake user had to translate it while an SQL-LLM user could pass it to the model as-is. Our results therefore characterize performance under heavily disambiguated framing rather than hint-free phrasing, a constraint we revisit in Section 5.3.

Participants recorded their screens with voice commentary while working, and recordings with usable audio were transcribed with Whisper for the qualitative analysis. That analysis codes reformulations as a behavioral measure, so the count must be comparable across arms. We count a reformulation symmetrically across arms: a substantively different submission of the same task, re-executed after the participant saw the previous result. In the SQL-LLM arm, that is a re-submitted changed natural-language prompt or an edited-and-re-executed returned query; in the Snowflake arm, an edited-and-re-run query or a semantically new attempt. An edit never re-executed does not count in either arm, and neither do trivial typo fixes or identical re-runs.

3.2.3 Procedure. We enrolled the first 20 applicants meeting the screening criteria and randomly assigned them to two equal groups of ten, one per arm; each completed all 12 tasks. Participants used their own hardware with a standardized browser configuration. Tasks could be attempted in any order, and participants could skip a task and return to it later. Each session had three phases:

(1) **Training** (15 minutes): participants explored the schemas of all three databases and completed two practice queries on the assigned system.
(2) **Testing** (1 hour and 45 minutes): participants executed the 12 tasks while screen and audio recordings captured their behavior.
(3) **Post-task debrief** (up to 30 minutes): participants answered an open-ended questionnaire on usability and perceived difficulty. No structured instrument (e.g., NASA-TLX [4] or trust scales) was administered, and these free-text responses are not systematically analyzed; the behavioral claims rest on the think-aloud record, not the debrief.

These durations were advisory: the remote, self-paced sessions let participants exceed an allotment without penalty, and many did. Summed on-task time across the 12 tasks had a median of roughly 100 minutes, but 10 of the 20 participants (3 SQL-LLM, 7 Snowflake) exceeded the nominal 105-minute testing allotment, the longest by more than a factor of two. Per-task completion times come from the screen recordings (Section 3.2.4) and are unaffected by these allotments.

Coverage of think-aloud audio. All 20 participants completed the 12 tasks and submitted screen recordings, and the timing and SQL-submission data we report (Tables 1, 4, and 5) cover all 20. Only 11 (4 SQL-LLM, 7 Snowflake) produced think-aloud audio good enough for behavioral coding; the other 9 had silent recordings, missing audio, or commentary too sparse to code. Because this screen was applied after participants used their interface, and one criterion (sparse commentary) could be influenced by the interface itself, the coded subset is a post-assignment selection whose arm-independence we cannot establish, though the failures were technical in character. Two checks bound the concern: the 11 coded participants did not differ detectably from the 9 uncoded in mean completion time (558 vs. 481 s, Welch $p = 0.49$) or accuracy (60% vs. 49%, $p = 0.33$), though both comparisons are low-powered. Transcript-derived analyses (Section 4.2) are limited to this $n = 11$ subset and should be read with this caveat; we retain all 20 for quantitative outcomes that do not depend on think-aloud commentary.

3.2.4 Measures. Two outcome measures cover all 20 participants: per-query completion time and answer accuracy. Completion time, taken from the screen recordings, sums the intervals a participant actively spent on each task and excludes time spent away. It is wall-clock on-task time and so includes model-generation latency, query execution, and within-task pauses without decomposing them, one of the two instrumentation gaps noted in Section 3.1.4. Timing covers 238 of the 240 participant–task cells: two observations (participant P17, tasks M-Q1 and M-Q2, Snowflake arm) are missing from the timing log, though both submissions exist and are graded, so accuracy denominators remain 120 per arm. Accuracy was graded against the BIRD gold answers: we executed each participant's final submitted query against the gold BIRD database and counted it correct only if its result set reproduced the gold answer. As a cross-check, three independent LLM judges scored the same submissions against the gold; their majority verdict agreed with the execution grade on 87.5% of queries. Because grading presupposes the gold is right, we audited all 12 gold queries against the prompts participants saw before finalizing grades; Section 4 reports the audit and the one gold mismatch it caught.

Behavioral measures such as frustration and reformulation counts are coded from the think-aloud transcripts and cover only the 11 sessions with codable audio, using the symmetric reformulation definition in Section 3.2.2.

4 Results

The results come in two parts. The quantitative half reports outcome statistics, completion time and accuracy, for all 20 participants, with the aggregate mixed-effects estimates in Section 4.1.1; the qualitative half (Section 4.2) reports behavioral coding of the 11 sessions with codable think-aloud audio. The headline contrast pairs a speed gain with an accuracy shortfall: SQL-LLM users took about 212 s less per query on average, roughly a one-third reduction relative to the Snowflake per-participant mean of 629 s (robust on the log scale at $p = 0.03$; $p = 0.04$–0.07 across raw-scale specifications, Section 4.1.1), yet they were not more accurate, answering 46% of queries correctly against 64% for Snowflake ($p = 0.08$ favoring Snowflake, with a confidence interval running from a 39-point deficit to rough parity). The behavioral coding shows what the extra Snowflake time was spent on: schema exploration and syntax construction that the model absorbed in the SQL-LLM arm. The visible work moved.

4.1 Quantitative Analysis

How do the headline numbers hold up under a model that accounts for task heterogeneity, and where do they fall short of significance? We report per-query completion times, then accuracy,

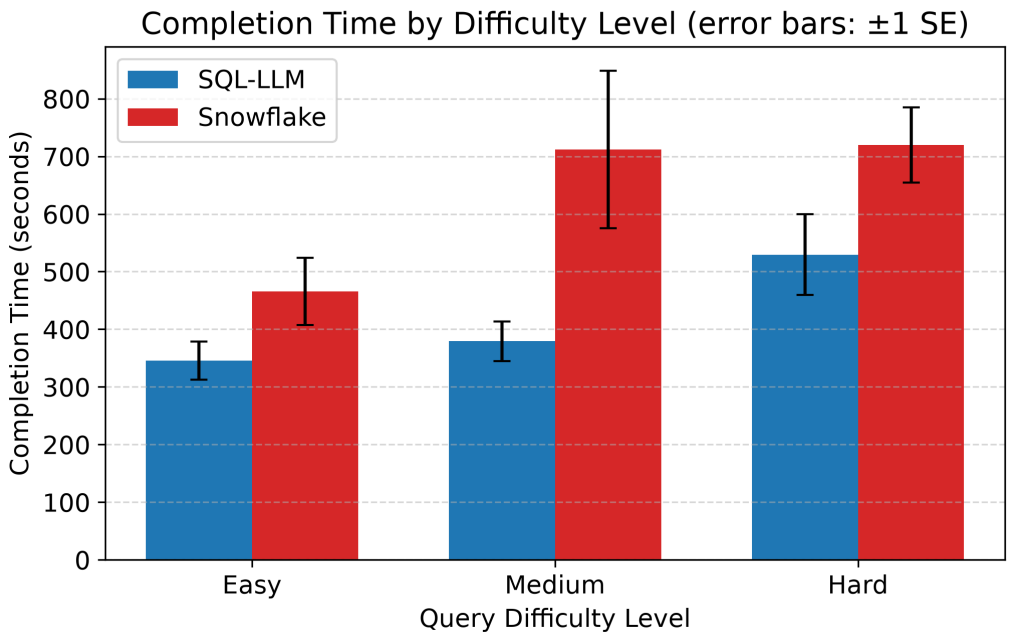

Fig. 2. Mean query completion time (s) by difficulty level (Easy, Medium, Hard) for SQL-LLM and Snowflake. Error bars are ±1 standard error of the mean, computed over the task observations in each difficulty × group cell ($n = 40$ per cell, less any missing observations).

reformulations, and throughput; per-query tests and power are summarized below, with full tables and the presentation-position traces in the appendices.

