

TRACE: Table–Text Relationalization and Completion of Evidence for Coverage-Sensitive QA

Sunho Cha Sungho Park Wook-Shin Han*

Pohang University of Science and Technology (POSTECH)

Pohang, South Korea

{shcha, shpark, wshan}@dblab.postech.ac.kr

Abstract

Table–Text question answering asks systems to answer natural-language questions whose evidence is split between structured tables and linked unstructured passages. Many realistic questions in this setting are *coverage-sensitive*: the answer depends on a *set* of evidence items distributed across table rows and linked passages, and the same failure mode that appears in aggregation also surfaces in joins, multi-row filters, list questions, and universal or existential queries. Existing hybrid systems retrieve and read such evidence, yet leave passage-derived facts as strings in the LLM context rather than as *typed, joinable, and repairable relational values* that SQL can operate on. We propose TRACE (Table–Text Relationalization And Completion of Evidence), a completeness-aware query-processing framework built on two relational state transformations. *Schema-Guided Relationalization* (SGR) turns linked passages into typed, foreign-keyed side tables, exposing textual evidence as SQL-visible relational values; *Question-Conditioned Relational Completion* (QCRC) then repairs this state per question by adding missing attributes and filling incomplete cells. SQL performs set-oriented operations over the completed state, while a lightweight passage fallback handles residual gaps. On the coverage-sensitive SPARTA benchmark, TRACE more than triples the strongest baseline F1 (60.5 vs. 19.2 with gpt-4o-mini); on lookup-heavy HYBRIDQA, it stays competitive while substantially reducing the drop on questions that require many evidence items.¹

1 Introduction

Table–Text question answering asks systems to answer natural-language questions whose evidence is split between structured tables and linked unstructured passages (Chen et al., 2020, 2021a; Zhu et al.,

2021; Chen et al., 2021b; Zhao et al., 2022). Early work emphasized lookup-style reasoning, where a system can often succeed after retrieving one table row and one supporting passage. Many realistic questions, however, are *coverage-sensitive*: their answers depend on a *set* of evidence items distributed across table rows and linked passages—most visible in aggregations, but equally present in joins, multi-row filters, list questions, and universal or existential queries.

Figure 1 illustrates why this regime requires a different query-processing interface from retrieve-then-read QA: the source table enumerates candidate entities, but the predicates needed to answer the question live in the linked passages, so the system must expose many passage-derived values as typed, normalized, and joinable evidence before the final set-oriented operation can be trusted. This exposes two challenges. **Challenge 1: linked text is not relational.** SQL cannot join, filter, group, or aggregate over a fact stated only in a passage until it is represented as a typed cell aligned with the source table. **Challenge 2: one-shot extraction is incomplete.** A fixed offline pass cannot anticipate every future question, and useful side-table columns may still contain NULL for relevant rows.

This failure mode is precisely what the recent SPARTA benchmark (Park et al., 2026) is designed to surface. Built to address limitations of earlier Table–Text QA resources—shallow manually written questions, small table scale, limited analytical operations, and noisy annotations—SPARTA generates executable SQL-backed questions over source tables and grounding tables, covering joins, grouping, aggregation, and deeper multi-hop reasoning over hybrid evidence. As reported by Park et al. (2026), systems that perform strongly on lookup-heavy benchmarks such as HYBRIDQA (Chen et al., 2020) and OTT-QA (Chen et al., 2021a) suffer large F1 drops on SPARTA, confirming that high scores on earlier benchmarks can mask the

* Corresponding author.

¹Code: <https://github.com/carpref/TRACE>

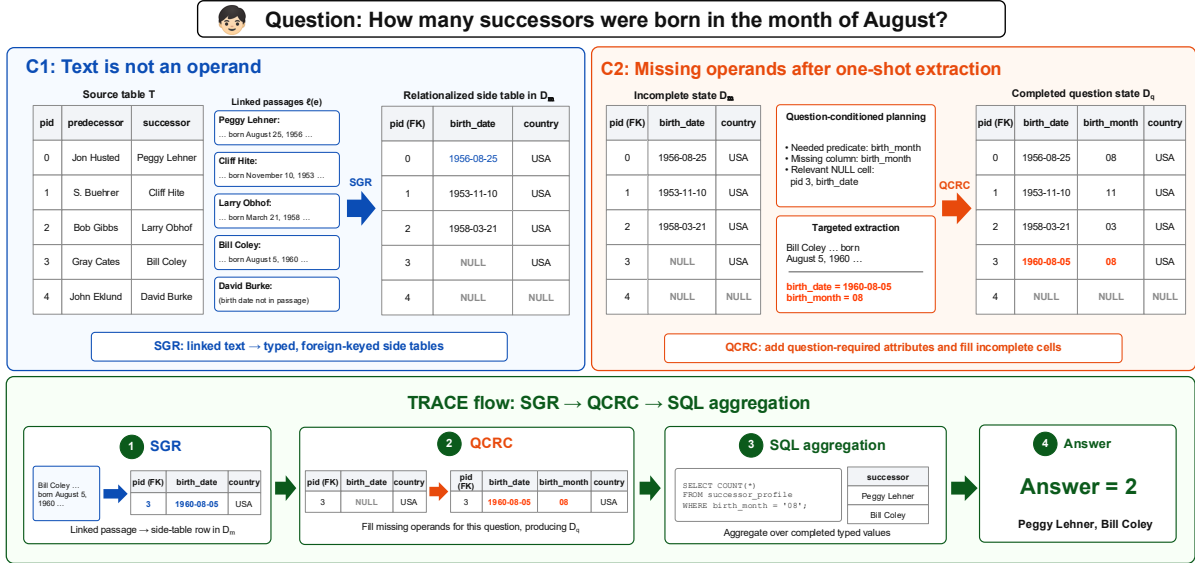


Figure 1: Why aggregation over hybrid table–text evidence requires relationalization and completion. The question asks for a count over table rows, but the counting predicate, birth month, lives only in each row’s linked passage. This creates two challenges. **C1: Textual operands are not relational.** The passage-derived values must first be converted into typed cells that SQL can filter and aggregate. **C2: One-shot extraction is incomplete.** Even after relationalization, missing columns and NULL values must be completed for the specific question. TRACE maps these challenges to SGR and QCRC, respectively.

coverage-sensitive failure mode. SPARTA is therefore not merely a more recent evaluation set in our experiments; it operationalizes the central difficulty studied in this paper—making the required passage-derived facts complete, normalized, and joinable for set-oriented reasoning.

Existing hybrid systems do not make passage-derived evidence relational enough for this regime (Park et al., 2025; Shi et al., 2024; Agarwal et al., 2025). Retrieval-oriented systems such as ODYSSEY (Agarwal et al., 2025) gather passages on demand and let the LLM reason over the resulting strings; program-synthesis systems such as HPROPRO (Shi et al., 2024) generate code over values extracted per query without persisting them. Both enrich the prompt, not the relational evidence state, so adding more retrieved text does not tell the system which source rows remain incomplete or whether the current evidence set is sufficient for a join, filter, list, group, or aggregate. We propose TRACE (Table–Text Relationalization And Completion of Evidence), which treats coverage-sensitive Table–Text QA as query processing over a *repairable relational state*. *Schema-Guided Relationalization* (SGR) runs once per domain corpus, using the source-table schema and entity–passage links to induce typed side-table schemas, extract and normalize passage facts, and attach each row

back to the source database through a foreign key (Challenge 1). *Question-Conditioned Relational Completion* (QCRC) runs per question, adding missing attributes and filling NULL cells from linked passages before SQL executes joins, filters, grouping, and aggregation over the completed state (Challenge 2).

On SPARTA, TRACE reaches 60.5 F1 with gpt-4o-mini versus 19.2 for the strongest baseline; the same gap holds with the open Qwen3.5-35B-A3B backbone. On lookup-heavy HYBRIDQA, TRACE remains competitive but does not uniformly dominate retrieval-style systems, which is expected because many questions require only one passage-level fact. TRACE is therefore most beneficial in the newer, more coverage-sensitive regime where the answer depends on many text-derived evidence items being jointly present, normalized, and joinable.

