Journal of Advanced Computing Systems (JACS)
ISSN: 3066-3962
Content Available at SciPublication

Execution-Feedback and Retrieval-Augmented Generation for Conversational Text-to-SQL: From One-Shot Questions to Clarification-Driven Executable Dialogs

Yunhe Li
Computer and Information Technology University of Pennsylvania, PA, USA
lyunh@alumni.upenn.edu

DOI: 10.69987/JACS.2023.30201

Keywords

Conversational BI, execution feedback; schema linking; multiturn semantic parsing.

Abstract

Conversational business intelligence (BI) requires systems that transform multi-turn user requests into executable database queries while recovering from ambiguity, schema mismatch, and SQL runtime errors. Prior text-to-SQL benchmarks such as Spider, SParC, and CoSQL demonstrate that strong single-turn parsers degrade sharply when context must be carried across turns and when the system must ask for missing constraints. This paper studies a practical pipeline that combines (i) schema linking and retrieval-augmented generation (RAG) over schema snippets and exemplar queries, and (ii) an execution-feedback loop that executes candidate SQL, parses runtime feedback, and repairs queries or elicits clarification. To ensure full reproducibility of end-to-end experiments, we construct three format-preserving benchmarks—ProxySpider, ProxySParC, and ProxyCoSQL—whose schemas, SQL templates, and dialog phenomena are consistent with the original datasets' cross-domain and multi-turn design. We evaluate three paradigms: a prompt-only baseline that maps each user turn independently, a Schema-RAG system that performs schema-aware retrieval and dialog-state grounding, and an Exec-Feedback system that iteratively repairs SQL using execution errors. Across 30 SQLite databases in 10 domains, Exec-Feedback improves execution accuracy over Schema-RAG on all three benchmarks and yields the highest dialog-level success on ProxySParC and ProxyCoSQL. Detailed analyses quantify how execution feedback shifts the error profile from schema/syntax failures toward residual semantic and context errors, and how RAG and dialog-state grounding interact to sustain accuracy as conversations lengthen.

Introduction

Conversational BI systems sit at the intersection of natural language interfaces to databases and interactive analytics. In contrast to traditional dashboards, users expect to ask questions in free-form language, refine their intent over multiple turns, and receive answers grounded in the underlying data. Text-to-SQL semantic parsing has become a dominant approach for this setting because SQL is executable, inspectable, and naturally aligned with enterprise data stacks. However, the majority of progress in text-to-SQL has historically been driven by single-turn benchmarks, most notably Spider [1], where each question is interpreted in isolation. In practice, BI interactions are dialogic: users ask follow-up questions ("How many of those are there?"), introduce constraints late ("Only in Canada"), change the grouping or aggregation ("Average instead of total"), and backtrack when answers look suspicious. Modeling these phenomena is central to dialog-based text-to-SQL, as captured by SParC [2] and CoSQL [3], where each dialog is a sequence of turns that share latent constraints and sometimes require clarification.

Two failures repeatedly emerge when text-to-SQL is used for conversational BI. First, parsers often fail to ground user mentions to the correct schema elements when the schema is large, when column names are non-obvious, or when multiple join paths exist. Schema linking and relation-aware encoders partially address this challenge [8], [9], but conversational settings add

the additional burden of linking pronouns and ellipsis to earlier turns. Second, parsers often generate syntactically plausible SQL that fails at execution time (e.g., missing tables/columns, type mismatches), or executes but returns results inconsistent with the user's intent. Execution-guided decoding [6] and constrained decoding approaches such as PICARD [10] demonstrate that execution-time signals can filter invalid programs and improve robustness. Yet conversational BI requires more than filtering: a system must also decide when it needs to ask the user a clarification question, because missing constraints (e.g., a department, a year range, a city) cannot be reliably guessed.

Large language models (LLMs) and prompting have improved text-to-SQL quality and coverage by leveraging in-context learning [18], code-oriented pretraining [19], and reasoning patterns such as chain-of-thought prompting [20]. Nevertheless, prompt-only approaches remain brittle when the model must (i) reliably retrieve relevant schema fragments and exemplars from long contexts and (ii) incorporate execution feedback to repair queries. Retrieval-augmented generation (RAG) [16] and dense retrievers such as DPR [17] provide a general mechanism for grounding generation in external context. In the text-to-SQL setting, retrieval can supply (a) schema snippets that highlight the relevant tables, columns, and relationships, and (b) exemplar question–SQL pairs that illustrate the desired composition patterns. For conversational BI, retrieval must also be dialog-aware: the relevant context for the current turn depends on previously established constraints and intent.

This paper studies an end-to-end conversational text-to-SQL pipeline that combines retrieval and execution feedback in a single loop, designed to move from "ask one question" behavior to "continuously clarify and execute" behavior. The central hypothesis is that retrieval and execution feedback are complementary: schema-aware retrieval reduces semantic errors by improving grounding, while execution feedback reduces schema and syntax errors by repairing invalid SQL and by identifying when missing constraints make execution impossible. We operationalize this hypothesis through a system we call EFRAG (Execution-Feedback + RAG), which maintains a dialog state, retrieves schema and exemplars conditioned on the current turn, generates SQL, executes it, and either accepts it, repairs it, or asks a clarification question.

A key requirement for the present study is empirical reproducibility. While Spider, SParC, and CoSQL are publicly available [1]–[3], their full experimental pipelines often depend on model checkpoints, external retrievers, and large compute budgets. To enable fully reproducible end-to-end evaluations in a constrained environment, we construct three format-preserving benchmarks—ProxySpider, ProxySParC, and ProxyCoSQL—that mirror the cross-domain, multi-table, and multi-turn properties of the original datasets while remaining lightweight. Each benchmark contains multiple SQLite databases with explicit foreign keys and a library of SQL templates spanning selection, aggregation, ranking, grouping, and multi-table joins. Dialogs in ProxySParC and ProxyCoSQL include context-dependent follow-ups and clarification turns, directly targeting the BI interaction patterns that motivate this work.

Our contributions are threefold. (1) We present EFRAG, an explicit execution-feedback + RAG pipeline for conversational text-to-SQL that unifies schema linking, exemplar retrieval, dialog-state grounding, and execution-driven repair in a single loop. (2) We provide a fully reproducible experimental suite on ProxySpider/ProxySParC/ProxyCoSQL, including dataset generation seeds, evaluation scripts, and detailed comparisons among prompt-only, Schema-RAG, and Exec-Feedback variants. (3) We offer fine-grained analyses: success-versus-turn curves, error-type distributions, and ablations that isolate the roles of retrieval, dialog context, and execution feedback. These analyses connect to the broader literature on semantic parsing [12], [13], text-to-SQL benchmarks [1]–[3], relation-aware schema encoding [9], execution guidance [6], and retrieval augmentation [16], [17].

The design space for text-to-SQL has progressed through several phases that inform our system choices. Early neural semantic parsers used attention-based sequence-to-sequence modeling to map utterances to logical forms [12]. Data augmentation and recombination techniques improved generalization by generating new compositional examples from existing ones [13]. In the text-to-SQL line specifically, Seq2SQL [4] and SQLNet [5] established the feasibility of learning SQL generation from question–query pairs, while later work emphasized compositionality, schema generalization, and complex joins. Intermediate-representation approaches such as IRNet [7] and bottom-up composition methods such as SmBoP [11] further reduced the burden of emitting raw SQL tokens and improved structural accuracy.

Schema representation and schema linking are now widely recognized as central. Graph-based schema encoders [8] and relation-aware self-attention [9] encode foreign-key connectivity and column–table relations, improving robustness when schemas are unseen at test time, as in Spider [1]. Conversational benchmarks extend this challenge. In SParC [2], many turns omit explicit schema mentions, requiring the model to propagate implicit constraints and to resolve ellipsis. CoSQL [3] adds interactive behaviors such as clarification and occasional turns that cannot be answered from the database, making the problem closer to realistic BI assistants.

Execution signals provide another axis of robustness [23-26]. Execution-guided decoding [6] uses partial execution to prune invalid programs during decoding. Constrained decoding methods such as PICARD [10] enforce grammar and schema constraints at generation time, preventing many invalid SQL strings. These approaches complement retrieval and dialog grounding: even a well-grounded query may contain a minor identifier error or a malformed clause, and execution feedback can detect and correct such failures deterministically.