Per-query completion times (Table 4, Appendix A) are lower for SQL-LLM on 11 of the 12 tasks, with the largest gaps on B-Q1 and B-Q3. Harder tasks take longer while SQL-LLM stays faster at every difficulty level (Figure 2); whether that advantage widens reliably with difficulty is taken up by the linear mixed-effects model in Section 4.1.1.

Speed did not buy accuracy. SQL-LLM users answered 46% of queries correctly (55 of 120) against 64% for Snowflake (77 of 120), each submission graded by executing it against the gold BIRD database and comparing result sets (Section 3.2.4). The 18-point difference favors Snowflake: a participant-level Welch comparison gives $p = 0.08$ (95% confidence interval $[-39, +2]$ percentage points), a logistic regression of correctness on interface with task fixed effects and participant-clustered errors puts the odds of a correct SQL-LLM submission at 0.44 times Snowflake's (95% CI $[0.18, 1.08]$, $p = 0.07$), and a participant-level permutation test agrees ($p = 0.09$). The interval is wide enough that the data cannot rule out a substantial accuracy loss.

Snowflake leads at every difficulty level (Figure 3): Easy (65% versus 42.5%), Medium (77.5% versus 55%), and Hard (50% versus 40%), where both arms fall. Failure modes were similar across interfaces: wrong table family, predicate scope, and join errors. We did not test the per-difficulty gaps for significance.

Auditing the gold answers. Because execution grading presupposes the gold is right, and BIRD golds carry documented noise (15–49% of question–SQL pairs depending on domain [26]), we audited all 12 gold queries against the prompts participants saw. The audit corrected one outright

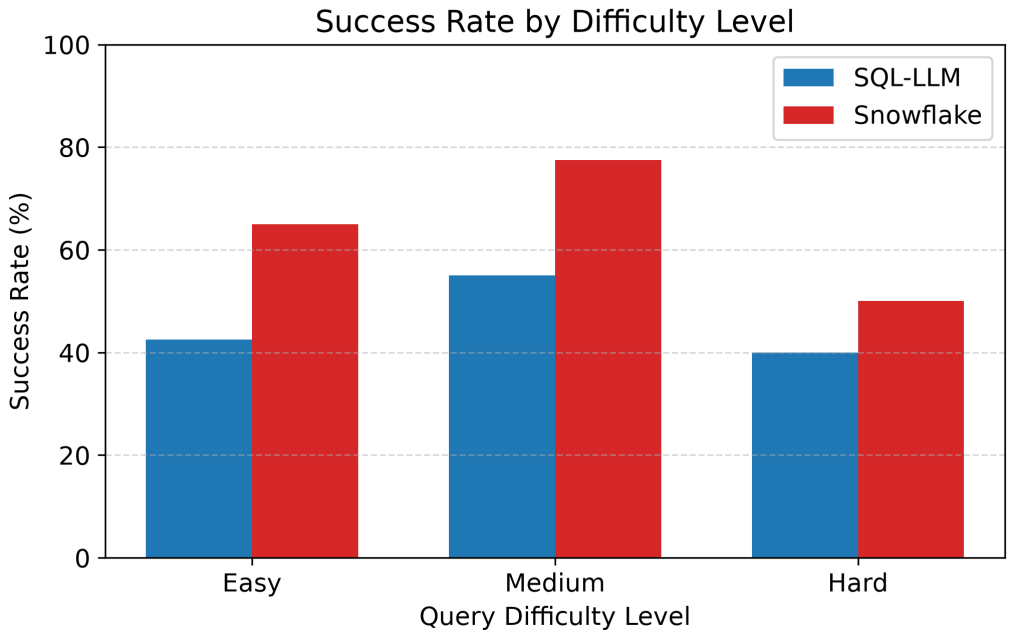

Fig. 3. Queries answered correctly (%) by difficulty, graded by executing each submission against the gold BIRD database and comparing to the gold answer (three independent LLM judges agree with the execution grade on 87.5% of queries).

mismatch (M-Q2, matched to the wrong BIRD question) and flagged one prompt/gold discrepancy (B-Q2, whose hint's literal `author_name LIKE 'Adam'` predicate excludes a name the gold's prefix match includes); Appendix B gives both. The headline grades of 46% vs. 64% already incorporate the corrected M-Q2 gold. Only the B-Q2 discrepancy admits an alternative scenario: its literal reading widens the accuracy gap to 46.7% vs. 68.3% (exact per-participant permutation test, $p = 0.050$). Under either reading, SQL-LLM confers no accuracy advantage. B-Q2 also shows what failure looked like in practice: asked "How many authors are named Adam?", 4 of the 10 SQL-LLM participants submitted the same `LIKE '%Adam%'` predicate, which matches the substring anywhere in the name and so sweeps in authors surnamed Adams (Abigail, Ansel, Douglas among them), returning 39 where the gold counts 21, wrong under both readings of the audited discrepancy. The query executes cleanly and returns a plausible integer; nothing in the result signals that the predicate answered a different question.

Completion time by presentation position. The completion-time traces show no descriptive decline across the task sequence, but the design cannot measure a learning curve: presentation position is confounded with task identity (all participants saw the same 12 tasks in the same order), so the traces, reported in Appendix A, cannot separate practice from task mix. The qualitative record agrees (Section 4.2).

Query reformulations. Reformulation showed no detectable per-minute difference across arms. Over the coded sessions SQL-LLM users averaged $M = 6.5$ reformulations ($SD = 1.5$) and Snowflake users $M = 9.1$ ($SD = 3.1$); the totals are per-session, and normalized per audited minute the two

Metric	SQL-LLM	Snowflake
Descriptive (per-participant means)		
Mean time (s)	418.1	628.7
SD	154.1	295.7
LME fixed effect: Group (ref = Snowflake)		
$\hat{\beta}$ (s)	-212	
SE	106	
z	-2.01	
p	0.044	
LME fixed effects: Difficulty (ref = Easy)		
Medium ($\hat{\beta}$, p)	$+132$ s, $p = 0.059$	
Hard ($\hat{\beta}$, p)	$+216$ s, $p = 0.002$	
Database effects (ref = Books)		
Mondial Geo (p)	0.832	
Legislator (p)	0.728	
Robustness of the Group effect (small-sample and log-scale)		
$t(18)$ reference for the Wald statistic	$p = 0.059$, CI $[-434, +9]$ s	
Welch / permutation, participant means	$p = 0.066$ / $p = 0.045$	
Cluster bootstrap CI, participant means	$[-421, -32]$ s	
Log-time LME: -31% per query	$t(18)$ $p = 0.029$, CI $[-51\%, -4\%]$	
Task fixed effects (raw / log time)	$p = 0.064$ / $p = 0.031$	
Excluding the extreme B-Q1 observation	$\hat{\beta} = -174$ s, $p = 0.042$	

Table 1. LME fixed-effect estimates for completion time. Model: Time ~ Group + Difficulty + Database + (1|Participant); $n = 238$ observations (of the 240 possible from 20 participants × 12 tasks, two were missing from the raw log), fit by REML. Participant random-intercept SD $\approx$ 201 s. The robustness block re-tests the Group contrast with participant-level degrees of freedom (t with 18 df, since treatment is assigned to 20 participants), distribution-appropriate refits, task fixed effects in place of the Difficulty and Database covariates, and outlier exclusion; the reanalysis script (`robustness_reanalysis.py`) is in the replication archive.

arms are indistinguishable ($p = 0.75$). A reformulation follows the symmetric definition in Section 3.2.2. The counts come from the double-coded transcript subset of Section 4.2, which reports the normalization and takes up *when* and *why* reformulations occurred.