Our contributions are:

- **Problem diagnosis.** We identify relational evidence coverage as a central failure mode of coverage-sensitive Table–Text QA, spanning aggregations, joins, filters, list questions, and universal or existential queries.
- **Method.** SGR and QCRC—two complementary relational-state transformations that make

passage-derived facts SQL-operable and repairable before query execution.

- **Empirical evidence.** Large gains on the recent coverage-sensitive SPARTA benchmark, competitive behavior on lookup-heavy HYBRIDQA, and ablations indicating that both relationalization and completion contribute to final performance.

2 Related Work

Related work is organized around the representation of evidence split between tables and text. While prior systems often keep text-derived facts outside the relational state, TRACE relationalizes linked passages into typed side tables and completes this evidence state before SQL execution.

2.1 Table–Text Question Answering

Table–Text QA studies questions whose evidence spans structured tables and linked passages (HYBRIDQA (Chen et al., 2020), OTT-QA (Chen et al., 2021a), SPARTA (Park et al., 2026)). Prior systems either improve how models traverse and reason over this hybrid evidence (HPROPRO (Shi et al., 2024), ODYSSEY (Agarwal et al., 2025)) or address the complementary problem of retrieving relevant tables and passages (TABLERAG (Yu et al., 2025), HELIOS (Park et al., 2025)); standalone table-reasoning and adaptive prompting systems (Rajgaria et al., 2025) attack a different bottleneck. What these methods share is that text-derived facts remain raw strings inside the LLM context rather than entering a persistent relational evidence state, even though the required predicates and attributes often live in linked passages rather than the table itself.

TRACE instead asks how already-linked evidence should be represented and completed for set-oriented query processing. SGR stores passage-derived facts as typed cells joined back to source rows via foreign keys, turning linked text into relational evidence before reasoning begins. This structural shift lets SQL reliably execute joins, filters, grouping, and aggregation over a single, completed relational state.

2.2 Text-to-SQL and Database-Grounded QA

Text-to-SQL maps natural-language questions to executable queries over a fixed relational schema, with strong results on Spider (Yu et al., 2018) and BIRD (Li et al., 2023) via schema linking,

execution-guided decoding (Wang et al., 2018), and database-aware prompting (Hong et al., 2025). The lesson is consistent: once evidence is represented as typed relational values, SQL executes complex operations more reliably than an LLM reasoning over strings. However, standard Text-to-SQL assumes all required values already exist in the database.

TRACE targets the regime where this assumption breaks down: coverage-sensitive QA whose key attributes live only in linked passages. Rather than optimizing SQL generation itself, our contribution is to *construct and complete* the relational data from text before SQL is generated—introducing text-derived side tables and repairing missing attributes via QCRC, so the SQL generator operates over a comprehensive schema that merges structured data with relationalized textual evidence.

2.3 Text-to-Table and Information Extraction

Text-to-table and information extraction systems convert unstructured text into structured records (Wu et al., 2022; Pietruszka et al., 2024; Pai et al., 2024) or synthesize queryable databases from documents (SQUID (Sadia et al., 2025)), showing that LLMs can induce useful tables from text. Their output, however, is typically an isolated extraction product or a corpus-level database rather than a repairable query-processing state aligned with an existing source table.

TRACE enforces this alignment because a passage-derived value is useful only if it remains tied to the source entity that made the passage relevant: SGR makes the alignment explicit via foreign keys, and QCRC replaces the static extraction lifecycle with a query-conditioned completion step that adds missing attributes and fills incomplete cells per question.

2.4 Agentic Reasoning and Iterative Retrieval

Agentic reasoning and iterative retrieval methods (REACT (Yao et al., 2023), TOOLFORMER (Schick et al., 2023), IRCOT (Trivedi et al., 2023), SELF-RAG (Asai et al., 2024), FLARE (Jiang et al., 2023)) interleave LLM reasoning with external actions such as tool use and retrieval. Effective for open-ended reasoning, these designs append retrieved evidence to the prompt as raw text—a prompt-centric stance that does not address relational evidence coverage, since more text cannot track which rows remain incomplete or whether the current evidence set is sufficient for set-oriented

operations.

TRACE instead adds missing evidence to the database state itself: QCRC repairs the schema online and reserves passage retrieval for residual gaps outside the completed relational state, shifting coverage-sensitive operations from string-level LLM reasoning into explicit relational query processing over typed values.

3 Problem Setup and Preliminaries

This section defines the relational state that TRACE constructs and completes. We separate the hybrid corpus into source tables, linked passages, and text-derived side tables, isolating the core bottleneck: coverage-sensitive questions fail when passage-derived evidence is not visible as typed, source-aligned relational values before SQL execution.

Hybrid Table–Text corpus. A hybrid Table–Text corpus is $\mathcal{C} = (\mathcal{T}, \mathcal{P}, \mathcal{L})$, where $\mathcal{T}$ is a set of source tables, $\mathcal{P}$ is a set of passages, and $\mathcal{L}$ contains entity–passage links. We further define a set of evidence anchors $\mathcal{E}$, where each $e \in \mathcal{E}$ identifies a source-table component (a cell or a row) and $\ell(e) \subseteq \mathcal{P}$ denotes its linked passages.

Coverage-sensitive question. A question q is *coverage-sensitive* if its answer depends on a set of evidence items distributed across source rows and linked passages—e.g., joins, multi-row filters, list questions, aggregations, and universal or existential queries; we call its complement *lookup*, where a single evidence item suffices.

Text-derived side tables. We extend the corpus with side tables $\mathcal{M} = \{M_1, \dots, M_K\}$. Each M_k has a typed schema $\mathcal{S}(M_k) = \{\text{fk}_k\} \cup \mathcal{A}_k$, separating the foreign key fk_k from extracted attributes $\mathcal{A}_k$. For every row $r \in M_k$, $\text{fk}_k(r) \in \mathcal{E}$ points back to the source anchor.

Relationalized database. The offline output of SGR is $\mathcal{D}_m = \mathcal{T} \cup \mathcal{M}$, built before any question is known. Because $\mathcal{D}_m$ is best-effort, it may leave some side-table cells as NULL.

Question-completed database. The online output of QCRC for a question q is $\mathcal{D}_q = \mathcal{T} \cup \mathcal{M}_q$, where $\mathcal{M}_q = \{M_1^{(q)}, \dots, M_K^{(q)}\}$. QCRC expands the schema or fills NULL cells from linked passages as required by q , and SQL is executed over $\mathcal{D}_q$.

Closed-domain setting. We evaluate in the closed-domain setting: each question q is bound

to a fixed corpus $\mathcal{C}(q)$, isolating coverage-sensitive reasoning from open-domain retrieval errors.

Relation to HYBRIDQA and SPARTA. HYBRIDQA (Chen et al., 2020) instantiates the single-table case (lookup-heavy), while SPARTA (Park et al., 2026) instantiates the domain-database case requiring set-oriented reasoning over multiple source and side-table values.

Scope of the task. TRACE targets *analytical Table–Text QA*, where passage-derived values must be reused across SQL operations such as aggregations, joins, and multi-row filters; standard retrieve-then-read pipelines remain sufficient for one-off factoid queries.

4 Method

TRACE answers coverage-sensitive Table–Text questions by converting linked textual evidence into a repairable relational state before SQL execution. The method consists of two stages: SGR constructs the relationalized database $\mathcal{D}_m = \mathcal{T} \cup \mathcal{M}$ offline once per domain corpus, and QCRC dynamically repairs it into the question-completed database $\mathcal{D}_q = \mathcal{T} \cup \mathcal{M}_q$ online. The final SQL agent operates explicitly over $\mathcal{D}_q$. Prompt templates for every step are listed in Appendix A.1 and indexed by step in Table 5.