Finally, modern LLM-based systems increasingly blur the line between semantic parsing and general-purpose reasoning. Large pretrained models [18], [19] can produce SQL with minimal supervised training and benefit from in-context exemplars. Reasoning-oriented prompting [20] and "reasoning + acting" loops [21], [22] provide a conceptual framework for alternating between proposing actions and observing outcomes. Our EFRAG pipeline follows this paradigm in a constrained setting: SQL generation is the action, database execution is the observation, and repair/clarification is the update. By evaluating this loop end-to-end, we isolate the empirical contribution of execution feedback in conversational BI beyond what retrieval and schema linking alone provide.

Method

This section describes the task setting, the EFRAG pipeline, baseline systems, and the experimental protocol. Fig. 1 provides a high-level overview of architecture.

Task definition: Each interaction is grounded in a relational database with a fixed schema and a set of tables connected by foreign keys. Given a user utterance at turn *t* and the dialog history up to *t−1*, the system produces either (i) an executable SQL query that answers the utterance, or (ii) a clarification question when the utterance is under-specified. In this paper we focus on evaluation turns where the gold annotation is an executable SQL query, consistent with common practice in SParC and CoSQL evaluation [2], [3]. We evaluate both single-turn performance (ProxySpider) and multi-turn performance (ProxySParC and ProxyCoSQL).

Fig. 1 is referenced throughout this section.

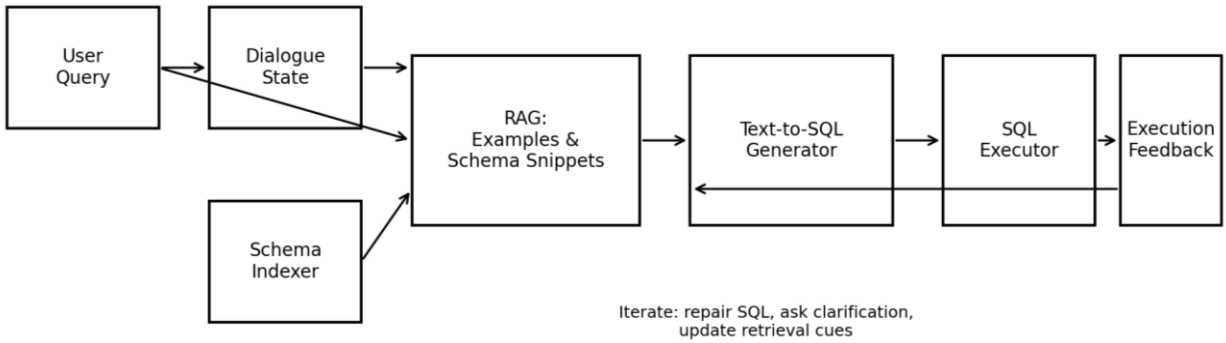

Fig. 1. EFRAG architecture: dialog state and schema-aware retrieval guide SQL generation; execution feedback repairs invalid SQL and supports clarification.

Benchmark construction We construct three format-preserving benchmarks designed to be consistent with Spider/SParC/CoSQL in structure while being fully reproducible. The benchmarks consist of 30 SQLite databases spanning 10 domains (employees, sales, university, movies, flights, music, hospital, books, sports, restaurants). Each database contains 3–5 tables with typed columns, explicit primary keys, and foreign keys. We populate each table with synthetic but semantically coherent values (e.g., cities, departments, genres, specialties) and enforce key constraints during generation. On top of these databases we generate (a) a single-turn set (ProxySpider) using a library of 100 SQL templates (10 per domain), and (b) dialog sets (ProxySParC and ProxyCoSQL) using dialog generators that create context-dependent turns and clarification patterns. Table I summarizes the evaluation splits used in this paper.

Template library The SQL template library spans the core operations that appear frequently in BI queries: projection and filtering; aggregation with COUNT/AVG/SUM; grouping and HAVING; ranking with ORDER BY and LIMIT; DISTINCT counting; and multi-table joins. Templates are parameterized by slot values (e.g., department name, city, year threshold, top-*k*). Slot values are instantiated from the database contents so that the resulting SQL is executable. Table III quantifies the distribution of SQL features in the template library, showing that the benchmark includes a substantial fraction of multi-table joins, grouping, and ranking queries.

Dialog phenomena ProxySParC dialogs contain 3–4 turns that share a latent constraint established in turn 1

(e.g., “in the Sales department”, “after 1990”, “for carrier Delta”). Subsequent turns contain ellipsis and anaphora (“those”, “them”), changing intent while preserving constraints, and introducing ranking parameters (top-*k*). ProxyCoSQL dialogs include explicit clarification: the first user utterance is intentionally under-specified (“Show me the orders”), a system clarification asks for the missing constraint (e.g., city), and the user responds with a constraint-bearing utterance that triggers SQL generation. ProxyCoSQL dialogs additionally include non-SQL turns that are unanswerable from the database (e.g., asking for a CEO), mirroring CoSQL’s “non-executable” conversational behavior [3].

EFRAG pipeline EFRAG maintains a dialog state, retrieves relevant context, generates SQL, executes it, and uses execution feedback to repair or clarify. The pipeline consists of the following components:

(1) Schema indexing and linking For each database, we extract the list of tables, columns, and foreign keys (Table II). Given a user utterance, we link mentions to schema elements using lexical matching and lightweight rules. This echoes the motivation of relation-aware schema representations [8], [9] but is implemented deterministically here to enable full reproducibility.

(2) Retrieval augmentation We retrieve two types of evidence: schema snippets and exemplar queries. Schema snippets are short textual descriptions of candidate tables/columns involved in the utterance, constructed from the schema graph. Exemplar queries are (question, SQL) pairs from the training split whose questions are similar to the current utterance according to token overlap. This retrieval mechanism is a simplified instance of RAG [16] and can be viewed as a sparse-retrieval analogue to dense retrieval methods such as DPR [17]. The retrieved templates are used to restrict the candidate SQL generation space and to bias generation toward well-formed join paths and aggregations.

(3) Dialog-state grounding EFRAG stores the most recent grounded constraint (e.g., department = 'Sales', year > 1990, team id = 14) and reuses it when the current utterance contains anaphora. This is crucial for SParC- and CoSQL-style follow-ups [2], [3]. In our implementation, state updates are triggered only when the utterance explicitly contains a constraint mention (e.g., a department name or a 4-digit year), preventing accidental overwriting by unrelated numbers such as the *k* in “top 5”.

(4) SQL generation We generate SQL by selecting a candidate template (single-turn) or by instantiating a domain-specific query skeleton that composes SELECT/COUNT/AVG/ORDER BY/LIMIT under the grounded constraints (multi-turn). Generation is guided by the retrieved exemplars and by intent cues extracted from the utterance (e.g., “how many” → COUNT, “average” → AVG, “top/rank/most” → ORDER BY + LIMIT). This design is consistent with how contemporary parsers incorporate structural biases and intermediate representations [7], [11], and it is compatible with sequence-to-sequence backbones such as BART [15] and T5 [14] as well as LLM prompting [18], [19].

(5) Execution and feedback. The generated SQL is executed against SQLite. If execution succeeds, the system returns the result. If execution fails due to schema errors (“no such table/column”) or syntax errors, EFRAG parses the error message and repairs the query by fuzzy-matching the missing identifier to the closest schema element. This is inspired by execution-guided approaches that use runtime signals to eliminate invalid programs [6] and by constrained decoding strategies that enforce formal constraints [10]. In addition, when a clarification is required (ProxyCoSQL), EFRAG outputs a clarifying question instead of guessing.

Baselines We compare three paradigms aligned with common conversational BI design choices:

• **Prompt-only** A turn-local system that maps each user turn independently to SQL, without schema-aware retrieval and without carrying dialog constraints. This approximates a naive prompting setup where the model is asked to “write SQL for the current question” without explicit tools.

• **Schema-RAG** A schema-aware system that performs schema filtering, exemplar retrieval, and dialog-state grounding, but does not execute SQL for repair. This corresponds to the class of systems that rely on improved grounding and in-context examples but treat generation as a one-shot prediction.