4.1.1 Overall Comparison. Because every participant completed all 12 tasks, the data have a repeated-measures structure that a simple two-sample t-test ignores. The primary analysis is therefore a linear mixed-effects model (LME) with participant as a random intercept and Group, Difficulty, and Database as fixed effects, fit by restricted maximum likelihood (REML) to all 238 observations (two missing from the raw log). Table 1 reports the key fixed-effect estimates; Table 2 summarizes results across all measured dimensions.

The LME estimates SQL-LLM users taking 212 s less per query on average ($\hat{\beta} = -212$, SE $= 106$), controlling for query difficulty and database (Table 1). How firmly that estimate holds depends on the inferential method, so we report the range rather than the single most favorable value. The model's asymptotic Wald test gives $p = 0.044$, but the treatment is assigned at the participant level, so the Group effect is identified from 20 clusters and asymptotic z inference is anticonservative at

that count; against a $t(18)$ reference and across Welch, permutation, and cluster-bootstrap refits the effect sits at the significance boundary (robustness block, Table 1). Raw completion times are also right-skewed, with one extreme observation (a 5456 s Snowflake time on B-Q1), so the specification we treat as primary is the log-time refit: the Group effect is -0.38 log-seconds, a 31% reduction per query ($t(18)$ $p = 0.029$, 95% CI -51% to -4%), which survives replacing the Difficulty and Database covariates with task fixed effects and excluding the extreme observation. The effect is robust on the log scale, borderline on the raw scale, and negative in every specification we fit.

Hard tasks took significantly longer than Easy tasks ($\hat{\beta} = +216$ s, $p = 0.002$, 95% CI $[+79, +353]$ s); database choice had no significant effect ($p > 0.7$). The large participant random-intercept SD ($\approx$ 201 s) reflects substantial individual variation in baseline speed, which the mixed model partitions from the interface effect. The Group × Difficulty interaction was not statistically significant ($p >$ 0.1), so the data do not support the gap widening with difficulty, though it is directionally larger for Medium queries.

Throughput. Because speed and accuracy are jointly determined (a user who quickly accepts a plausible but wrong query improves the first and degrades the second at once), we also report the two outcomes as a single rate. Per participant, correct answers per hour of on-task time average 4.6 in the SQL-LLM arm and 4.2 in the Snowflake arm (Welch $p = 0.77$; permutation $p = 0.78$); at the arm level, one correct answer cost roughly 912 s of on-task time under SQL-LLM and 992 s under Snowflake. The 4.6 and 4.2 are means of per-participant rates; the 912 s and 992 s are pooled arm-level figures, so the two aggregations do not invert into each other. On throughput the arms are near parity: the interface changed how time was spent far more than how much correct output came back per unit of time. Within both arms, submissions that turned out correct took *less* time than those that turned out wrong (SQL-LLM 374 vs. 456 s; Snowflake 613 vs. 662 s), consistent with errors concentrating on tasks participants found hard rather than with fast, careless acceptance, though this conditional comparison is descriptive.

Table 2 sets the dimensions side by side: faster completion without an accuracy advantage, near-parity throughput, and behavioral counts with no detectable per-minute difference in any category (normalization in Section 4.2). We read this as the interfaces relocating the observable work: SQL-LLM users left schema discovery to the model and verified outputs, while Snowflake users spent their time on manual schema exploration and syntax construction.

4.1.2 Per-Query Tests and Power. Per-query independent two-sample t-tests, reported with effect sizes and confidence intervals in Appendix C, check whether the aggregate Group effect rests on one or two queries: point estimates favor SQL-LLM on 11 of 12 tasks, but at $n = 10$ per arm only B-Q3's interval excludes zero, so these tests are exploratory and carry no inferential weight of their own. The formal power analysis (also Appendix C) sizes the design at medium-to-large effects: a two-sample test on participant means has roughly 47% power at the observed effect, which is why the corrected participant-level inference in Table 1 hovers near the significance boundary on the raw scale.

This power profile shapes how the null findings should be read. The behavioral *rates* (events per audited session minute; Table 2) showed no detectable between-arm differences at the $n = 11$ behavioral subset. We frame these as "no detectable difference at this N" rather than as evidence of equivalence between interfaces: distinguishing "the interfaces shift the same total effort" from "the interfaces produce subtly different counts we cannot resolve here" would take a follow-up sized for behavioral counts, on the order of $n = 30$ to 40 per arm given the observed variances.

Metric	SQL-LLM	Snowflake	Difference
Quantitative outcomes: all 20 participants ($n = 10$ per arm)			
Completion time	418.1 (154.1) s	628.7 (295.7) s	−31% log scale ($p = 0.03$); −212 s raw ($p = 0.04$–0.07)
Accuracy	46%	64%	−18 pts, $p = 0.08$, CI $[-39, +2]$ pts
Correct answers per hour	4.6	4.2	Near parity ($p = 0.77$)
Behavioral counts: 11 coded transcripts (4 SQL-LLM, 7 Snowflake), double-coded, $\alpha = 0.90$			
Query reformulations	6.5 (1.5)	9.1 (3.1)	$p = 0.75$; $\alpha = 0.66$, tentative
Frustration episodes	6.4 (10.3)	4.5 (5.4)	$p = 0.82$; source differed
Schema checks	14.6 (4.8)	25.7 (9.7)	$p = 0.32$; raw gap tracks session length
Hesitation episodes	3.8 (1.8)	7.4 (3.2)	$p = 0.63$; $\alpha = 0.20$, read directionally
Expressions of confusion	10.0 (9.9)	16.2 (7.2)	$p = 0.36$
Positive confirmations	9.5 (0.4)	13.9 (1.4)	$p = 0.89$
User strategy	Output verification	Schema-first build	Observable work relocated (coded subset)

Table 2. Comparison of SQL-LLM vs. Snowflake across measured dimensions, with the sample behind each block stated in its divider row. Numeric cells are mean (SD). Behavioral cells are per-session event counts from the $n = 11$ transcripts, independently double-coded by two analysts (overall Krippendorff's $\alpha = 0.90$). Raw counts scale with session length; the per-minute normalization is in Section 4.2. For each behavioral row, the Difference column reports the per-minute between-arm comparison; no category shows a detectable difference (all $p \geq 0.3$).

4.2 Qualitative Analysis

The aggregate outcomes told us that SQL-LLM users were faster, not what changed in their work. Think-aloud transcripts and screen recordings, available for 11 of 20 participants (4 SQL-LLM, 7 Snowflake), let us ask where the effort went. We read each session along three dimensions (confidence in query output, schema discovery strategy, and error recovery) and counted events in the six coded categories below. Under SQL-LLM, users left schema navigation to the model and spent their effort verifying the generated SQL; under Snowflake, users spent that same effort earlier, in manual schema exploration and syntax construction. The coded record shows where the observable work went, not that total effort was equal on each side (per-minute rates below).