4.1 Schema-Guided Relationalization (SGR)

Given a hybrid corpus $\mathcal{C} = (\mathcal{T}, \mathcal{P}, \mathcal{L})$, SGR induces text-derived side tables $\mathcal{M} = \{M_1, \dots, M_K\}$ to assemble the offline relationalized database $\mathcal{D}_m = \mathcal{T} \cup \mathcal{M}$. This stage runs once per corpus and caches passage embeddings $\phi(p)$ to accelerate online completion.

(i) Schema-guided passage grouping. To prevent unrelated facts from collapsing into a flat schema, TRACE partitions linked source attributes into groups $\{g_1, \dots, g_K\}$ by their target entity types. Each group g_k induces a coherent side table M_k bounded to a specific subset of source anchors $\mathcal{E}_k \subseteq \mathcal{E}$.

(ii) Side-table schema induction. For each g_k , TRACE prompts an LLM to propose an extraction guide π_k and a typed attribute set $\mathcal{A}_k$. The explicit schema strictly separates these new attributes from the source alignment:

$$\mathcal{S}(M_k) = \{\text{fk}_k\} \cup \mathcal{A}_k.$$

(iii) Attribute-value extraction. SGR populates the tables by applying π_k to linked texts. For every anchor $e \in \mathcal{E}_k$ and passage $p \in \ell(e)$, the extracted row $r \in M_k$ must strictly satisfy: $\text{fk}_k(r) = e$. This constraint guarantees the row is relationally joined back to the database, while the remaining cells are populated with values for $\mathcal{A}_k$ or NULL.

(iv) Normalization and database assembly. To guarantee SQL compatibility, raw extractions are normalized into canonical types (e.g., dates, numbers) and type-violating outliers are repaired. The resulting tables merge with $\mathcal{T}$ to finalize $\mathcal{D}_m$.

4.2 Question-Conditioned Relational Completion (QCRC)

QCRC formally repairs $\mathcal{D}_m$ into $\mathcal{D}_q$ for the current question q . Rather than expanding the LLM context window, it directly modifies the database schema and fills incomplete cells while strictly preserving foreign-key alignments. This process produces the question-completed state:

$$\mathcal{M}_q = \{M_1^{(q)}, \dots, M_K^{(q)}\}, \quad \mathcal{D}_q = \mathcal{T} \cup \mathcal{M}_q$$

(i) Question-conditioned planning. An LLM planner identifies necessary side-table attributes for q . For each M_k , it selects existing attributes $\mathcal{A}_{q,k}^{\text{exist}} \subseteq \mathcal{A}_k$ and proposes new ones $\mathcal{A}_{q,k}^{\text{new}}$, which are appended as initially NULL columns. The system then defines the global candidate cells requiring repair: $\mathcal{C}_q = \bigcup_k \{(r, A) \mid r \in M_k^{(q)}, A \in \mathcal{A}_{q,k}^{\text{exist}} \cup \mathcal{A}_{q,k}^{\text{new}}, M_k^{(q)}(r, A) = \text{NULL}\}$

(ii) Embedding-similarity gate. To minimize passage reads, a similarity gate filters candidates in $\mathcal{C}_q$. Let $e = \text{fk}_k(r)$. Using a textual query $u_{r,A}$ derived from the anchor and attribute descriptions, a cell is retained only if its linked passages surpass a similarity threshold τ : $\max_{p \in \ell(e)} \cos(\phi(u_{r,A}), \phi(p)) > \tau$. Candidates failing this gate remain NULL, explicitly exposing residual incompleteness to the database state.

(iii) Targeted extraction. For surviving candidates, TRACE executes a single-attribute extraction prompt on the gated passages. Returned values overwrite the NULL cells in $M_k^{(q)}(r, A)$ while strictly preserving $\text{fk}_k(r) = e$. This dynamically completes the relational evidence state without re-running full extractions.

(iv) SQL execution over the completed state. The agent executes SQL exactly

over $\mathcal{D}_q$. At turn t , observing history $h_{<t} = \{(\text{THOUGHT}_i, \text{SQL}_i, \text{OBS}_i)\}_{i < t}$ and a passage buffer $\mathcal{B}_{<t}$, the agent emits SQL. When queries return rows, TRACE appends the linked passages of returned anchors into $\mathcal{B}_{<t}$, allowing textual inspection for residual interpretation while keeping set-oriented operations safely in SQL.

(v) Residual evidence fallback. If a SQL query yields no rows, TRACE falls back to lightweight dense retrieval over $\mathcal{P}$, appending the top 3 passages to the passage buffer. This ensures that residual evidence completely outside $\mathcal{D}_q$ remains accessible without convoluting the core relational reasoning.

Takeaway. SGR and QCRC play complementary roles: the former offline-constructs a baseline relational state ($\mathcal{D}_m$), and the latter online-repairs it for the specific question ($\mathcal{D}_q$). Together, they fundamentally shift coverage-sensitive QA from string-level reasoning over opaque text to robust set-oriented SQL execution over a completed relational state.

5 Experiments

The experiments evaluate whether converting linked text into a completed relational state improves coverage-sensitive Table-Text QA. We conduct evaluations in the closed-domain setting (§3) to isolate reasoning over a given hybrid corpus from open-domain retrieval, focusing on three questions: (1) Does TRACE improve performance on questions requiring extensive evidence coverage? (2) Do SGR and QCRC both contribute to the final accuracy? (3) What is the cost trade-off of shifting evidence repair from the prompt context into the relational database state?

5.1 Experimental Setup

Datasets. We evaluate on two datasets to stress different extremes of relational evidence coverage: SPARTA (Park et al., 2026) is the primary coverage-sensitive benchmark requiring complex SQL operations (joins, grouping, aggregation) over multiple tables and passages. We use its validation split (565 questions per domain: NBA, Medical, Movie). Conversely, HYBRIDQA (Chen et al., 2020) is a lookup-heavy complement where single passage-level facts often suffice, evaluated on its 3,466-question development split. This pairing tests our core claim: TRACE excels when text-derived val-

Method	Sparta-NBA					Sparta-Medical					Sparta-Movie					Sparta-Avg				
	EM	F1	P	R	LJ	EM	F1	P	R	LJ	EM	F1	P	R	LJ	EM	F1	P	R	LJ
GPT-4o-mini																				
HProPro	3.02	3.69	4.57	3.67	5.84	19.12	29.54	29.05	30.81	24.42	7.79	10.67	10.69	11.66	9.56	9.98	14.63	14.77	15.38	13.27
ODYSSEY	6.24	13.53	26.16	13.75	9.09	6.73	22.48	33.57	21.30	8.50	17.20	21.45	30.07	22.35	18.97	10.06	19.15	29.93	19.13	12.19
TRACE	43.89	51.58	53.08	53.91	49.03	60.88	72.45	74.78	73.35	69.56	47.43	57.47	60.82	58.71	50.62	50.73	60.50	62.89	61.99	56.40
Qwen3.5-35B-A3B																				
HProPro	3.01	3.35	3.87	3.28	4.78	23.01	34.92	34.89	35.88	29.73	8.67	11.48	11.66	12.88	9.91	11.56	16.58	16.81	17.35	14.81
ODYSSEY	6.37	15.24	21.70	17.89	8.85	9.56	24.45	31.34	26.31	9.91	18.23	21.46	26.72	22.78	19.65	11.39	20.38	26.59	22.33	12.80
TRACE	42.12	47.76	49.04	49.69	49.20	58.94	73.86	75.20	74.11	76.46	43.89	53.23	54.78	53.67	48.85	48.32	58.28	59.67	59.16	58.17

Table 1: Results on the three SPARTA domains ($n=565$ each), grouped by backbone LLM. **Sparta-Avg** averages over the three domains. **P/R**: token-level precision/recall. **LJ**: gpt-4o LLM-AS-JUDGE accuracy under a binary rubric (Appendix A.2). Best per column within each LLM in **bold**.