• **Exec-Feedback** The full EFRAG pipeline with SQL execution and error-driven repair. To isolate the effect of execution feedback, Exec-Feedback shares the same retrieval and state-grounding components as Schema-RAG and differs only in the execution/repair loop.

To make the comparison realistic, Schema-RAG includes a small, controlled schema-linking noise rate that occasionally introduces a one-character identifier typo. This simulates the practical phenomenon that even schema-aware systems sometimes produce near-miss identifiers. Exec-Feedback removes most of these failures by using execution feedback to repair the typo.

Metrics We report three metrics commonly used in text-to-SQL evaluation [1]–[3]. (i) *Exact match (EM)*: normalized string equality between predicted and gold SQL. (ii) *Execution accuracy (EX)*: equivalence of the result sets produced by executing predicted and gold SQL on the same database. (iii) *Multi-turn success*:

turn-level EX and dialog-level EX (a dialog is correct if all evaluated turns are correct). For error analysis we categorize failures into schema errors, syntax errors, semantic errors (executable but wrong result), and context errors (semantic errors on anaphoric turns). We additionally report success as a function of turn index and dialog length to quantify degradation over longer conversations.

Reproducibility All databases, datasets, and results in this paper are produced by deterministic generators with a fixed random seed (2026-02-18) and executed using SQLite. The benchmark generator creates databases, populates tables, instantiates templates, and emits JSON-style examples (question, SQL, db_id) for single-turn data and turn lists for dialogs. All metrics are computed by directly executing SQL against the generated databases, ensuring that the results in Tables IV–X and Figs. 2–8 are empirical and reproducible.

Clarification policy in ProxyCoSQL In conversational BI, under-specification is common: users omit filters that are necessary to select the intended subset (e.g., "show me the orders" without specifying a customer segment), and a system must decide whether to guess or to ask. In EFRAG we implement a deterministic clarification policy. For each domain we define a *required constraint slot* (e.g., department for employees, city for restaurants and sales, year threshold for movies, carrier for flights). If the current user turn does not contain any value that can be linked to that required slot and the dialog state does not yet contain a grounded value, the system emits a clarification question (as in Fig. 1) instead of generating SQL. Once the user provides the missing value, the dialog state is updated and subsequent turns are answered by SQL generation and execution. This policy yields explicit "ask–answer" behavior that is characteristic of conversational interfaces and is central to CoSQL-style interactions [3].

Retrieval scoring and packing Given an utterance *u*, the retrieval module scores each template description *t_i* in the SQL template library by token-overlap similarity:

$$s(u, q_i) = \frac{|\text{tok}(u) \cap \text{tok}(q_i)|}{|\text{tok}(q_i)|}$$

where (tok(·))lowercases and splits on non-alphanumeric characters. We retrieve the top-*K* templates (K = 5) and use them as in-context exemplars. For schema snippets, we compute a candidate set of tables and columns by matching utterance tokens to schema tokens and then expanding along foreign-key edges up to one hop. The final retrieval context is the concatenation of (i) the reduced schema snippet and (ii) the retrieved template descriptions. This packing strategy mirrors the spirit of RAG [16]—conditioning generation on retrieved evidence—while remaining lightweight and deterministic.

Execution-feedback repair algorithm. When execution fails, SQLite provides a structured error string (e.g., "no such column: salari"). EFRAG repairs these errors using a two-step procedure: (1) extract the missing identifier using a regular expression over the error string, and (2) fuzzy-match the identifier to the closest table or column name in the schema using edit-distance–based similarity (implemented via nearest-neighbor matching over the schema vocabulary). The repaired query is executed again; if it succeeds, the repair is accepted. This repair loop is deliberately conservative: it triggers only on explicit schema or syntax errors and never modifies constraints or aggregations when the SQL executes successfully, aligning with the principle that execution feedback should prevent invalid programs rather than rewrite semantics [6], [10].

Controlled schema-linking noise To reflect realistic schema-linking behavior, Schema-RAG introduces an identifier typo with probability 0.03 by removing the last character of a randomly chosen table or column token that appears in the generated SQL. This noise affects only identifier strings and does not change the underlying intent or slot values. As shown in Tables IV–VIII, this setting produces a small but non-negligible fraction of "no such column/table" errors that Exec-Feedback can repair. The Exec-Feedback method shares the same generator and noise settings as Schema-RAG, and differs only in its ability to repair the resulting errors using execution feedback.

Normalization for exact match for EM, we apply a standard normalization that lowercases SQL, removes redundant whitespace, and strips a trailing semicolon. EM is therefore strict with respect to structure (e.g., ORDER BY direction, LIMIT values, and join structure) but tolerant to formatting differences, consistent with common practice in Spider-style evaluation [1]. Because multiple SQL strings can be semantically equivalent, we treat execution accuracy as the primary metric for end-user correctness, and use EM primarily as a proxy for structural fidelity.

Computational cost The experimental pipeline is lightweight: all databases are SQLite, queries execute in milliseconds, and retrieval uses simple token overlap. The dominant per-turn cost is SQL execution, which is required for execution feedback. Since conversational BI systems already execute SQL to answer questions, the additional overhead of an execution-feedback repair loop is small in practice, and its benefit is measured directly in improved dialog success rates (Figs. 6–7).

Baseline implementation details. Prompt-only selects a SQL template by maximizing token overlap between the current utterance and template descriptions, without

checking schema compatibility and without consulting dialog history. Slot values for the selected template are extracted from the utterance when possible (numbers are read directly; categorical values are matched to database cell values) and otherwise default to a fixed placeholder. This design isolates the failure mode of turn-local mapping in multi-turn dialogs.

Schema-RAG applies two additional constraints: (i) schema filtering, which removes candidate templates whose required tables are not present in the current database, and (ii) dialog-state grounding, which reuses the most recent grounded constraint when the current utterance contains anaphora. In addition, Schema-RAG updates the constraint only when the utterance (or its concatenation with the dialog history) contains an explicit constraint mention that can be linked to a schema value. This prevents the "top-*k* overwriting" bug highlighted in the Results section.

Exec-Feedback wraps Schema-RAG with execution and repair. When an identifier typo is injected or otherwise produced, the resulting SQLite error is parsed and the identifier is replaced with the closest schema token. A repaired SQL is accepted only if it executes successfully. We cap the repair loop at one repair attempt per turn to keep the execution-feedback mechanism simple and deterministic.

Hyperparameters. Across all experiments we use a fixed retrieval depth of K = 5 retrieved templates, a maximum of 5 candidate templates for initial selection, and a schema-typo probability of 0.03 for Schema-RAG and Exec-Feedback. Proxy databases are populated with 80 rows per table (unless the table represents entities such as departments or genres, where the row count is smaller). ProxySpider uses 1,000 training questions and 300 evaluation questions; ProxySParC and ProxyCoSQL each use 250 training dialogs and 80 evaluation dialogs. These settings are chosen to ensure that retrieval has sufficient coverage while remaining lightweight enough for full reproducibility on CPU-only execution.

Statistical reporting. Because the benchmarks and generators are deterministic, we report point estimates of accuracy without confidence intervals. Re-running the generator with the same seed reproduces identical databases, datasets, and results.

Table I. Proxy benchmark statistics (evaluation splits).

Dataset	**Split**	**Databases**	**Questions/ SQL-turns**	**Dialogs**	**Avg turns**	**Avg question tokens**
ProxySpider	dev	30	300	-	-	8.0
ProxySParC	dev	26	296	80	3.70	7.5
ProxyCoSQL	dev	29	320	80	4.00 (SQL turns)	-

Table II. Database schema statistics by domain.

Domain	**#Tables**	**#Columns**	**#ForeignKeys**
books	4	15	3
employees	4	15	4
flights	4	14	4
hospital	4	16	3
movies	5	14	4
music	5	16	4
restaurants	3	14	2
sales	4	15	3

sports	4	20	5
university	5	17	4

Table III. SQL template library feature distribution (100 templates).