Coding methodology and inter-coder reliability. Two analysts independently coded all 11 transcripts across six categories: frustration episodes, schema-check events, hesitation episodes, reformulations, expressions of confusion, and positive confirmations. Both were hired for the task and blind to the study's hypotheses, to each other's coding, and to any prior counts. Transcripts were relabeled in random order, though the interaction modality (writing SQL vs. reviewing generated SQL) is evident from the transcript itself. Coding was conservative and event-based: each continuous stretch of one activity counted as a single event, ambiguous utterances were not counted, and

each utterance was assigned its single best-fit category. Overall inter-coder agreement was high (Krippendorff's $\alpha = 0.90$; interval metric, cluster-bootstrap 95% CI $[0.81, 0.93]$), with per-category α ranging from 0.99 (frustration) and 0.94 (confusion) down to 0.20 (hesitation); schema-check, the category our account leans on, reached $\alpha = 0.78$. We report the two-coder mean counts, and for the two lower-agreement categories, hesitation ($\alpha = 0.20$) and, more mildly, reformulation ($\alpha = 0.66$, at the tentative threshold), rely on direction rather than exact magnitude. Each coded participant's strategy class is recorded in Table 3; with 11 coded participants we do not relate strategy class to task outcomes. Two transcripts (P2, P16) had sparse audio and yielded conservative lower bounds. Quotations are lightly edited to correct automatic-transcription artifacts, and participants use the neutral codes from Section 3.2.1. The full rubric, both coders' event logs, the reliability script, per-arm group means, per-minute rates, and representative events are in the replication archive (`human_irr.py`, `human_coding_*.csv`, `coding-kit/CODEBOOK.md`, `coding_examples.md`).

As a further check that the coding scheme is codable rather than idiosyncratic, we also ran three independent LLM coders over the same transcripts; their annotations and their agreement with an earlier single-coder pass are reported in Appendix D.

Because the behavioral counts are per-session totals, they scale with session length, and the arms' sessions differed (audited means 81 vs. 129 minutes). Normalizing each count to events per audited minute removes that confound, and on this basis no category differs detectably between arms (all $p \geq 0.3$; the result is unchanged when restricted to the seven sessions with reliable recording clocks). The raw counts therefore locate where activity occurred, not how much more of it one arm produced. Positive confirmations make the point: an earlier single-coder pass had recorded more under SQL-LLM, but the two independent coders and all three LLM coders agree the raw total is higher for Snowflake, and the per-minute rate indistinguishable, because Snowflake users, constructing queries step by step, voice a short confirmation as each fragment runs ("okay, looks good", "there we go"), while SQL-LLM users receive an answer in one shot and confirm once.

4.2.1 Qualitative Metrics. The counts live in Table 2; we read them as a map of where each arm's observable activity fell. This subsection leads with schema interaction, the category on which the account turns and where two-coder agreement is solid ($\alpha = 0.78$).

The finding our account rests on is qualitative rather than a gap in counts: *when* and *why* schema interaction occurred. Snowflake users issued `INFORMATION_SCHEMA` queries as their primary discovery mechanism, treating schema exploration as a prerequisite before any query could be attempted. SQL-LLM users verified specific column names *after* receiving model output, checking that generated SQL aligned with the schema rather than building queries from scratch. Schema-check counts run higher for Snowflake in raw totals, but the arms' per-minute rates are indistinguishable (Table 2), consistent with the model resolving the initial schema automatically while the analyst's own schema work shifts downstream. The schema work did not disappear; it moved from construction to verification.

Frustration showed no per-minute difference (Table 2); only its *source* differed, arising under SQL-LLM from semantic mismatches between the user's intent and the model's generated query and from interface friction such as lag and difficulty interrupting generation, and under Snowflake from schema navigation failures and SQL syntax errors.

4.2.2 Coded Behavioral Patterns. The transcripts show what the shift felt like. On confidence, SQL-LLM users displayed early trust in the model's output, treating it as a first pass to verify rather than rewrite, while Snowflake users showed more hesitancy and trial-and-error:

> *"Okay yes, the query given by the AI is correct. [...] I will just verify it quickly."* – P1, SQL-LLM

Participant	System	Schema Strategy
P1	SQL-LLM	Light Verif.
P2	SQL-LLM	Minimal
P7	SQL-LLM	Minimal
P9	SQL-LLM	Moderate
P11	Snowflake	Manual Expl.
P12	Snowflake	Info Schema
P13	Snowflake	Manual Expl.
P14	Snowflake	Browse + Trial
P15	Snowflake	Manual Expl.
P16	Snowflake	Trial-and-Error
P19	Snowflake	Join Validation

Table 3. Schema-discovery strategy class of each coded participant, identified from the think-aloud protocols: whether and how the participant explored the schema before or instead of querying (minimal consultation, verification of specific columns, systematic manual exploration, `INFORMATION_SCHEMA` queries, browse-and-trial, or join-focused validation). Affective dimensions (frustration, confidence) are deliberately not tabulated per participant: their event counts show no detectable between-arm difference (Table 2) and we have no validated rubric for ordinal per-person labels.

On schema discovery, Snowflake users relied on exploratory strategies, browsing tables, issuing information-schema queries, and trialing joins, whereas SQL-LLM users bypassed exploration and verified joins and columns after the fact:

> *"What I'm going to do is take a look at these tables. [...] We can say select table name from information schema."* – P12, Snowflake

On error recovery the same split held: Snowflake users showed frustration navigating ambiguous joins or failing queries, while SQL-LLM users debugged at the level of logical alignment rather than syntax.

Across the three themes the difference was strategic, not affective: both groups sustained attention and adjusted iteratively, and the interface set *where* the work happened, not whether it happened. Table 3 records the per-participant coding behind this pattern.

The transcripts add the spread the group means hide: frustration was mixed within the SQL-LLM group, concentrated in participants working complex multi-join queries or hitting interface lag, so the low-effort verification pattern held for most SQL-LLM users but not all.

The design implication is specific: SQL-LLM reduces the upfront burden of schema navigation (shrinking and shifting it downstream, since users still verified generated joins and column choices) and adds the burden of verifying and correcting generated queries, most sharply for complex joins and multi-step reasoning. Schema visibility, a readable generated query, and clear error recovery are the levers an organization can actually pull; interactive subquery previews, highlighted faulty clauses, or partial execution could align better with how users repair queries and reduce the effort of correction. We return to these levers as adoption pathways in Section 5.2.

4.2.3 Micro-level Repair Strategies. The shift also showed up at the level of single clauses, in how users repaired a query once it went wrong. Snowflake users attempted localized adjustments, often with difficulty: P19 caught a missing grouping only after running the query, and P12 returned to `INFORMATION_SCHEMA` repeatedly between attempts, re-checking table and column names with little system-level support for contextual feedback. SQL-LLM users instead verified and adjusted

at the semantic level, treating a system error as a prompt for incremental refinement rather than a reason to start over, isolating and checking one component at a time, as when P1 commented out part of a generated query to test the inner query on its own. Snowflake users were likelier to locate the fault in their foundations and rebuild rather than patch:

> *"Maybe I was just using the wrong table and I don't understand how the historical table works."* – P11, Snowflake

The contrast is between trusting and refining a generated query and rebuilding one from scratch. SQL-LLM thus supports low-friction, micro-level revision, whereas traditional SQL interfaces demand more complete syntactic correctness upfront. These repair episodes are selected from 11 transcripts, not a census of repair behavior. One pattern appears only in the SQL-LLM transcripts: how users adapted their phrasing across tasks.