Method	HybridQA				
	EM	F1	P	R	LJ
<i>GPT-4o-mini</i>					
HProPro	36.30 $\pm$ 1.60	42.64 $\pm$ 1.57	43.63 $\pm$ 1.60	43.63 $\pm$ 1.61	45.82 $\pm$ 1.66
ODYSSEY	44.84 $\pm$ 1.66	55.59 $\pm$ 1.50	56.30 $\pm$ 1.52	57.32 $\pm$ 1.54	62.32 $\pm$ 1.61
TRACE	44.32 $\pm$ 1.66	53.78 $\pm$ 1.53	54.12 $\pm$ 1.56	56.25 $\pm$ 1.57	64.97 $\pm$ 1.58
<i>Qwen3.5-35B-A3B</i>					
HProPro	43.02 $\pm$ 1.65	51.11 $\pm$ 1.57	51.37 $\pm$ 1.58	53.10 $\pm$ 1.61	53.95 $\pm$ 1.66
ODYSSEY	56.26 $\pm$ 1.65	67.64 $\pm$ 1.40	68.29 $\pm$ 1.43	69.99 $\pm$ 1.42	72.99 $\pm$ 1.48
TRACE	52.05 $\pm$ 1.67	62.05 $\pm$ 1.48	62.72 $\pm$ 1.50	66.65 $\pm$ 1.47	72.76 $\pm$ 1.48

Table 2: Results on HYBRIDQA dev ($n=3,466$), grouped by backbone LLM. **P/R**: token-level precision/recall. **LJ**: gpt-4o LLM-AS-JUDGE accuracy under a binary rubric (Appendix A.2). Subscripts are question-level 95% CI half-widths: Wilson score interval for the binary metrics (EM, LJ) and a percentile bootstrap over 20,000 question-level resamples for F1/P/R. Best per column within each LLM in **bold**.

ues must be relationally joinable, while remaining competitive when simple retrieval is sufficient.

Baselines. The baselines represent state-of-the-art alternatives that operate over hybrid evidence without a repairable relational state: HPROPRO (Shi et al., 2024) executes generated programs over query-time extractions rather than maintaining persistent side tables. ODYSSEY (Agarwal et al., 2025) conducts entity-oriented reasoning over retrieved tables and passages. Following standard protocols, we adapt baselines to SPARTA by providing necessary domain tables and join links.

Models. To ensure generalizability across model families, we evaluate one closed LLM (gpt-4o-mini via OpenAI API) and one open LLM (Qwen3.5-35B-A3B via vLLM (Kwon et al., 2023)). All components use temperature 0 to isolate the effectiveness of query-processing strategies over decoding diversity.

Implementation details. TRACE unifies the source database and side tables within a single

SQLite instance. For residual fallback, we retrieve the top-3 passages via BGE-M3 (Chen et al., 2024). Open-model inference runs on two 48GB NVIDIA RTX A6000 GPUs. Additional implementation details are provided in Appendix A.4.

Metrics. We report exact match (EM) and token-level F1. To account for valid surface-form variations caused by normalization, we supplement these with LLM-AS-JUDGE (LJ) accuracy. Using an independent gpt-4o (temperature 0), LJ serves strictly as a semantic-equivalence complement to EM/F1, not a replacement.

5.2 Main Results

Observation 1: TRACE yields the largest gains on coverage-sensitive QA. TRACE substantially outperforms baselines on all SPARTA domains across both LLM backbones. With gpt-4o-mini, TRACE achieves 60.50 average F1, substantially outperforming ODYSSEY (19.15) and HPROPRO (14.63). This margin holds with Qwen3.5-35B-A3B (58.28 vs. ODYSSEY’s 20.38). Detailed domain-wise breakdowns (EM, F1, preci-

Variant	NBA	Medical	Movie	Avg	Δ Avg
TRACE (full)	49.20	76.46	48.85	58.17	—
w/o online enrich.	45.84	69.56	41.59	52.33	-5.84
w/o offline mat.	46.02	49.73	32.74	42.83	-15.34

Table 3: Component ablation on SPARTA domains with Qwen3.5-35B-A3B, measured by LLM-AS-JUDGE accuracy (§5.1).

sion, recall, LJ) are in Table 1. These results validate our core mechanism: when questions demand extensive relational evidence, materializing and repairing side tables pre-SQL is far more reliable than reasoning over retrieved strings or query-time extractions. To assess robustness, we further evaluate TRACE under missing and incorrect entity–passage links; Appendix B reports the full study.

Observation 2: TRACE remains competitive on lookup-heavy QA, though it is not its primary target. On HYBRIDQA, retrieval-oriented baselines edge TRACE on F1 (ODYSSEY leads by up to ~ 6 points across backbones), but LLM-AS-JUDGE scores show semantic parity: TRACE matches or surpasses ODYSSEY under both backbones. Their question-level 95% LJ confidence intervals overlap in both cases (Table 2), so we characterize TRACE as semantically competitive on this benchmark rather than superior. TRACE thus does not aim to replace retrieve-then-read pipelines for single-fact lookups, but targets scenarios requiring extensive relational evidence coverage across multiple linked passages.

5.3 Coverage-Sensitive Subset Analysis

Observation 3: The performance gap widens on coverage-sensitive questions. We partition each benchmark into *coverage-sensitive* subsets (in the sense of §3)—SPARTA queries whose gold SQL contains coverage-sensitive operators (e.g., aggregations) and the COMPUTED subset in HYBRIDQA (Chen et al., 2020)—versus their *lookup* counterparts. As Figure 2 shows, the strongest baselines suffer sharp LLM-AS-JUDGE (LJ) drops on coverage-sensitive subsets, while TRACE retains substantially higher LJ across all domains. This confirms the bottleneck is not retrieving a single passage but materializing the full set of text-derived values required for SQL execution.

5.4 Component Analysis

Observation 4: SGR drives the primary gains, while QCRC ensures dynamic completeness.

Method	HybridQA	SPARTA		
		NBA	Medical	Movie
HProPro	6,297	148,722	56,891	143,964
ODYSSEY	6,412	130,906	15,843	89,319
TRACE — online stages:				
extraction	2,776	7,317	2,340	1,648
execution	10,128	33,105	7,840	6,448
total	12,904	40,422	10,180	8,096

Table 4: Per-query *online* cost in tokens (input + output) under the closed-domain setting with gpt-4o-mini. For TRACE, costs split into the planner with targeted extractions and the iterative SQL execution.

Ablations (Table 3) confirm both stages are essential. Removing QCRC (relying solely on the offline $\mathcal{D}_m$) drops average SPARTA LJ by 5.84 points, while removing SGR (relying purely on online extraction) causes a far steeper 15.34-point drop. The asymmetry identifies the offline relational state as the core driver of robustness, and the QCRC penalty shows that even a strong static state cannot anticipate every query—together resolving the dual challenges in Figure 1. Where the ablations decompose the gains, Appendix A.3 decomposes what remains: gold grounding tables bound how much of the residual error stems from state construction, and a manual audit of wrong non-empty results identifies omission—not wrong values—as the dominant extraction failure.

5.5 Efficiency and Amortization

Observation 5: TRACE reduces online token costs on SPARTA through offline relationalization. Because QCRC selectively repairs only question-relevant cells and delegates reasoning to SQL, TRACE strictly bounds query-time token consumption. With gpt-4o-mini, TRACE uses 1.6–11 $\times$ fewer online tokens per query than ODYSSEY across the three SPARTA domains, since the baselines repeatedly read retrieved text (Table 4). This isolates marginal online costs (post-question arrival), confirming the efficiency of shifting heavy evidence-extraction to a *reusable* offline state. On HYBRIDQA, where one or two passages usually suffice, there is little retrieved text to save on, and TRACE’s fixed planning and SQL-execution cost makes it spend about twice the online tokens of either baseline.