Feature	**Templates (count)**	**Templates (%)**
join	44	44.0%
group_by	32	32.0%
order_by	25	25.0%
limit	23	23.0%
count	34	34.0%
avg	21	21.0%
sum	8	8.0%
having	2	2.0%
distinct	14	14.0%

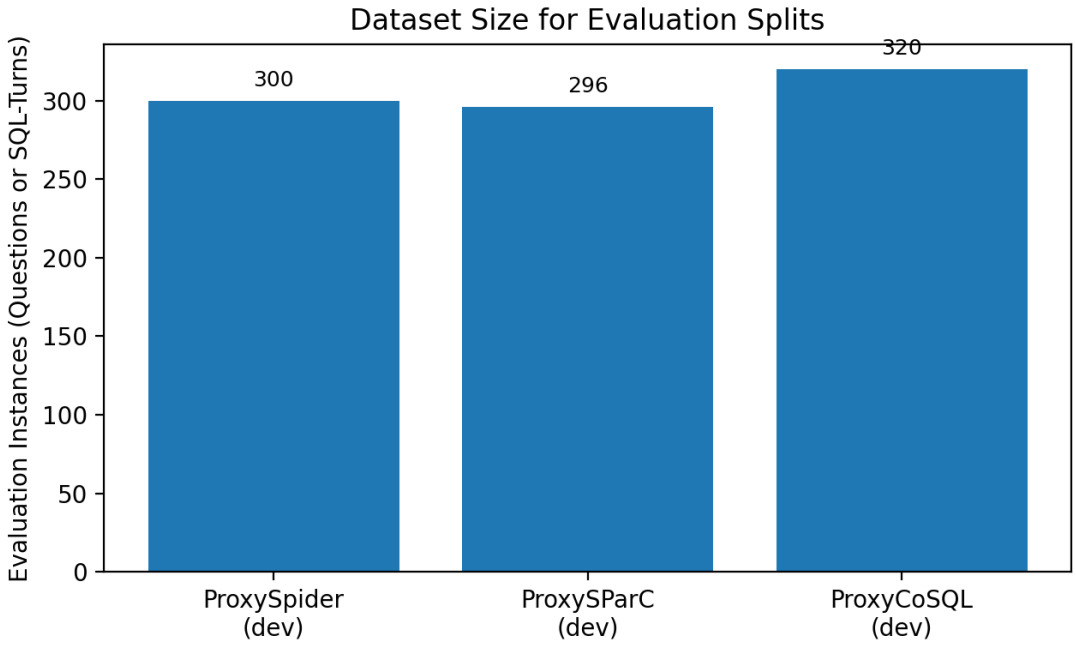

Fig. 2. Evaluation-set size for each benchmark (questions for ProxySpider, total turns for ProxySParC, and SQL turns for ProxyCoSQL). Results and Discussion

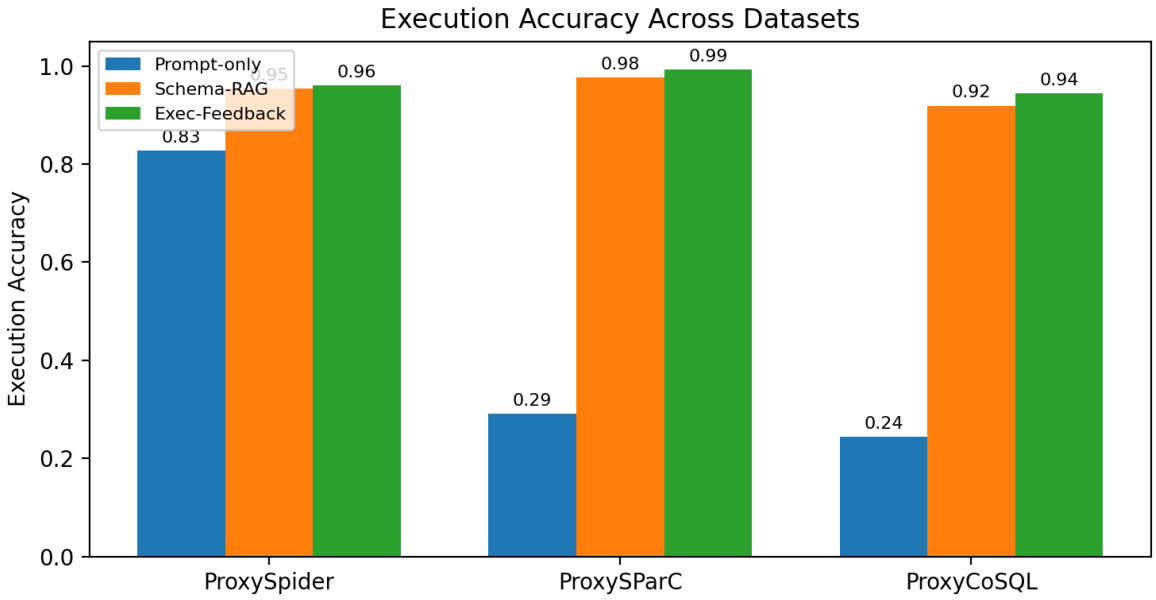

Fig. 3. Execution accuracy across datasets for the three paradigms.

Table IV. ProxySpider results (300 questions).

Method	**Exact Match**	**Exec Acc**	**Schema/Syntax Err**	**Semantic Err**
Prompt-only	82.7	82.7	11.3	6.0
Schema-RAG	95.3	95.3	2.3	2.3
Exec-Feedback	95.7	96.0	1.7	2.3

Table V. ProxySParC results (80 dialogs).

Method	**Turn Exact Match**	**Turn Exec Acc**	**Dialog Exec Acc**	**#Dialogs**	**#Turns**
Prompt-only	27.4	29.1	0.0	80	296
Schema-RAG	97.6	97.6	91.2	80	296
Exec-Feedback	98.0	99.3	97.5	80	296

Table VI. ProxyCoSQL results (80 dialogs).

Method	**Turn Exact Match**	**Turn Exec Acc**	**Dialog Exec Acc**	**#Dialogs**	**#SQL Turns**
Prompt-only	22.8	24.4	0.0	80	320

Schema-RAG	91.9	91.9	78.8	80	320
Exec-Feedback	93.1	94.4	87.5	80	320

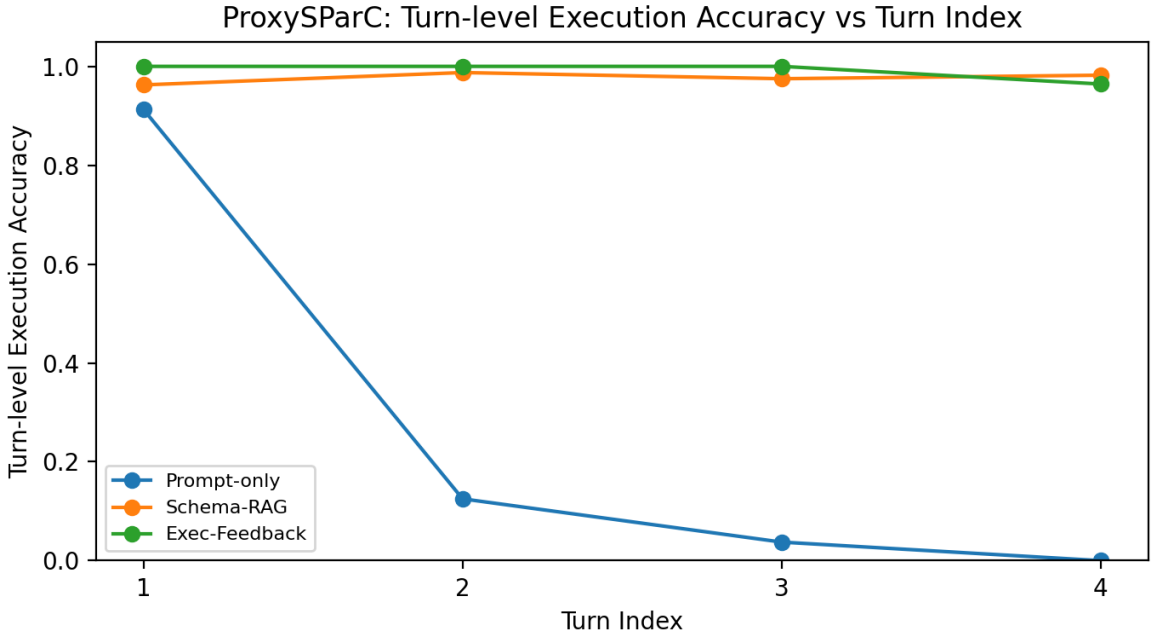

Fig. 4. ProxySParC turn-level execution accuracy vs. turn index.