4.2.4 Emergent Strategies in the SQL-LLM Arm. Only four SQL-LLM participants produced codable think-aloud audio, so everything here is an observation from a small set, reported as raw counts rather than rates. Three strategies surfaced, unevenly: prompt-structure reuse, where a participant adapted a structure that had worked on an earlier task rather than phrasing each from scratch (one participant); deliberate probing of the system's flexibility by varying prompt specificity (one participant); and the refine-not-rebuild move from the previous subsection (two participants). We did not see the learning curve one might expect across successive tasks: the participant who reused prompt structures adopted the routine on an early task and kept it, so we read the variation as differences in habitual style that participants brought with them rather than a trajectory the session produced. What the interface did was let some users exercise a strategy they already had, which argues for aligning error feedback and interaction flow with the reuse, probing, and refinement we observed rather than an assumed learning curve. Whether users without SQL training would develop or benefit from the same strategies is taken up in the discussion that follows.

5 Discussion

In our study, SQL-LLM cut about 212 s per query (31% on the log scale) relative to the manual-SQL arm, a large speed effect. The nearest comparable study saw none: Ning et al. [17] tested interactive error-handling interfaces against a no-support baseline and found no time difference. On accuracy the two studies agree: our SQL-LLM users were no more correct than their Snowflake counterparts, and our point estimate runs 18 points lower, a possible loss that $N = 20$ can neither confirm nor exclude. One conjecture is that speed was the model's problem and the models have since matured, while verification is the user's problem, and our data give no sign it has moved; two studies differing in task set, interface, and population cannot test this. The implications below turn on verification as the user's problem.

5.1 Practical Implications

Our central finding is that SQL-LLM reallocated the work of querying rather than removing it: in the coded sessions, observable work moved from query construction toward output verification. Behavioral coding showed no per-minute difference in any coded category (Section 4.2); the faster completion time brought no gain in accuracy, and correct answers per hour stayed near parity. The same cost appears in AI-assisted programming, where verifying suggestions occupies a measurable share of programmers' time [15]. The interface made querying faster, not more correct. For users who already have working SQL knowledge it redistributes the analyst's time: faster turnaround per query, with a possible accuracy cost the present sample cannot pin down. That reframes what counts as a benefit for organizations evaluating NLIDB adoption: an argument built around "less

work" [2] misreads the tool, and an adoption case must weigh the speed gain against an accuracy risk our data flag but do not settle.

The 46% success rate needs context, because on its face it reads as an indictment. Trained users making errors is the expected condition, not an anomaly: the literature documents high rates of syntactically valid but semantically wrong queries among trained SQL users [1, 21], and even BIRD's own human baseline among data engineers and database students is 92.96% rather than 100% [12]. That baseline covers different tasks, tools, and conditions and does not explain rates as low as 46% and 64%; we cite it only to establish that wrong-but-executable queries are the working condition of SQL, whoever writes it. What matters for adoption is who catches them, and our data leave open whether anyone does: would a user catch a query that is plausible but quietly wrong [18], such as the B-Q2 substring query in Section 4? Positive confirmations, the one behavioral signal that might have answered it, ran at indistinguishable per-minute rates across arms (Section 4.2). Until confidence in generated SQL is shown to track correctness, the prudent default is to keep the generated query where a reader can check it, which Section 5.2 states as a design hypothesis.

5.2 Readiness and Adoption Pathways

An organization that hands its analysts an NLIDB shifts a skill requirement rather than eliminating it: someone must read the generated SQL and confirm it means what the question asked. That single dependency governs who benefits, what has to change, and where deployment should start [25].

The analyst's job shifts from authoring queries to reviewing them. In the think-aloud transcripts, the SQL-LLM participants who verified successfully read the generated SQL against their intent, and the effort saved on syntax reappeared as verification work. Adopting SQL-LLM is an investment in review skill, not a way to retire SQL skill: staff still need to read a join, spot a wrong grouping, and recognize a result that is plausible but wrong. Training that sells the tool as a route around SQL literacy removes the competency the speed-up depends on.

Schema governance is the substrate both the model and the reviewer lean on. The model absorbed schema discovery because our schemas were small, clean, and clearly named, and users verified column names against them afterward. Beyond the schemas we tested, an enterprise schema with hundreds of ambiguously named tables should make both halves of that loop harder: the model has more room to guess wrong, and the analyst has more to check, though our data cannot quantify either effect; on the model's half there is direct evidence that schema-identifier naturalness correlates with LLM Text-to-SQL performance [13]. Documented, consistently named schemas keep the generated SQL auditable, and an organization with poor schema hygiene should expect the verification burden to grow before the construction saving arrives.

Keeping the generated SQL visible is, on our evidence, a prudent default rather than a demonstrated safeguard, and we state it as a design hypothesis. Expressed confidence in the transcripts followed successful checks of the visible query, but visibility was constant within the SQL-LLM arm, we did not measure whether those checks caught errors, and SQL-LLM accuracy was if anything lower. An interface that hides the SQL to look simpler would close the only verification channel our SQL-literate participants used; testing whether visibility actually protects correctness, for instance with seeded plausible-but-wrong queries under visible and hidden SQL, is the experiment the hypothesis needs.

Deployment should be sequenced by who can verify. The benefit concentrates where users can read the output: analyst and data-engineering teams gain a speed-up on routine querying while retaining the ability to review generated SQL. Extending the same interface to staff who cannot read SQL is a different proposition, because the verification step our participants relied on is unavailable to them, and our data cannot say how that trade-off resolves. A staged rollout that reaches

SQL-literate analysts first, as an accelerator rather than a replacement, matches the tool to the users the evidence covers.

5.3 Limitations

The interpretation is bounded twice: internally, by what a 20-participant design can statistically confirm, and externally, by the conditions we studied. Those conditions are also the axes along which the findings are most likely to vary elsewhere: team SQL literacy, schema design, task framing, the system we deployed, and the single-session snapshot we measured.

- **Statistical power and sample size**: The between-subjects design splits 20 participants 10/10, powered to detect only medium-to-large effects. The completion-time result clears this bar on the log scale ($p = 0.03$) and hovers at the boundary on the raw scale ($p = 0.04$–0.07 across specifications; Table 1). Several headline comparisons do not. The accuracy gap (46% vs. 64%, $p = 0.08$, CI $[-39, +2]$ points) shows no detectable gain for SQL-LLM, and an interval this wide cannot rule out a substantial loss, so the direction favoring Snowflake is plausible but not established. The coded behaviors show no between-arm difference in per-minute rate and support no directional claim. Where we report no difference, read it as no detectable difference at this N. Establishing that accuracy is *preserved* would require a noninferiority design with a prespecified margin and a larger sample.
- **Participant background and diversity**: All participants had at least basic SQL literacy; none were complete novices, and all came from data analysis backgrounds. This ensured fair comparisons between systems, but for users with little or no SQL experience, NLIDBs may yield greater benefits or different failure modes. Claims about "non-technical user" benefit elsewhere in the Text-to-SQL literature are not supported by our data. The SQL-education literature sharpens the concern: novices produce syntactically valid but wrong-result queries at high rates, with the same logical errors recurring across students [21], so the verification channel our SQL-literate participants relied on is exactly the channel a novice population would lack.
- **Task hints removed natural ambiguity**: The textual hints that disambiguated each task (Section 3.2.2) reduced the natural ambiguity real users would encounter, and their net effect on the between-arm comparison is not identified. Results should be read as usability under *disambiguated* task framing; performance under hint-free, naturalistic phrasing remains an open question.
- **Flat compensation**: Participants received a fixed $100 with no accuracy-contingent component, so submitting a plausible but unverified answer carried no cost. This satisficing risk applies to both arms but is not symmetric: accepting a generated query and moving on is easier than abandoning a half-written one, so the accuracy comparison may partly reflect incentives rather than interfaces, and generalization to settings where analysts bear the cost of wrong answers is limited.
- **Behavioral coding reliability**: All 11 transcripts were independently double-coded by two analysts (overall Krippendorff's $\alpha = 0.90$), but agreement is uneven across categories: high for frustration, confusion, schema-check, and positive confirmations, low for hesitation ($\alpha = 0.20$), which we read only directionally, and tentative for reformulation ($\alpha = 0.66$). The coded subset is also a post-assignment selection (11 of 20, unevenly 4/7 across arms), and per-session event counts scale with session length, so we compare per-minute rates rather than raw totals. These bound how much weight the behavioral counts can bear.