Observation 6: Upfront relationalization cost is amortized over repeated workloads. Because SGR is a one-time per-domain pass, its cost amortizes across queries: over 565 dev queries, the

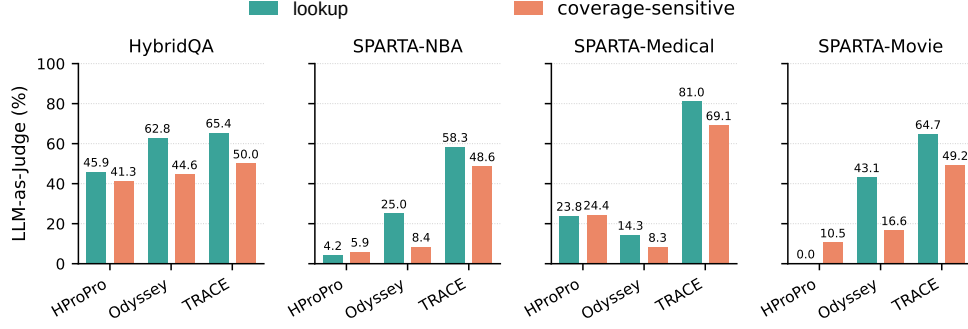


Figure 2: Per-subset LLM-AS-JUDGE accuracy on HYBRIDQA and the three SPARTA domains (gpt-4o-mini). Each panel shows three systems; teal = lookup subset, coral = coverage-sensitive subset.

Case (A) SPARTA-MEDICAL.

At Central Hospital, what is the average years of experience among doctors whose specialization does not start with 'Delmato'? **Gold: 25.75**

ODYSSEY

Start Point: doc_entity hop matches 'Central Hospital' on D004.

Hop 1: no AVG primitive $\Rightarrow$ returns matched cell's value as-is.

Pred: 28

HProPro

Generated Code:

```
spec = extract_info(row['doctor_name'][1], 'specialization')
if spec and not spec.startswith('Delmato'):
    exp = extract_info(row['doctor_name'][1], 'years of experience')
```

Pred: 21.5

TRACE

Offline materialized table: doctors

name	specialization	workplace	experience_years
David Jones	Pediatrics	Central Hospital	28
⋮	⋮	⋮	⋮

Online enrich: no new columns.

SQL Turn 1:

```
SELECT AVG(experience_years) FROM doctors
WHERE workplace='Central Hospital'
AND specialization NOT LIKE 'Delmato%'
```

Pred: 25.75

Case (B) HYBRIDQA.

How many athletes were born in the month of June? **Gold: 2**

ODYSSEY

Start Points: 4 doc_entity hops match 'June' span across athlete passages.

Hop 1: no COUNT primitive $\Rightarrow$ returns name list as-is.

Pred: Karin Balzer, Ginka Zagorcheva, Lucyna Kałek, Anneliese Ehrhardt

HProPro

Generated Code:

```
june_count = 0
for row in table.iterrows():
    birth_date = extract_info(row['Athlete'][1], 'birth date')
    if 'June' in birth_date: june_count += 1
```

Pred: 4

TRACE

Offline materialized table: athlete

name	birth_date	specialty
Karin Balzer	1938-06-05	100 m hurdles
⋮	⋮	⋮

Online enrich: no new columns.

SQL Turn 1:

```
SELECT COUNT(*) FROM athlete
WHERE strftime('%m', birth_date) = '06'
```

Pred: 2

Figure 3: Case studies with intermediate traces across compared methods: (A) SPARTA-MEDICAL, (B) HYBRIDQA.

overhead shrinks to ~ 48 tokens/query on SPARTA-MEDICAL, $\sim 13K$ on SPARTA-MOVIE, and $\sim 115K$ on SPARTA-NBA, decreasing linearly with more queries. While retrieve-then-read remains cheaper for one-off factoids, TRACE replaces recurring query-time extraction with a single reusable relational state, making it best suited for sustained, coverage-sensitive workloads. Break-even arrives within a handful of queries on Medical and grows with offline cost on Movie and NBA, while lookup-heavy HYBRIDQA admits no finite break-even (Appendix C).

5.6 Case Studies

Observation 7: Relational evidence prevents silent coverage failures. Figure 3 traces two representative cases—(A) SPARTA-MEDICAL and (B) HYBRIDQA—that illustrate the mechanisms behind the quantitative gains. In both cases, a retrieval-oriented baseline (ODYSSEY) finds evi-

dence for one entity and returns its local value as if it represented the full set—a failure of evidence coverage, not arithmetic. A program-based baseline (HProPro) checks a string extracted from a passage and counts irrelevant mentions because the value was never normalized as a typed cell. TRACE avoids both failures once the relevant attributes are relationalized: source-table rows join to side-table values through fk_k , and SQL performs the set-oriented operation over typed values. Coverage-sensitive reasoning becomes less brittle when the evidence set is explicit, source-aligned, and repairable before the final answer is produced.

Takeaway. The experiments support the paper’s central claim: TRACE is most useful when a question requires many text-derived evidence items to be present, normalized, and source-aligned. SGR provides a reusable relational evidence base, QCRC repairs that base per question, and SQL exe-

cutes the final set-oriented reasoning over the completed state. The remaining weaknesses—lookup-heavy questions, one-off workloads, and wrong non-empty results—are consistent with this scope rather than contradictions of the method.

6 Conclusion

This paper addresses the challenge of coverage-sensitive Table–Text QA, where answering complex questions (e.g., aggregations, joins, and multi-row filters) requires the joint presence, normalization, and source-alignment of multiple passage-derived evidence items. To overcome the limitation of existing models where linked text facts cannot be reused as typed values in set-oriented operations, we propose TRACE, a framework that reframes hybrid evidence coverage as a query-processing problem over a repairable relational state. Specifically, TRACE constructs a reusable relationalized database $\mathcal{D}_m$ via typed, foreign-keyed side tables, and dynamically transforms it into a question-completed state $\mathcal{D}_q$ by filling missing attributes on demand, allowing SQL to execute over $\mathcal{D}_q$ while minimizing residual passage retrieval. Extensive experiments demonstrate that TRACE substantially outperforms baselines on coverage-heavy workloads across both closed and open backbones, while remaining highly competitive on lookup-heavy tasks. Ultimately, this work shifts the paradigm for complex hybrid QA: when a workload demands the reuse and aggregation of scattered evidence, the optimal solution is not merely enhanced retrieval, but an explicit, inspectable relational state that elevates hidden passage facts into structured query-processing objects.

Limitations

Extraction quality bounds the completed relational state. TRACE makes text-derived evidence explicit and repairable, but it does not guarantee that LLM extraction is complete or correct. SGR may miss useful attributes during side-table schema induction, extract wrong values from linked passages, normalize units inconsistently, or assign a value to an overly broad attribute. QCRC can repair missing attributes and visible NULL cells, but it cannot fully recover from systematic extraction errors that are already present in $\mathcal{D}_m$. As a result, SQL is exact only with respect to the current question-completed database $\mathcal{D}_q$; if $\mathcal{D}_q$ contains biased or incorrect text-derived values, the final

join result, filtered set, or aggregate may also be biased. Table 7 bounds how large this effect is on each domain.

Wrong non-empty results are harder to detect than missing evidence. QCRC is naturally suited to repairing visible incompleteness: a missing column can be added, and a NULL cell can be filled from linked passages. A harder case occurs when SQL returns a plausible non-empty result that is nevertheless incomplete. For example, a count of 5 may be returned when the correct count is 7 because two passage-derived values were never extracted, or a filtered list may omit entities whose linked evidence remained below the gate threshold. Since this failure does not necessarily trigger the empty-result fallback, future work should add explicit evidence-coverage estimation, aggregate-level consistency checks, or uncertainty signals over completed side-table cells. Appendix A.3 audits 164 such cases and finds omission, not wrong values, to be the dominant cause.

The current system assumes source-table alignment is available. TRACE relies on entity–passage links and source evidence anchors $\mathcal{E}$ to align side-table rows with the source database through foreign keys fk_k . This assumption matches established Table–Text QA benchmarks, where cells or rows are linked to passages, but it may not hold in noisier corpora. If entity links are missing, ambiguous, or wrong, SGR may attach an extracted value to the wrong source anchor, making the value joinable but semantically incorrect. A fully robust system would need joint entity linking, side-table construction, and confidence-aware source alignment. Appendix B quantifies how far performance degrades as links are removed or redirected.