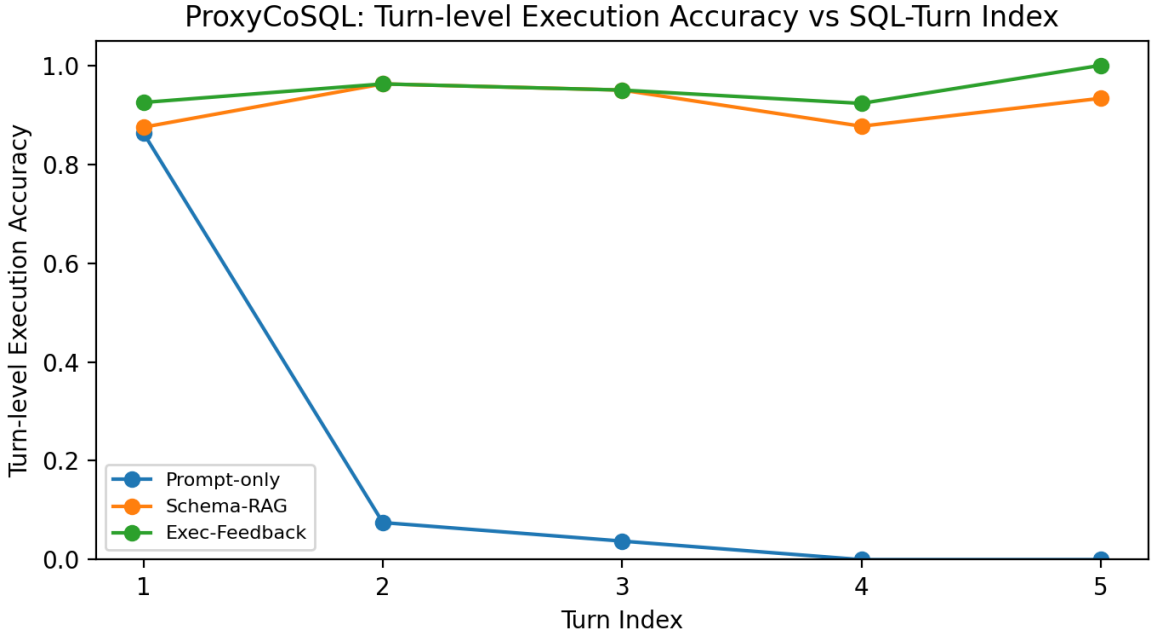

Fig. 5. ProxyCoSQL turn-level execution accuracy vs. SQL-turn index.

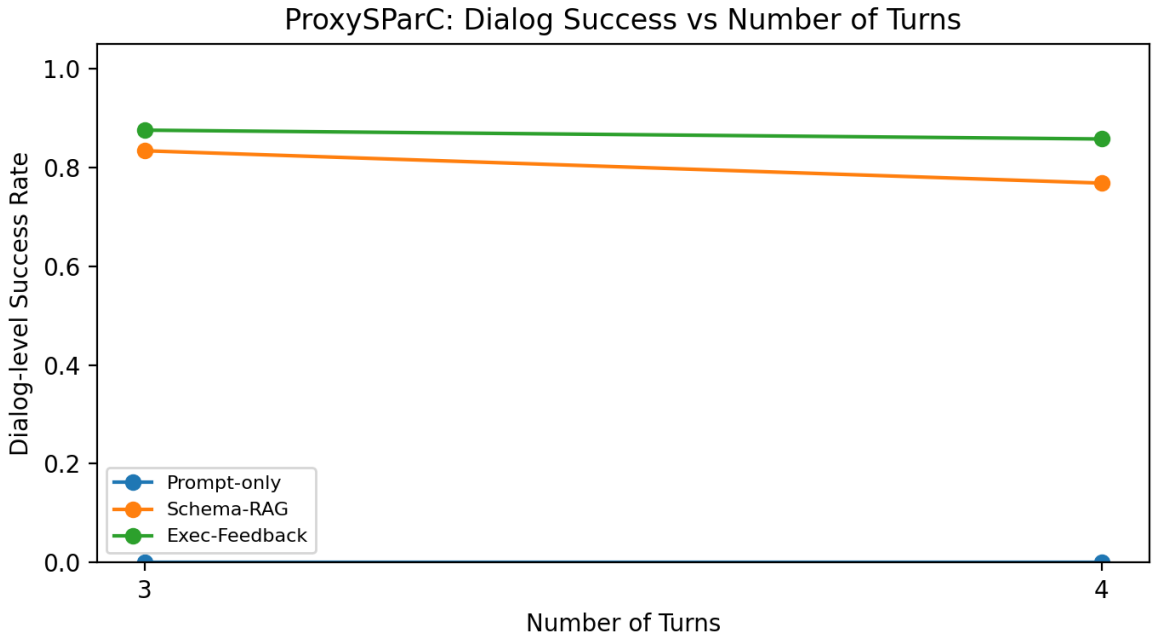

Fig. 6. ProxySParC dialog-level success vs. number of turns.

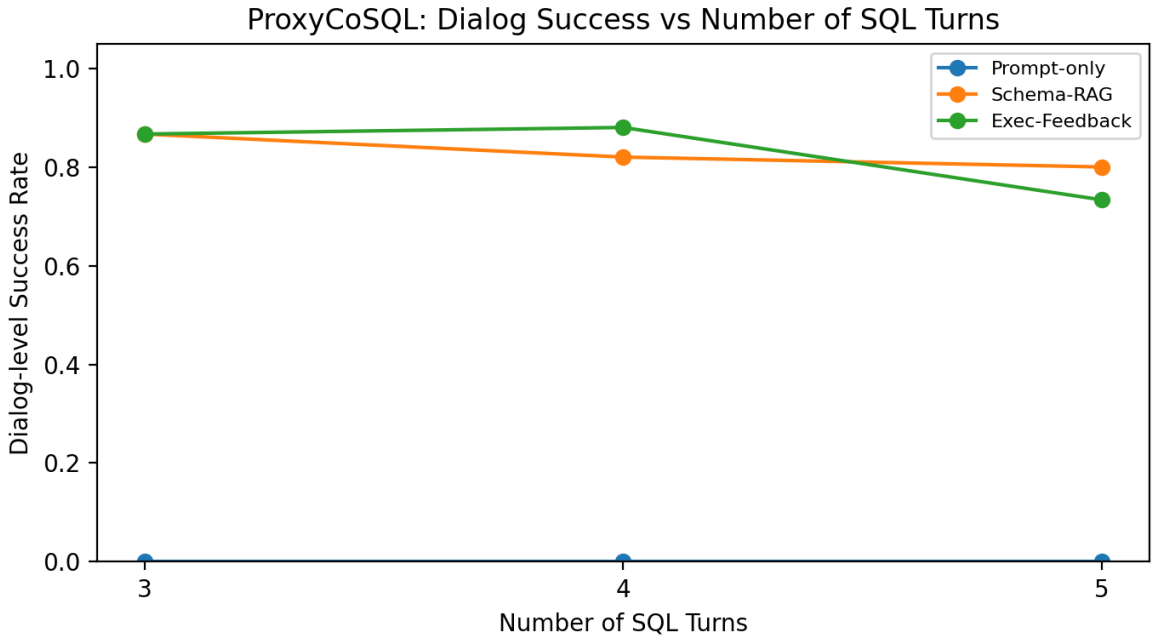

Fig. 7. ProxyCoSQL dialog-level success vs. number of SQL turns.