- **Transfer to large, complex, real-world schemas**: Tasks covered Easy to Hard difficulty levels but ran on small- to medium-scale synthetic schemas (e.g., Books, Mondial Geo, Legislator). Enterprise databases with hundreds of tables, complex joins, and security constraints may present challenges these tasks did not surface.
- **A paradigm comparison, not a system benchmark**: We compared one deployed NLIDB against one conventional SQL editor. The findings speak to the NLIDB-versus-manual-SQL paradigm for SQL-literate users, not to how SQL-LLM ranks against other NLIDBs. A different backing model, prompting strategy, or interface could change the size of the speed effect and the shape of the verification burden, so the reallocation account should be read as a property of exposing an editable generated query, not a claim about this particular system.
- **Longitudinal effects**: The study measured immediate usability, a snapshot that cannot see how learning, trust, and workflow fit evolve with use [5].

6 Future Work

The study leaves one question above the rest: when the model produces a quietly wrong query, do users catch it? Section 5 explains why our data cannot answer it. Most directions below attack that gap; one removes the framing aid our task hints supplied, and one re-runs the comparison on tasks no public corpus contains.

- **Verification accuracy under seeded errors**: Inject syntactically valid but wrong-result queries, and measure how often each interface's users catch the error before accepting it. This turns a calibration question our data cannot settle into a measurable catch rate and false-confidence rate.
- **A finer rubric for confirmation behavior**: Our coding (Section 4.2) does not separate final-result confirmations from the running step-confirmations that accumulate when a query is built by hand, so raw counts stay confounded with construction style. A rubric distinguishing the two would let confirmation behavior be compared on equal footing rather than through a count that mixes the two.
- **Verification without a readable query**: Our participants could read and check a generated query before accepting it. Study a population that cannot read it: with the verification channel closed, does confidence run ahead of correctness, and does anything replace the check? This tests whether the reallocation account holds when its verifying step is unavailable.
- **Hint-free, naturalistic phrasing**: Re-run the task set without the textual hints that disambiguated phrases such as "most orders," the framing aid Section 5.3 flags as a limitation. Removing them restores the intent ambiguity real users face and isolates how much of the construction saving depends on disambiguated framing.
- **Privately authored tasks**: BIRD and Spider are public corpora the backing model may have seen in training (Section 3.1.4). A confirmatory study should use privately authored tasks over schemas absent from public corpora, logging the model snapshot, prompts, and validator behavior. This would settle whether the headline comparison survives on tasks the model cannot have seen.
- **Instrumented interaction logging**: Capture the initial generated SQL, every edit, regeneration, chat turn, and validator rejection. These traces would measure the path from generated to submitted query directly (initial model accuracy, how often users repair a wrong query or break a right one), turning a mechanism this paper infers from transcripts into a logged, quantifiable process.

7 Conclusion

A natural-language interface does not remove the work of querying a database; on our evidence, it reallocates that work, moving its visible part from constructing SQL to verifying it. SQL-LLM, the GPT-4o-backed NLIDB we studied, cut completion time by about a third relative to Snowflake, the manual-SQL arm (31% on the log scale, $p = 0.03$; about 212 s per query, borderline on the raw scale; Section 4.1.1). But the saved time did not turn into more correct answers: graded against the BIRD gold answers, SQL-LLM users were correct on 46% of queries versus 64% for Snowflake, an 18-point difference favoring Snowflake ($p = 0.08$ at $N = 20$), with correct answers per hour near parity. The model absorbed schema discovery and syntax construction, and in the coded sessions that saved effort resurfaced as the work of verifying the SQL it returned. What our data support is reallocation, of visible work and of time per query, not a demonstrated reduction in work: the gain is real, but neither effortless nor universal, and it carries a possible accuracy cost that a larger study must price.

For organizations evaluating NLIDB deployment, the lesson follows from who was in the room. Our SQL-literate participants could read and verify the model's output; the trade-off shifts for users without that background, since the checking channel our participants relied on is the one they cannot use.

That channel pays off only if trust in it is calibrated [8]: analysts believing correct output and doubting incorrect output. For the SQL-literate users our evidence covers, the open question a deployment must answer is this: does confidence in generated SQL track correctness, and do users catch the quietly wrong query?

Ethics Statement

This study involved human participants recruited via Upwork. All participants provided informed consent, and their personal information was anonymized. The study protocol was reviewed and approved by the Institutional Review Board of New York University (IRB-FY2023-7595).

Conflicts of Interest

SQL-LLM is a deployed system that we evaluate under a pseudonym, a condition of the anonymity promised to its provider. The authors have no financial interest in, and no professional affiliation with, the provider of SQL-LLM or with Snowflake. Snowflake was chosen as a representative SQL platform, which, at the time of the experiment, did not include any Text-to-SQL capabilities in its Web UI; the comparison evaluates interface paradigms, not vendors.

Data Availability

The analysis code and study data (think-aloud transcripts, submitted queries, timing logs, behavioral codings, the execution-grading pipeline, and the robustness-reanalysis script behind Table 1) are available as a replication package on Zenodo: https://doi.org/10.5281/zenodo.21433589. Participants appear throughout the package under the same neutral codes (P1–P20) used in this paper; recordings and raw files that could identify individuals are excluded. Where an artifact does not exist, we say so rather than omit silently: the exact GPT-4o snapshot identifier is not recoverable, and validator rejections, fine-grained interaction traces, and the open-ended debrief responses were not captured or are not analyzable (Sections 3.1.4 and 3.2.3).

A Supplementary Quantitative Material

Table 4 reports per-query completion times; the aggregate model is in Section 4.1.1.

Query	Difficulty	SQL-LLM (s)	Snowflake (s)
B-Q1	Medium	402.9 ± 255.2	1103.9 ± 1553.9
B-Q2	Easy	275.6 ± 184.3	269.8 ± 62.6
B-Q3	Medium	474.7 ± 268.0	803.9 ± 375.5
B-Q4	Hard	364.8 ± 130.1	634.7 ± 385.9
M-Q1	Medium	395.9 ± 139.8	615.1 ± 341.6
M-Q2	Hard	291.5 ± 107.8	410.3 ± 247.8
M-Q3	Hard	893.1 ± 657.2	978.6 ± 385.6
M-Q4	Easy	291.6 ± 290.3	484.3 ± 365.6
L-Q1	Easy	386.4 ± 136.2	442.1 ± 168.9
L-Q2	Medium	242.8 ± 116.8	313.7 ± 218.7
L-Q3	Hard	568.3 ± 391.5	824.8 ± 398.9
L-Q4	Easy	429.8 ± 183.2	664.9 ± 577.5

Table 4. Per-query completion time (mean ± SD, in seconds) for SQL-LLM and Snowflake, with each query's difficulty level.