Upfront relationalization cost favors repeated workloads. SGR performs a one-time extraction pass over linked passages, so its cost can be substantial for large corpora or frequently changing data. This design is most appropriate for stable or slowly updated Table–Text corpora that receive many coverage-sensitive questions, where the relationalized database $\mathcal{D}_m$ can be reused. For one-off lookup questions, a retrieve-then-read pipeline may be cheaper because it avoids building side tables that may never be reused. Incremental maintenance of $\mathcal{D}_m$ under document updates is therefore an important next step.

Coverage-sensitive QA is broader than the benchmarks fully measure. SPARTA stresses coverage-sensitive reasoning through domain databases, joins, filters, and aggregation, while HYBRIDQA provides a lookup-heavy complement. However, the benchmarks do not exhaust the full space of coverage-sensitive Table–Text QA. Universal queries, existential queries, long list construction, temporal joins, and adversarially incomplete linked passages deserve more targeted evaluation. Our results therefore support the relational evidence coverage hypothesis in the evaluated settings, but broader benchmarks are needed to test how far the conclusion generalizes.

LLM-as-Judge is only a complementary metric. We report LLM-AS-JUDGE because normalized relational answers may differ from reference surface forms, especially for dates, entity aliases, and numeric expressions. However, judge models can be sensitive to wording and may make mistakes on numerical equivalence or set equivalence. For this reason, we treat judge accuracy as a complement to EM and F1 rather than as a replacement. Appendix A.2 validates the judge against manual annotation of all open-model SPARTA outputs; extending this validation to closed-model outputs and other datasets remains future work.

Ethical Considerations

Data and benchmarks. We evaluate on HYBRIDQA (Chen et al., 2020) and SPARTA (Park et al., 2026), both publicly released for research use. HYBRIDQA is derived from Wikipedia content; SPARTA is synthetically generated over public source tables. No personally identifying information beyond what already appears in these public sources is introduced by our pipeline. We use these benchmarks within the scope of their original licenses.

Model use and reproducibility. TRACE relies on closed (gpt-4o-mini, gpt-4o as judge) and open (Qwen3.5-35B-A3B) large language models. Closed-model behavior may change over time due to provider-side updates; we report the API model versions used in our experiments to support faithful reproduction, and release prompts and code through the repository linked in the abstract. All experiments use temperature 0 to reduce non-determinism.

Risks of LLM-based extraction and judging.

Two stages in TRACE are LLM-driven and may inherit model biases: (i) SGR and QCRC extract typed values from linked passages, which can amplify factual or demographic biases already present in the underlying text, and (ii) LLM-AS-JUDGE may be sensitive to surface form. We mitigate (i) by exposing residual incompleteness as NULL cells rather than fabricating values (§4.2), and (ii) by reporting EM/F1 alongside judge accuracy. We do not claim that the extracted relational state is a substitute for verified knowledge.

Intended use. TRACE is intended as a research framework for coverage-sensitive Table–Text QA over stable or slowly updated hybrid corpora. It is not designed for high-stakes decision-making (medical, legal, financial advice) without expert verification of the completed relational state.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. RS-2024-00509258, Global AI Frontier Lab, 35%, No. RS-2024-00454666, Developing a Vector DB for Long-Term Memory Storage of Hyperscale AI Models, 5%) and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS. 2025-00517736, 60%).

References

- Ankush Agarwal, Chaitanya Devaguptapu, and Ganesh S. 2025. [Hybrid graphs for table-and-text based question answering using LLMs](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 858–875, Albuquerque, New Mexico. Association for Computational Linguistics.
- Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. [Self-RAG: Learning to retrieve, generate, and critique through self-reflection](#). In *The Twelfth International Conference on Learning Representations*.
- Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. [M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

- Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W. Cohen. 2021a. [Open question answering over tables and text](#). In *International Conference on Learning Representations*.
- Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. [HybridQA: A dataset of multi-hop question answering over tabular and textual data](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.
- Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021b. [FinQA: A dataset of numerical reasoning over financial data](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. [Next-generation database interfaces: A survey of LLM-based text-to-SQL](#). *IEEE Transactions on Knowledge and Data Engineering*, 37(12):7328–7345.
- Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [Active retrieval augmented generation](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore. Association for Computational Linguistics.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient memory management for large language model serving with pagedattention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23*, page 611–626, New York, NY, USA. Association for Computing Machinery.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. [Can LLM already serve as a database interface? A BIG Bench for large-scale database grounded text-to-SQLs](#). In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Datasets and Benchmarks Track*.
- Liu Pai, Wenyang Gao, Wenjie Dong, Lin Ai, Ziwei Gong, Songfang Huang, Li Zongsheng, Ehsan Hoque, Julia Hirschberg, and Yue Zhang. 2024. [A survey on open information extraction from rule-based model to large language model](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 9586–9608, Miami, Florida, USA. Association for Computational Linguistics.
- Sungho Park, Jueun Kim, and Wook-Shin Han. 2026. [SPARTA: Scalable and principled benchmark of tree-structured multi-hop QA over text and tables](#). In *The Fourteenth International Conference on Learning Representations*, Rio de Janeiro, Brazil.
- Sungho Park, Joohyung Yun, Jongwuk Lee, and Wook-Shin Han. 2025. [HELIOS: Harmonizing early fusion, late fusion, and LLM reasoning for multi-granular table-text retrieval](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32424–32444, Vienna, Austria. Association for Computational Linguistics.
- Michał Pietruszka, Michał Turski, Łukasz Borchmann, Tomasz Dwojak, Gabriela Nowakowska, Karolina Szyndler, Dawid Jurkiewicz, and Łukasz Garncarek. 2024. [STable: Table generation framework for encoder-decoder models](#). In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2454–2472, St. Julian's, Malta. Association for Computational Linguistics.
- Abhishek Rajgaria, Kushagra Dixit, Mayank Vyas, Harshvardhan Kalalbandi, Dan Roth, and Vivek Gupta. 2025. [No universal prompt: Unifying reasoning through adaptive prompting for temporal table reasoning](#). In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 2800–2821, Mumbai, India. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics.
- Mushtari Sadia, Zhenning Yang, Yunming Xiao, Ang Chen, and Amrita Roy Chowdhury. 2025. [SQUiD: Synthesizing relational databases from unstructured text](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31987–32012, Suzhou, China. Association for Computational Linguistics.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*.
- Qi Shi, Han Cui, Haofeng Wang, Qingfu Zhu, Wanxiang Che, and Ting Liu. 2024. [Exploring hybrid question answering via program-based prompting](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11035–11046, Bangkok, Thailand. Association for Computational Linguistics.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. [Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,

pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. [Robust text-to-SQL generation with execution-guided decoding](#). *arXiv preprint arXiv:1807.03100*.

Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. [Text-to-table: A new way of information extraction](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2518–2533, Dublin, Ireland. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [ReAct: Synergizing reasoning and acting in language models](#). In *The Eleventh International Conference on Learning Representations*.

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*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Xiaohan Yu, Pu Jian, and Chong Chen. 2025. [TableRAG: A retrieval augmented generation framework for heterogeneous document reasoning](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14063–14082, Suzhou, China. Association for Computational Linguistics.

Yilun Zhao, Yunxiang Li, Chenying Li, and Rui Zhang. 2022. [MultiHiertt: Numerical reasoning over multi hierarchical tabular and textual data](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6588–6600, Dublin, Ireland. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. [Judging LLM-as-a-judge with MT-bench and chatbot arena](#). In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Datasets and Benchmarks Track*, volume 36.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. [TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance](#). In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long*

Stage	Purpose	Figure
<i>SGR</i>		
(i)	Schema-guided passage grouping	Fig. 4
(ii.a)	Side-table schema induction	Fig. 5
(ii.b)	Per-table extraction prompt design	Fig. 6
(iii)	Attribute-value extraction	Fig. 7
(iv)	Normalization and database assembly	Fig. 8
<i>QCRC</i>		
(i)	Question-conditioned planning	Fig. 9
(ii)	Embedding-similarity gate	—
(iii)	Targeted extraction	Fig. 10
(iv)	SQL execution over the completed state	Fig. 11
(v)	Residual evidence fallback	—
<i>Eval</i>	LLM-as-Judge	Fig. 12

Table 5: Index of prompt templates used in this paper. SGR is described in §4.1 and QCRC in §4.2.