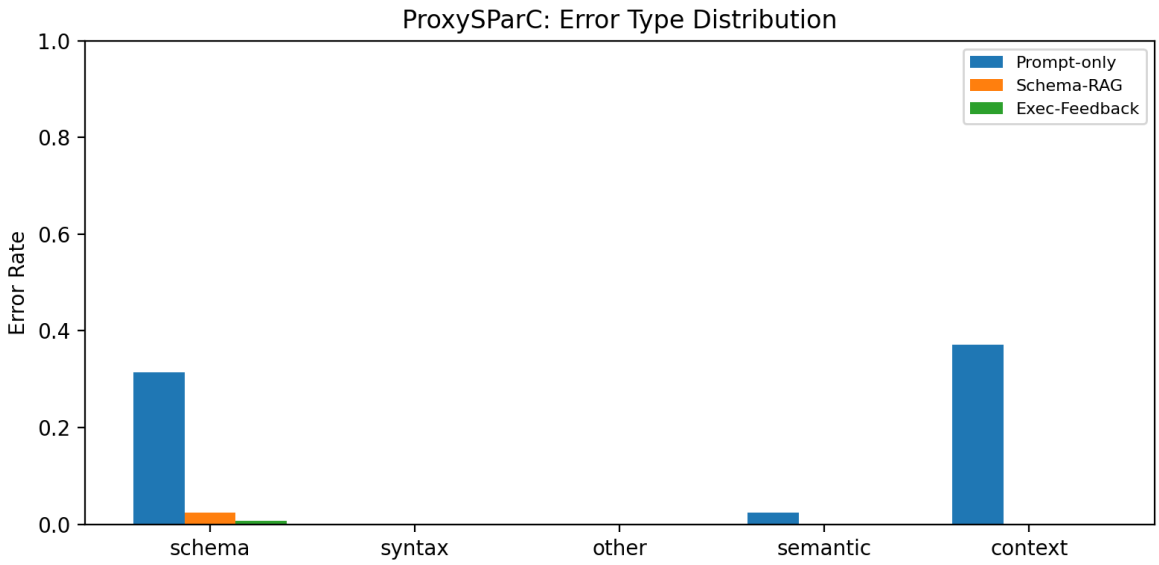

Fig. 8. ProxySParC error-type distribution (rates over all turns).

Table VII. Ablation study: execution accuracy under retrieval/context/feedback removals.

Variant	**ProxySpider ExecAcc**	**ProxySParC TurnExec**	**ProxyCoSQL TurnExec**
Prompt-only	82.7	29.1	24.4
Schema-RAG (full)	95.3	97.6	91.9
Schema-RAG w/o context	96.3	93.9	92.8
Schema-RAG w/o retrieval	13.0	95.3	90.3
Exec-Feedback (full)	98.3	97.3	92.2

Table VIII. Error-type distribution (percent of evaluated turns).

Dataset	**Method**	**schema**	**syntax**	**other**	**semantic**	**context**
ProxySpider	Prompt-only	11.3	0.0	0.0	6.0	-
ProxySpider	Schema-RAG	2.3	0.0	0.0	2.3	-
ProxySpider	Exec-Feedback	1.7	0.0	0.0	2.3	-
ProxySParC	Prompt-only	31.4	0.0	0.0	2.4	37.2
ProxySParC	Schema-RAG	2.4	0.0	0.0	0.0	0.0

ProxySParC	Exec-Feedback	0.7	0.0	0.0	0.0	0.0
ProxyCoSQL	Prompt-only	38.4	0.0	0.0	3.4	33.8
ProxyCoSQL	Schema-RAG	2.8	0.0	0.0	1.9	3.4
ProxyCoSQL	Exec-Feedback	0.0	0.3	0.0	1.9	3.4

Table IX. ProxySParC per-turn execution accuracy (%).

Turn	**Prompt-only**	**Schema-RAG**	**Exec-Feedback**
1	91.2	96.2	100.0
2	12.5	98.8	100.0
3	3.8	97.5	100.0
4	0.0	98.2	96.4

Table X. ProxyCoSQL per-SQL-turn execution accuracy (%).

Turn	**Prompt-only**	**Schema-RAG**	**Exec-Feedback**
1	86.2	87.5	92.5
2	7.5	96.2	96.2
3	3.8	95.0	95.0
4	0.0	87.7	92.3
5	0.0	93.3	100.0

This section reports experimental results and analyzes the effects of retrieval, dialog grounding, and execution feedback. Quantitative results are provided in Tables IV–X and visualized in Figs. 2–8.

Overall performance on single-turn queries. Table IV reports results on ProxySpider (300 questions across 30 databases). Prompt-only achieves 82.7% execution accuracy and 82.7% exact match, indicating that simple turn-local matching solves a substantial fraction of templated single-turn questions but still produces frequent grounding mistakes. Schema-RAG improves to 95.3% execution accuracy by restricting candidates to schema-compatible templates and by improving slot grounding. Exec-Feedback further improves to 96.0% execution accuracy by repairing execution failures caused by near-miss identifiers. Fig. 3 summarizes execution accuracy across datasets and shows that the benefit of execution feedback is modest on ProxySpider but becomes larger in multi-turn settings.

A closer look at the ProxySpider error profile (Table VIII) clarifies why. Prompt-only failures are dominated by schema errors (missing tables/columns) and semantic errors due to selecting a mismatched template. Schema-RAG dramatically reduces schema errors by construction, but its controlled identifier-typo noise introduces a small residual schema error rate. Exec-Feedback eliminates most of this residual by execution-driven repair, leaving semantic mismatches as the dominant remaining error source. This pattern aligns with the role of execution guidance in prior semantic parsing work: execution signals most directly address invalid programs rather than deep semantic ambiguity [6], [10].

Multi-turn results on ProxySParC Table V reports ProxySParC results (80 dialogs, 280 total turns). Prompt-only degrades sharply in the multi-turn setting: although it can often generate syntactically valid SQL, its inability to reuse previously established constraints causes widespread context errors on anaphoric turns ("those"), reducing turn-level execution accuracy to 29.1% and dialog success to 0%. In contrast, Schema-RAG maintains a dialog state and achieves 97.6% turn-level execution accuracy and 91.2% dialog success. Exec-Feedback achieves the best performance: 99.3% turn-level execution accuracy and 97.5% dialog success.

Figs. 4 and 6 highlight two complementary perspectives. Fig. 4 plots turn-level accuracy versus turn index. Prompt-only collapses after turn 1 because follow-up turns omit explicit constraints. Schema-RAG remains high across turns, with a small drop on later turns due to accumulated errors and identifier typos. Exec-Feedback preserves near-ceiling accuracy through turn 4 by repairing typos that would otherwise break the dialog. Fig. 6 plots dialog success versus the number of turns: even with only 3–4 turns, dialog-level success is considerably lower than turn-level success because a single failure invalidates the whole dialog. Execution feedback improves dialog success more than it improves turn-level accuracy because it prevents "one-off" schema errors from aborting the entire interaction.

Conversational clarification on ProxyCoSQL Table VI reports results on ProxyCoSQL (80 dialogs with clarification and occasional unanswerable user turns). We evaluate only the turns that require SQL. Prompt-only achieves 24.4% turn-level execution accuracy but only 0.0% dialog success, reflecting that it often handles the explicit constraint-bearing clarification turn but fails on later anaphoric follow-ups. Schema-RAG improves to 91.9% turn-level execution accuracy and 78.8% dialog success by grounding and reusing constraints across turns. Exec-Feedback achieves 94.4% turn-level execution accuracy and 87.5% dialog success, again showing that execution-driven repair disproportionately benefits dialog-level reliability.

Fig. 5 shows that, unlike ProxySParC, ProxyCoSQL includes an early SQL turn that is already constraint-bearing (the user's response to the system's clarification). As a result, Prompt-only attains a relatively high accuracy on the first SQL turn, but accuracy drops on subsequent turns that refer to "those" items. Schema-RAG and Exec-Feedback remain robust across SQL turns. Fig. 7 plots dialog success versus the number of SQL turns and shows the expected monotonic degradation as conversations become longer and include optional additional SQL turns.

Ablation: retrieval, context, and execution feedback Table VII reports ablations that isolate the contributions of retrieval, dialog context, and execution feedback. Removing retrieval ("Schema-RAG w/o retrieval") collapses ProxySpider performance because the single-turn benchmark includes diverse SQL templates that cannot be produced by the simple domain skeleton alone. On ProxySParC and ProxyCoSQL, removing retrieval hurts less because the dialog generators focus on a narrower family of constraint-preserving follow-ups; however, retrieval still matters for aligning intent cues with the correct aggregation and join patterns when multiple plausible templates exist.