Completion time by presentation position. The data neither establish nor rule out a within-session practice effect, because task position is confounded with task identity: every participant saw the same 12 tasks in the same listed order, so the mean time at each position reflects that position's specific task (its difficulty and database) as much as any practice. What we can report is the descriptive pattern by *presentation position* (Figure 4). A linear fit of mean completion time against position is essentially flat for SQL-LLM (about +6 s per task, $r = 0.12$) and weakly negative for Snowflake (about −14 s per task, $r = -0.20$), with neither slope distinguishable from zero and no positional trend in accuracy. Measuring learning would require randomized or counterbalanced task order.

The apparent early drop in the Snowflake curve is driven largely by a slow, high-variance first task rather than by steady speedup, and the SQL-LLM curve does not compress over the 12 positions; given the position–task confound, the flat traces cannot rule out learning that the task mix masks. The qualitative record (Section 4.2) is consistent: we saw no shift from instance-specific to more abstract prompting as a session progressed.

B Gold-Answer Audit

Grading against BIRD gold answers presupposes the gold is right, and audits of BIRD have found noise in 15–49% of question–SQL pairs depending on domain [26]. We therefore audited all 12 gold queries against the task prompts participants actually saw, cross-checked against the think-aloud recordings, and caught one outright mismatch. Our grading pipeline had matched M-Q2 to the BIRD question "Which country has the lowest inflation rate?", but participants were shown a different BIRD question over the same tables ("What is the name of the country whose citizens have the lowest purchasing power?"), whose gold answer is the country with the *highest* inflation. Because participants in both arms overwhelmingly answered the question they were asked, we re-matched the gold and re-graded: the correction turned nine wrongly-failed submissions (two SQL-LLM, seven Snowflake) into passes and one previously-passing submission, which had answered the lowest-inflation reading, into a fail. The mismatch also drives the LLM-judge cross-check reported in the Measures: the largest share of the judge-execution disagreements, 10 of 30, falls on M-Q2, where the judges had been supplied the same mismatched gold this audit corrects. The remaining disagreements scatter across the other tasks.

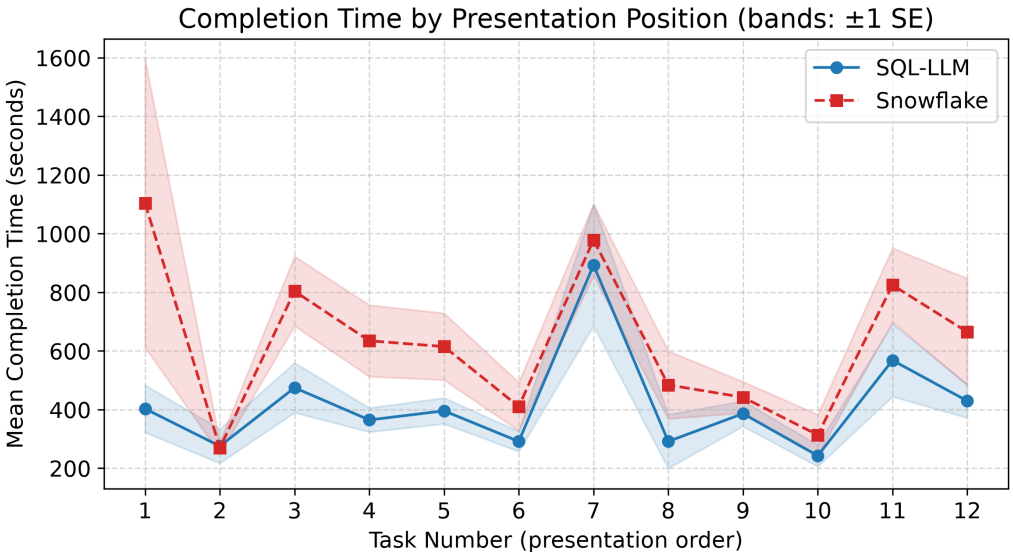

Fig. 4. Mean completion time (s) by presentation position across the 12-task sequence, by group, with ±1 standard-error bands. Position is confounded with task identity (all participants saw the same tasks in the same listed order), so these traces describe the session profile and are not a learning curve.

Ten of the remaining eleven golds pass the audit: the two `LIMIT 1` golds have unique optima, the most-prevalent-religion gold gives the same answer whether or not percentages are population-weighted, and the two with idiosyncratic output shapes (a `NULL` district row; a `SELECT *`) are graded by accepting any submission that returns the same underlying answer set. The remaining gold, B-Q2 ("How many authors are named Adam?"), is a prompt/gold mismatch rather than mere ambiguity: the hint participants saw gives the literal predicate `author_name LIKE 'Adam'`, which in SQL (absent wildcards) matches the exact string and so excludes "Adam-Troy Castro," while the gold uses a prefix match that includes it. A participant who implemented the hint exactly as written is graded incorrect under the primary grading. Six participants answered under the hint's literal reading (one SQL-LLM, five Snowflake); re-grading those six as correct widens the gap to 46.7% vs. 68.3% and brings it to the significance boundary (exact per-participant permutation test, $p = 0.050$). Under either reading SQL-LLM confers no accuracy advantage.

C Per-Query Tests and Statistical Power

Table 5 reports the per-query two-sample t-tests with effect sizes and confidence intervals. These tests are exploratory: at $n = 10$ per arm each is underpowered, they are not adjusted for multiplicity, and no individual task carries inferential weight. Their role is descriptive, showing that the aggregate Group effect is consistent in direction across tasks (11 of 12 point estimates favor SQL-LLM) rather than resting on one or two queries. The inferential analysis is the mixed-effects model with its small-sample robustness block (Table 1).

Statistical power. With 10 participants per arm in a between-subjects design, the study is sized to detect medium-to-large interface effects. The observed effect on per-participant mean completion times is large: the raw between-arm difference is a 211 s reduction, matching the model's adjusted

Query	t-statistic	p-value	Cohen's d (95% CI)
B-Q1	−1.41	0.191	−0.63 [−1.52, 0.28]
B-Q2	0.09	0.927	+0.04 [−0.84, 0.92]
B-Q3	−2.26	0.038	−1.01 [−1.93, −0.06]
B-Q4	−2.10	0.060	−0.94 [−1.85, 0.00]
M-Q1	−1.80	0.102	−0.86 [−1.79, 0.10]
M-Q2	−1.33	0.211	−0.64 [−1.55, 0.30]
M-Q3	−0.36	0.728	−0.16 [−1.04, 0.72]
M-Q4	−1.31	0.209	−0.58 [−1.47, 0.32]
L-Q1	−0.81	0.428	−0.36 [−1.24, 0.53]
L-Q2	−0.90	0.381	−0.40 [−1.29, 0.49]
L-Q3	−1.45	0.164	−0.65 [−1.54, 0.26]
L-Q4	−1.23	0.246	−0.55 [−1.44, 0.35]

Table 5. Exploratory per-query two-sample t-tests on completion time, with between-subjects Cohen's d and 95% confidence intervals from the noncentral-t distribution, the exact small-sample interval for a standardized difference (negative d favors SQL-LLM); the per-query test and effect-size tables are included in the replication archive. The tests are unadjusted for multiplicity and individually underpowered ($n = 10$ per arm); read the effect estimates and intervals, not isolated p-values.