Papers), pages 3277–3287, Online. Association for Computational Linguistics.

A Additional Details and Analyses

This section provides implementation details and supporting analyses that complement the main paper without changing the experimental protocol. It covers the full set of prompt templates (§A.1), the LLM-AS-JUDGE prompt and its agreement with manual annotation (§A.2), an error analysis that separates residual relational-state error from downstream SQL error (§A.3), and the remaining implementation settings (§A.4). Appendix B and Appendix C then report the entity-link robustness study and the extended cost analysis.

A.1 Prompt Templates

We list every prompt template used by TRACE. Each prompt is rendered as a single light-framed box containing both the system and user messages; curly-brace placeholders (`{like_this}`) denote inputs filled in per call. All calls are run at temperature 0. Table 5 indexes the prompts by the implementation step they support. The prompt text is preserved verbatim from the implementation, so its internal wording, placeholder names, and JSON keys are intentionally left unchanged; the table and captions provide only a mapping to the terminology of the main paper, in which SGR constructs the relationalized database $\mathcal{D}_m$ and QCRC repairs it into the question-completed database $\mathcal{D}_q$.

A.2 LLM-AS-JUDGE Prompt

We adopt a factual-equivalence judging template in the style of Zheng et al. (2023), restricted to a binary verdict to match the QA setting. The judge is gpt-4o, decoding is at temperature 0,

SPARTA	LJ	Manual	Agree.	κ
NBA	49.20	49.03	98.41	0.968
Medical	76.46	80.35	96.11	0.885
Movie	48.85	49.03	98.41	0.968

Table 6: Agreement between LLM-AS-JUDGE (LJ) and manual annotation on SPARTA (565 predictions per domain, Qwen3.5-35B-A3B). **Agree.**: percentage of matching verdicts; κ : Cohen’s kappa.

SPARTA	TRACE	Gold	Δ
NBA	47.76	68.00	+20.24
Medical	73.86	73.99	+0.13
Movie	53.23	64.71	+11.48

Table 7: Token-level F1 with Qwen3.5-35B-A3B using SGR-induced side tables (TRACE) versus the benchmark’s gold grounding tables (**Gold**), with the rest of the pipeline unchanged.

and the same system/user template is used for every example. Both the gold answer and the prediction are passed as a list, so singleton-answer and multi-answer questions are handled by the same prompt. We parse the verdict with the regular expression `\[ \[ (CORRECT|INCORRECT) \] \]` applied to the judge’s output; if the pattern does not match, which happens in $< 0.5\%$ of calls, we re-run the judge once and otherwise count the prediction as INCORRECT. Figure 12 gives the exact judge prompt used in the implementation.

Agreement with human judgments. To verify that LLM-AS-JUDGE is a faithful stand-in for human assessment, we manually annotated *all* 1,695 TRACE predictions produced by Qwen3.5-35B-A3B (565 per SPARTA domain) rather than a sample. Table 6 reports the result: the judge and the human annotator agree on 96.11–98.41% of examples, with Cohen’s κ between 0.885 and 0.968.

A.3 Error Analysis

Because SGR and QCRC are LLM-driven, errors in the completed relational state and errors in downstream SQL reasoning are easily conflated. We separate them in two ways.

Gold grounding tables. Replacing only the induced side tables with the benchmark’s gold grounding tables, leaving every other component unchanged, bounds how much of the remaining error is attributable to state construction (Table 7). The bound is domain-dependent: on SPARTA-Medical the induced state is already near-gold (+0.13 F1), whereas SPARTA-NBA leaves substan-

tial headroom (+20.24). The residual gap after substitution is attributable to query formulation and execution rather than to extraction.

Error audit. To characterize *how* the state fails, we audited the 164 genuinely incorrect SPARTA-Movie outputs whose SQL returned a non-empty result. Only 2.4% were caused by incorrectly populated cells, while 56.7% involved an incomplete state. Wrong-value corruption is therefore rare among the audited errors, and omission is the dominant extraction-related failure mode. This is consistent with the design choice of exposing residual incompleteness as NULL rather than fabricating values (§4.2), and it identifies coverage—not fidelity—as the more productive target for future work.

A.4 Additional Implementation Details

Table 8 lists the additional implementation details used in the experiments. The gate query templates are the textual queries $u_{r,A}$ of §4.2, rendered from the anchor cell value and the attribute name; for attributes proposed by the planner, the attribute description is appended so that a newly introduced column is matched on its intended meaning rather than on its name alone.

B Robustness to Imperfect Entity Links

We evaluated all 565 SPARTA-Medical questions with Qwen3.5-35B-A3B under two complementary link perturbations: *removal* deletes sampled entity–passage links, and *replacement* redirects them to alternative passages from the same source column. Each is applied *edge-wise* (sampling links independently) or *document-wise* (sampling documents and perturbing every link incident to them, which jointly eliminates redundant evidence paths). As Table 9 shows, TRACE degrades gracefully under both missing and incorrect links, and even at 100% removal remains above HPROPRO (34.92 F1) and ODYSSEY (24.45 F1) evaluated with complete links under the same backbone and split.

C Extended Cost Analysis

Complementing the token-based analysis in §5.5, we report dollar cost under gpt-4o-mini pricing, measured wall-clock latency, and storage of the materialized state (Table 10). On SPARTA, the one-time offline cost ranges from negligible on Medical (\$0.005) to substantial on NBA (\$12.67), after which each query costs well under a cent; on HYBRIDQA, where each instance carries its own

Component	Setting
<i>SGR (§4.1)</i>	
Schema induction	10 sampled passages per side-table group
<i>QCRC (§4.2)</i>	
Gate threshold τ	0.65
Gate query, existing col.	{cell_value}: {column_name}
Gate query, added col.	{cell_value}: {column_name} - {description}
SQL agent	at most 5 turns; SQL errors returned as observations for self-correction
Residual fallback	top-3 BGE-M3 passages, invoked only after an empty SQL result
<i>Models</i>	
Closed backbone	gpt-4o-mini-2024-07-18
Open backbone	Qwen3.5-35B-A3B
LLM-AS-JUDGE	gpt-4o-2024-08-06
Decoding	temperature 0 for all components

Table 8: Additional implementation details used in all experiments.

Perturbation	Granularity	Perturbation rate								
		0%	10%	20%	30%	50%	80%	90%	95%	100%
Removal	Edge-wise	73.86	73.52	73.07	73.81	73.61	67.76	61.30	58.85	49.94
	Document-wise		67.15	65.73	63.78	58.90	—	—	—	—
Replacement	Edge-wise	73.86	73.45	73.24	70.46	72.13	67.87	62.21	59.00	57.75
	Document-wise		68.25	66.50	62.50	59.30	—	—	—	—

Table 9: Token-level F1 on SPARTA-Medical ($n=565$) with Qwen3.5-35B-A3B under entity–passage link perturbation (§B). Dashes mark rates not evaluated.

	Monetary cost (USD)			Wall-clock latency (s)		Materialized relational state
	Offline (SGR)	Online (QCRC) / query	Total	Offline (SGR)	Online (QCRC) / query	
SPARTA (reported per domain)						
Medical	0.005	0.0017	0.98	34	7.6	60 MB
Movie	1.43	0.0014	2.21	1,385	20.9	2.3 GB
NBA	12.67	0.0069	16.54	7,723	15.2	4.7 GB
HYBRIDQA (reported per instance)						
	0.0028	0.0022	0.0050	6.9	6.9	22.6 KB

Table 10: Cost of TRACE under gpt-4o-mini pricing. *Offline* is the one-time SGR pass that builds $\mathcal{D}_m$; *Online* is the per-question QCRC completion and SQL execution over $\mathcal{D}_q$. **Total** accumulates the online cost over all 565 queries of a SPARTA domain.