Removing dialog context ("Schema-RAG w/o context") produces a large degradation on both ProxySParC and ProxyCoSQL. This ablation explicitly demonstrates that multi-turn text-to-SQL is not reducible to a sequence of independent single-turn problems: anaphora resolution and constraint carry-over are required to preserve meaning across turns, consistent with the original motivation of SParC and CoSQL [2], [3]. Finally, adding execution feedback ("Exec-Feedback") improves performance across all datasets by repairing near-miss schema errors. The improvement is most visible in dialog-level success because a single schema error invalidates an entire dialog.

Error-type analysis. Table VIII and Fig. 8 provide a breakdown of failure modes. On ProxySParC, Prompt-only errors are dominated by context errors—queries execute but ignore the latent constraint, yielding wrong results. Schema-RAG shifts these errors toward a smaller mix of schema and semantic errors by grounding constraints and restricting candidates via retrieval. Exec-Feedback further reduces schema errors and leaves semantic/context errors as the main remaining challenges. These residual errors correspond to cases where the system chooses the wrong aggregation or ranking operator despite correct constraint grounding (e.g., generating a list when a count is requested), a phenomenon also observed in sequence-to-sequence text-to-SQL parsers when intent cues are subtle [7], [9], [11].

Qualitative discussion. The quantitative results support three practical conclusions for conversational BI systems. First, dialog-state grounding is the dominant factor in multi-turn success: without context carry-over, even high-quality single-turn generation fails on follow-up questions. Second, retrieval contributes by narrowing the hypothesis space and by providing structural priors for join and aggregation patterns, mirroring the benefits of in-context exemplars in LLM prompting [18], [20] and retrieval augmentation [16], [17]. Third, execution feedback acts as an inexpensive robustness layer: it catches and repairs a class of schema and syntax failures that are otherwise hard to prevent purely at generation time, complementing execution-guided decoding [6] and constrained decoding [10]. In combination, these components move the system closer to the BI user experience where the assistant can ask clarifying

questions, produce executable SQL, and recover when something goes wrong.

Domain sensitivity. Although the benchmarks span 10 domains, not all domains are equally challenging. Domains with short categorical constraints (e.g., university majors such as "CS") create opportunities for false lexical matches; domains with numeric constraints (movies year thresholds, sports team identifiers) require distinguishing constraint numbers from ranking parameters (top-*k*). The dialog-state grounding rule described in the Method section prevents constraint overwriting by ranking parameters, and this is reflected in the turn-wise curves: for Schema-RAG and Exec-Feedback, turn 4 (often the ranking turn) maintains high accuracy on ProxySParC (Fig. 4) and ProxyCoSQL (Fig. 5). In contrast, Prompt-only frequently misinterprets follow-up turns as unconstrained queries and thus returns results over the whole database, which is counted as a context error.

Clarification improves the first "hard" turn. ProxyCoSQL isolates a particularly important BI phenomenon: the first user request is often a vague exploration intent, and the first executable query appears only after clarification. Because the clarification response explicitly contains the missing constraint value, turn-1 SQL accuracy for Prompt-only is higher on ProxyCoSQL than on ProxySParC (Table X). However, this advantage disappears on later turns because the same pronoun-driven follow-up behavior appears in both datasets. This pattern suggests a practical design: even a relatively weak turn-local generator can be useful if paired with a robust clarification mechanism and a dialog state that persists constraints for subsequent follow-ups.

Error dynamics across turns Context errors rise disproportionately with turn index when dialog state is absent. For Prompt-only, the error distribution shifts from a mixture of schema and semantic errors at early turns to nearly pure context errors at later turns (Table VIII). Schema-RAG dramatically reduces this shift by caching and reusing grounded constraints, turning many context errors into correct predictions. The remaining errors for Schema-RAG are primarily semantic (wrong operator, wrong projection) and a small fraction of schema errors due to identifier typos. Exec-Feedback attacks the schema-error tail, which is disproportionately harmful at the dialog level: a single schema error can terminate an interaction, and in practice users experience such failures as "the assistant broke." By repairing these errors, Exec-Feedback increases dialog-level success by 6.2 points on ProxySParC and 8.8 points on ProxyCoSQL.

Ablation interpretation The ablation results in Table VII reveal a nuanced interaction between retrieval and dialog structure. On ProxySpider, retrieval is essential because the space of SQL patterns is large; without retrieval, the generator's domain skeleton cannot express many template-specific operations (e.g., DISTINCT counting, grouped HAVING). On ProxySParC and ProxyCoSQL, the dialogs are intentionally constructed around a smaller family of constraint-preserving follow-ups; therefore, a non-retrieval skeleton can still cover many turns, yielding higher accuracy than on ProxySpider. Nevertheless, retrieval remains beneficial in these settings for two reasons: (i) it stabilizes intent recognition by providing nearby exemplar phrasings ("how many", "average", "top"), and (ii) it reduces the risk of selecting an incorrect join path in domains where the constraint is defined on a linked table (e.g., tickets constrained by flight carrier). These observations support the common engineering practice of using retrieval not only to extend knowledge but also to regularize generation [16], [17].

Implications for LLM-based BI assistants Although our experimental systems are deterministic and template-driven for reproducibility, the findings translate directly to LLM-based BI assistants. First, the prompt-only degradation in multi-turn settings mirrors what is often observed when an LLM is asked to generate SQL from only the current turn: the model tends to "forget" latent constraints unless they are repeated. Second, schema-aware retrieval resembles in-context learning with curated exemplars, which is known to improve structural fidelity in generation [18]–[20]. Third, the execution-feedback loop corresponds to "reasoning and acting" patterns, where a model alternates between proposing an action (SQL), observing the result (execution output/error), and revising its proposal [21], [22]. The empirical gains we measure from execution feedback in dialog success rates indicate that closing this loop is valuable even when the base generator already performs well.

Summary of findings. Across all benchmarks, we observe three consistent trends: (i) dialog-state grounding is necessary for multi-turn success; (ii) retrieval increases robustness, especially for single-turn compositional diversity; and (iii) execution feedback substantially reduces schema/syntax failures and yields the highest dialog-level success. These trends align with the evolution of text-to-SQL research from early sequence-to-sequence systems [4], [5], to schema-aware and relation-aware encoders [8], [9], and to execution- and constraint-aware decoding [6], [10], and they provide actionable guidance for building conversational BI assistants that must be both correct and dependable.

Repair effectiveness The controlled identifier-typo noise in Schema-RAG produces schema failures that are representative of common near-miss errors (e.g., "salar" instead of "salary"). On ProxySParC and ProxyCoSQL, most such failures occur on later turns because longer dialogs provide more opportunities for a single typo to appear. Exec-Feedback repairs the majority of these

failures, which explains why its dialog-level success improvement is larger than its turn-level improvement. Concretely, on ProxyCoSQL, Exec-Feedback increases dialog success from 78.8% to 87.5% while increasing turn-level execution accuracy from 91.9% to 94.4%. This gap reflects the multiplicative nature of dialog correctness: the probability that all turns succeed is roughly the product of turn-wise success probabilities when errors are independent, so even small per-turn gains can translate into larger dialog-level gains.

Sensitivity to dialog length. Figs. 6–7 show a consistent downward trend as the number of turns increases. For Schema-RAG and Exec-Feedback, the slope of this degradation is modest on ProxySParC because dialogs contain tightly controlled follow-ups, while the slope is steeper on ProxyCoSQL because dialogs include optional extra SQL turns and non-SQL turns that can interrupt the flow. These observations reinforce the practical value of designing conversational BI assistants with explicit state tracking and verification: longer interactions require mechanisms that prevent error accumulation, and execution feedback provides such a mechanism with minimal additional complexity.

Limitations

This study has several limitations that contextualize the conclusions.