Group estimate of $\hat{\beta} = -212$ s to within rounding, or Cohen's $d \approx 0.89$. Computed at that observed effect size ($d \approx 0.89$, per-participant means, 10 per arm, two-sided $\alpha = 0.05$), a two-sample test achieves only about 47% power, and reaching 80% power at the same observed d would require roughly 21 participants per arm; a conventional medium effect ($d = 0.5$) would require about 64 per arm. The same low power shapes how to read the per-query tests in Table 5: at $n = 10$ per arm, a per-query effect near the aggregate would be detected less than half the time, so the near-boundary per-query p-values are consistent with limited power and, on their own, neither establish nor rule out an effect at the task level.

A caveat on what the repeated measures buy: each participant contributes 12 observations, which sharpens the estimate of within-participant variation and lets the model partition individual baseline speed ($\sigma_{\text{participant}} \approx 201$ s) from the Group difference, but the repeated tasks do not create additional between-participant information. The Group effect is identified from the 20 participants to whom treatment was assigned, which is why the primary inference in Table 1 uses participant-level degrees of freedom and permutation checks rather than treating the 238 observations as independent evidence about the interface contrast.

D LLM-Coder Cross-Check

As an additional check that the coding scheme is not idiosyncratic, we also ran three independent LLM coders (OpenAI GPT-5.5, Google Gemini 3 Pro Preview, and Anthropic Claude Opus 4.7) at `temperature=0` where supported, each asked to return a list of events (category, 5–25-word verbatim quote, one-sentence justification) rather than aggregate counts. Counts are derived as the size of each event list. Per-utterance annotations from all three coders are included in the replication archive (`llm_annotations.jsonl`) so that any specific count can be audited.

Aggregate agreement between the LLM coders and an earlier single-coder pass (Pearson correlation over the 11 sessions) was strongest for the affective and exploratory categories, where frustration reached $r = 0.82$ and schema-check $r = 0.84$. The LLMs' absolute counts diverged unevenly rather than uniformly inflating (they over-counted most categories but under-counted

hesitation), consistent with their varying in how strictly they applied the one-event-per-activity rule. These correlations predate the two-analyst coding and are not our reliability estimate; the human α reported in Section 4.2 is that, and we cite the LLM coders only as a coarse check that the scheme is codable. The single-coder-versus-LLM correlations and event totals are in the replication archive (`coding_agreement.csv`).

References

[1] Stefan Brass and Christian Goldberg. 2006. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software* 79, 5 (2006), 630–644. doi:10.1016/j.jss.2005.06.028

[2] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. doi:10.2307/249008

[3] Izzeddin Gür, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue based Structured Query Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1339–1349. doi:10.18653/v1/P18-1124

[4] S. G. Hart and L. E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Advances in Psychology*. Vol. 52. Elsevier, 139–183.

[5] Kevin Anthony Hoff and Masooda Bashir. 2015. Trust in Automation: Integrating Empirical Evidence on Factors That Influence Trust. *Human Factors* 57, 3 (2015), 407–434. doi:10.1177/0018720814547570

[6] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. 2007. Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (Beijing, China) *(SIGMOD '07)*. Association for Computing Machinery, New York, NY, USA, 13–24. doi:10.1145/1247480.1247483

[7] Denis Kochedykov, Fenglin Yin, and Sreevidya Khatravath. 2023. Conversing with databases: Practical Natural Language Querying. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, Mingxuan Wang and Imed Zitouni (Eds.). Association for Computational Linguistics, Singapore, 372–379. doi:10.18653/v1/2023.emnlp-industry.36

[8] John D. Lee and Katrina A. See. 2004. Trust in Automation: Designing for Appropriate Reliance. *Human Factors* 46, 1 (2004), 50–80. doi:10.1518/hfes.46.1.50_30392

[9] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, H. V. Jagadish, and Mirek Riedewald. 2020. QueryVis: Logic-based diagrams help users understand complicated SQL queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2303–2318. doi:10.1145/3318464.3389767

[10] Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84. doi:10.14778/2735461.2735468

[11] Fei Li and H. V. Jagadish. 2016. Understanding Natural Language Queries over Relational Databases. *SIGMOD Record* 45, 1 (2016), 6–13. doi:10.1145/2949741.2949744

[12] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2024).

[13] Kyle Luoma and Arun Kumar. 2025. SNAILS: Schema Naming Assessments for Improved LLM-Based SQL Inference. *Proceedings of the ACM on Management of Data* 3, 1, Article 77 (2025). doi:10.1145/3709727

[14] Daphne Miedema and George H. L. Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–5. doi:10.1109/VL/HCC51201.2021.9576431

[15] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. Article 142. doi:10.1145/3613904.3641936

[16] Arpit Narechania, Adam Fourney, Bongshin Lee, and Gonzalo Ramos. 2021. DIY: Assessing the correctness of natural language to SQL systems. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*. 597–607. doi:10.1145/3397481.3450667

[17] Zheng Ning, Yuan Tian, Zheng Zhang, Tianyi Zhang, and Toby Jia-Jun Li. 2024. Insights into Natural Language Database Query Errors: From Attention Misalignment to User Handling Strategies. *ACM Transactions on Interactive Intelligent Systems* 14, 4, Article 25 (2024), 32 pages. arXiv:2402.07304 doi:10.1145/3650114

[18] Raja Parasuraman and Victor Riley. 1997. Humans and Automation: Use, Misuse, Disuse, Abuse. *Human Factors* 39, 2 (1997), 230–253. doi:10.1518/001872097778543886

[19] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New

Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1577, 10 pages.

[20] Alkis Simitsis and Yannis E. Ioannidis. 2009. DBMSs Should Talk Back Too. In *Fourth Biennial Conference on Innovative Data Systems Research (CIDR 2009), Asilomar, CA, USA, January 4–7, 2009, Online Proceedings*. www.cidrdb.org. arXiv:0909.1786 https://arxiv.org/abs/0909.1786

[21] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. 2018. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education* 18, 3, Article 15 (2018), 29 pages. doi:10.1145/3231712

[22] Yuan Tian, Jonathan K. Kummerfeld, Toby Jia-Jun Li, and Tianyi Zhang. 2024. SQLucid: Grounding Natural Language Database Queries with Interactive Explanations. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 12, 20 pages. doi:10.1145/3654777.3676368

[23] Yuan Tian, Zheng Zhang, Zheng Ning, Toby Jia-Jun Li, Jonathan K. Kummerfeld, and Tianyi Zhang. 2023. Interactive Text-to-SQL Generation via Editable Step-by-Step Explanations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 16149–16166. doi:10.18653/v1/2023.emnlp-main.1004

[24] Yannis Vassiliou, Matthias Jarke, Edward A. Stohr, Jon A. Turner, and Norman H. White. 1983. Natural Language for Database Queries: A Laboratory Study. *MIS Quarterly* 7, 4 (1983), 47–61. doi:10.2307/248746

[25] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis, and Fred D. Davis. 2003. User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly* 27, 3 (2003), 425–478. doi:10.2307/30036540

[26] Niklas Wretblad, Fredrik Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. 2024. Understanding the Effects of Noise in Text-to-SQL: An Examination of the BIRD-Bench Benchmark. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 356–369. doi:10.18653/v1/2024.acl-short.34

[27] Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-Based Interactive Semantic Parsing: A Unified Framework and a Text-to-SQL Case Study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 5447–5458. doi:10.18653/v1/D19-1547

[28] 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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921. doi:10.18653/v1/D18-1425