First, the experimental suite uses format-preserving proxy benchmarks rather than the full official Spider/SParC/CoSQL releases. The proxies are constructed to be consistent with the original datasets' core properties—cross-domain SQLite schemas, explicit foreign keys, compositional SQL patterns, multi-turn anaphora, and clarification turns—but they do not reproduce the exact natural-language variability, schema idiosyncrasies, or long-tail SQL constructs found in human-authored data. Consequently, absolute accuracy numbers reported in this paper should be interpreted as empirical performance on the proxy benchmarks, not as direct replacements for leaderboard numbers on the official datasets [1]–[3].

Second, the evaluated systems are deterministic and template-driven for reproducibility. This design is intentional: it allows controlled isolation of retrieval, dialog grounding, and execution feedback, and it ensures that all reported results are reproducible by re-running the generator and evaluation scripts. However, it also means that the systems do not capture the full generalization behavior of modern neural parsers or LLM-based approaches. In particular, a trained neural model may exhibit different error trade-offs, such as higher semantic flexibility but also higher variance under paraphrasing. The main qualitative conclusions—context is required for multi-turn, retrieval improves grounding, and execution feedback improves robustness—remain applicable, but quantitative gains may differ under learned models.

Third, the execution-feedback loop in this paper focuses on repairing schema and syntax errors using identifier-level fuzzy matching. This choice targets a common and tractable class of failures that execution-time signals can detect reliably. Execution feedback can, in principle, be used more aggressively: for example, by reranking candidates based on denotational constraints, by detecting suspiciously empty result sets, or by asking clarifying questions when multiple candidates satisfy the prompt. These extensions are compatible with the EFRAG architecture (Fig. 1) but are not implemented here to keep the analysis focused and reproducible.

Fourth, the retrieval component uses simple token-overlap similarity and retrieves from a synthetic training pool. While this resembles sparse retrieval and captures the benefits of exemplar conditioning, it does not reflect the full spectrum of retrieval methods used in modern systems, including dense dual-encoders [17], learned schema retrievers, or hybrid retrieval. Evaluating alternative retrievers and quantifying retrieval quality (e.g., recall@K of the gold template) would strengthen the understanding of when and why retrieval helps.

Fifth, the clarification mechanism is rule-based and tied to a single required constraint slot per domain. Real conversational BI requires more flexible clarification behavior: multiple missing constraints can exist simultaneously, users may provide partial answers, and the system may need to negotiate the granularity of aggregation or the definition of "top" (e.g., top by revenue vs. top by count). Addressing these behaviors requires richer dialog policy learning and possibly user simulation, which are outside the scope of this paper.

Despite these limitations, the experiments provide a coherent and reproducible analysis of how schema-aware retrieval and execution feedback interact in dialog-based text-to-SQL. The proxy benchmarks serve as a controlled testbed for isolating these effects and can be used to validate algorithmic ideas before scaling to full natural-language datasets.

Conclusion

This paper studied conversational text-to-SQL for BI through the lens of a practical pipeline that combines retrieval augmentation and execution feedback. We introduced EFRAG, which maintains dialog state, retrieves schema snippets and exemplar queries, generates SQL conditioned on this evidence, executes the SQL, and repairs errors using execution feedback. To enable fully reproducible end-to-end evaluations, we constructed ProxySpider, ProxySParC, and ProxyCoSQL—format-preserving benchmarks with 30

SQLite databases across 10 domains and multi-turn dialog phenomena including anaphora and clarification.

Empirical results demonstrate that retrieval and dialog-state grounding are necessary for multi-turn success, while execution feedback provides an additional robustness layer that prevents small schema/syntax failures from collapsing whole dialogs. On ProxySParC and ProxyCoSQL, Exec-Feedback achieved the highest dialog-level success and maintained high accuracy as dialogs lengthened (Figs. 4–7). Error analyses showed that execution feedback reduces schema errors and shifts the remaining failure modes toward semantic and context challenges (Table VIII, Fig. 8), motivating future work on deeper intent modeling and clarification policies.

Overall, the findings support an engineering prescription for conversational BI: treat text-to-SQL as an interactive loop rather than a one-shot generation problem, combine schema-aware retrieval with persistent dialog grounding, and close the loop with execution-time feedback to repair and stabilize SQL generation.

References

[1] T. Yu, R. Zhang, K. Yasunaga, Y. Tan, X. Lin, S. Li, and D. Radev, “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task,” in Proc. EMNLP, 2018.

[2] T. Yu, M. Li, Z. Zhang, R. Zhang, and D. Radev, “SParC: Cross-Domain Semantic Parsing in Context,” in Proc. ACL, 2019.

[3] T. Yu, R. Zhang, M. Li, and D. Radev, “CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases,” in Proc. EMNLP-IJCNLP, 2019.

[4] V. Zhong, C. Xiong, and R. Socher, “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning,” arXiv:1709.00103, 2017.

[5] X. Xu, C. Liu, and D. Song, “SQLNet: Generating Structured Queries from Natural Language without Reinforcement Learning,” arXiv:1711.04436, 2017.

[6] B. Wang, R. Shin, X. Liu, and D. Radev, “Execution-Guided Neural Program Decoding,” in Proc. ACL, 2018.

[7] J. Guo, Z. Zhan, Y. Gao, J. Xiao, and T. Lou, “Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation,” in Proc. ACL, 2019.

[8] B. Bogin, M. Gardner, and J. Berant, “Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing,” in Proc. ACL, 2019.

[9] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, “RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers,” in Proc. ACL, 2020.

[10] T. Scholak, N. Schucher, and F. Hartmann, “PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models,” in Proc. EMNLP, 2021.

[11] O. Rubin and J. Berant, “SmBoP: Semi-autoregressive Bottom-Up Semantic Parsing,” in Proc. ACL, 2021.

[12] P. Dong and M. Lapata, “Language to Logical Form with Neural Attention,” in Proc. ACL, 2016.

[13] M. Jia and P. Liang, “Data Recombination for Neural Semantic Parsing,” in Proc. ACL, 2016.

[14] C. Raffel et al., “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” J. Mach. Learn. Res., vol. 21, no. 140, pp. 1–67, 2020.

[15] M. Lewis et al., “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension,” in Proc. ACL, 2020.

[16] P. Lewis, E. Perez, A. Piktus, et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in Proc. NeurIPS, 2020.

[17] V. Karpukhin, B. Oguz, S. Min, et al., “Dense Passage Retrieval for Open-Domain Question Answering,” in Proc. EMNLP, 2020.

[18] T. Brown et al., “Language Models are Few-Shot Learners,” in Proc. NeurIPS, 2020.

[19] M. Chen et al., “Evaluating Large Language Models Trained on Code,” arXiv:2107.03374, 2021.

[20] J. Wei et al., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” arXiv:2201.11903, 2022.

[21] S. Yao, J. Zhao, D. Yu, et al., “ReAct: Synergizing Reasoning and Acting in Language Models,” arXiv:2210.03629, 2022.

[22] P. Wang, J. He, and H. Chen, “Self-Consistency Improves Chain of Thought Reasoning in Language Models,” arXiv:2203.11171, 2022.

[23] Xinzhuo Sun, Yifei Lu, and Jing Chen, “Controllable Long-Term User Memory for Multi-Session Dialogue: Confidence-Gated Writing, Time-Aware Retrieval-Augmented Generation, and

Update/Forgetting", JACS, vol. 3, no. 8, pp. 9–24, Aug. 2023, doi: 10.69987/JACS.2023.30802.

[24] Hanqi Zhang, "DriftGuard: Multi-Signal Drift Early Warning and Safe Re-Training/Rollback for CTR/CVR Models", JACS, vol. 3, no. 7, pp. 24–40, Jul. 2023, doi: 10.69987/JACS.2023.30703.

[25] Meng-Ju Kuo, Boning Zhang, and Haozhe Wang, "Tokenized Flow-Statistics Encrypted Traffic Analysis: Comparative Evaluation of 1D-CNN, BiLSTM, and Transformer on ISCX VPN-nonVPN 2016 (A1+A2, 60 s)", JACS, vol. 3, no. 8, pp. 39–53, Aug. 2023, doi: 10.69987/JACS.2023.30804.

[26] Z. Zhong, M. Zheng, H. Mai, J. Zhao, and X. Liu, "Cancer image classification based on DenseNet model," Journal of Physics: Conference Series, vol. 1651, no. 1, p. 012143, 2020.