Break-even (queries)	Medical	Movie	NBA
vs. HPROPRO	1	70	809
vs. ODYSSEY	8	119	984

Table 11: Minimum query count at which TRACE’s offline-plus-online cumulative cost falls below each baseline’s online cost (gpt-4o-mini pricing).

tables and passages, relationalization is incurred per instance instead. We define *break-even* as the minimum number of queries at which TRACE’s offline-plus-online cumulative cost falls below the baseline’s cumulative online cost.

Table 11 reports the resulting break-even points. These values delimit the workload regime in which TRACE should be deployed rather than a requirement on every workload: TRACE is appropriate when a stable corpus is expected to receive enough

coverage-sensitive queries to amortize relationalization, while one-off or predominantly lookup-oriented workloads are better served by a conventional retrieval pipeline. Consistent with this, HYBRIDQA has no finite monetary break-even under the current configuration.

D AI Assistants

We used Claude (Anthropic) as a coding assistant during implementation and debugging, and ChatGPT (OpenAI) to draft figures and improve the clarity of the writing. The system design, experimental methodology, and analyses were conceived by the authors, and all code, results, and references were manually verified by the authors.

System:

For each input column, decide what kind of entity its linked passages collectively describe. That entity is the new_table_name. The source tables are given so you don't duplicate them.

- Each column gets exactly one new_table_name.
- Reuse the same new_table_name across columns whose passages describe the same kind of entity.
- new_table_name must be short snake_case.
- new_table_name must NOT match any source table name; pick a name that reflects the entity in the passages.
- The column field must use the exact form ``<src>.<col>`` shown in the input.

Strict JSON output:

```
{
  "labels": [
    {
      "column": "<src>.<col>",
      "new_table_name": "...",
    },
    ...
  ]
}
```

User:

Source tables (do NOT name your new tables after these):

```
{source_tables_block}
```

Linked columns and sample passages:

```
{linked_columns_block}
```

Figure 4: Schema-guided passage grouping prompt (Step i of §4.1). Decides which source-table columns should be grouped into the same materialized table based on the kind of entity their linked passages describe.

System:

Given an entity (table_name) and sample passages, produce a list of attributes. Each attribute's type must be one of int | float | date | text. Prefer sortable types (int / float / date) whenever the passages support it. Cover as much information from the passages as possible with the attributes.

Strict JSON output:

```
{
  "attr": [
    {
      "name": "<snake_case>",
      "type": "<int|float|date|text>",
    },
    ...
  ]
}
```

User:

```
table_name: {table_name}
```

passages:

```
{passages_block}
```

Figure 5: Side-table schema induction prompt (Step ii of §4.1). Proposes the typed columns of a materialized table given sample passages describing instances of its entity type.

System:

You author the extraction-prompt content for one materialized table.

Inputs are the table's schema (table_name + typed attributes) and sample passages describing instances of that entity.

Produce:

- "overview": one or two short sentences describing what these passages share -- the entity kind and the typical content.
- "attrs": for each schema attribute, a short phrase describing the expected value format (units, ordering, conventions) and where in a passage it usually appears.

Strict JSON output:

```
{
  "overview": "...",
  "attrs": [
    { "name": "<attr_name>",
      "format": "..."},
    ...
  ]
}
```

User:

table_name: {table_name}

schema:

{schema_block}

passages:

{passages_block}

Figure 6: Per-table extraction prompt design (part of Step ii of §4.1). Writes the materialized-table-specific extraction prompt content (overview and per-attribute hints) that step (iii) later applies to each passage in the group.

System:

Extract the listed attributes from the passage.

Output JSON {attribute: value-or-null}; use null when the passage does not state it.

Extract only what the passage says about the entity indicated by cell_value.

Each attribute is given as `name (type): format`:

- `format` describes how the value appears in the passage (surface form, units, location hints) -- use it to locate and read the raw value.
- `type` describes how to convert that raw value into the JSON output. Convert units / parse the surface form accordingly:
 - int -> plain integer, no thousand separators, no units.
 - float -> plain number, no thousand separators, no units.
 - date -> 'YYYY-MM-DD'. Year-only -> 'YYYY-01-01'; month/day-only -> '9999-MM-DD'.
 - text -> short literal string copied from the passage.

Never invent values not present in the passage.

User:

table_name: {table_name}

overview: {overview}

attrs:

{attrs_block}

cell_value: {cell_value}

passage:

{passage}

Figure 7: Attribute-value extraction prompt (Step iii of §4.1). Populates one row of a materialized table from one linked passage, using the overview and per-attribute formats produced by Fig. 6.

System (integer):

Normalize the cell to a JSON integer (no thousand separators, no units).
 Return null if the cell does not state a number.
 Output STRICT JSON: {"value": <int>|null}.

System (real):

Normalize the cell to a JSON number (no thousand separators, no units).
 Return null if the cell does not state a number.
 Output STRICT JSON: {"value": <number>|null}.

System (date):

Normalize the cell to a calendar date 'YYYY-MM-DD'.
 Return null if the cell does not state a date.
 Output STRICT JSON: {"value": "YYYY-MM-DD"|null}.

User (shared):

{examples_block}
 cell: {cell}

Figure 8: Outlier rescue prompts (Steps iv of §4.1). One system prompt per target type (int, float, date); few-shot in-column examples are passed via the shared user prompt.

System:

You are a planner for a database QA task.
 You are given a natural-language question and a SQLite DB schema. The DB has two kinds of tables:
 - Source tables: original structured data.
 - Side tables: derived by LLM extraction from passages -- values may be incomplete or imprecise.

Your job: decide which tables and columns are required to answer the question.

If the existing columns are insufficient, you may propose new attributes to be extracted from passages -- but new attributes can only be added to materialized tables (since they are extracted from passages). Source tables cannot gain new columns.

Output STRICT JSON:

```
{
  "needed": [
    {"table": "<table>",
     "columns": ["<existing_col1>", "<existing_col2>", ...],
     "new_columns": [
       {"name": "<snake_case>",
        "type": "<int|float|date|text>",
        "description": "<ONE short sentence describing what this attribute represents>"},
       ...
     ]},
    ...
  ],
  ...
}
```

- Use exact table/column names from the schema for existing columns.
- Omit `new\_columns` (or use an empty list) if the existing columns are sufficient.
- `new\_columns` may appear ONLY on materialized tables (not source tables).
- Propose `new\_columns` only when no existing column already captures the needed information.
- Each `description` MUST be a single short sentence (no enumeration, no multi-step reasoning).

User:

Question: {question}

DB SCHEMA:

```
/*
{schema}
*/
```

Based on the question and the schema, list the tables and columns required to answer.

Figure 9: Question-conditioned planning prompt (Step i of §4.2). The planner reads the schema of $\mathcal{D}_m$ and the question, then lists the materialized-table columns the answer requires, separating NULL-cell completion of existing columns from on-the-fly addition of new columns.

System:
 You extract one attribute value for a subject from a passage.
 The original question is provided as context -- extract the value that is most relevant for answering it.
 Only extract what the passage states about the subject. Use null if the passage does not state the attribute.

Type rules for the output value:

- int -> plain integer (no thousand separators, no units).
- float -> plain number (no thousand separators, no units).
- date -> 'YYYY-MM-DD'. Year-only -> 'YYYY-01-01'; month/day-only -> '9999-MM-DD'.
- text -> short literal string copied from the passage.

Output STRICT JSON: {"value": <typed value> | null}.

User:
 original_question (context): {question}

attribute: {name} ({type}) -- {description}

subject (cell_value): {cell_value}

passage:
 {passage}

Extract the attribute value for this subject.

Figure 10: Targeted extraction prompt (Step iii of §4.2). For each candidate (row, column) that the planner requested—and that passed the embedding similarity gate—this prompt extracts a single typed value from one linked passage.

```
System:
You are ReAct style QA expert.
You are working on SQLite DB.

Each turn, you can execute sql to find more information.
If enough information is gathered, you must answer the original question.

Output strict JSON:
{
  "thought": "...",
  "sql": "<a self-contained SQLite SELECT>"
}
or
{
  "thought": "...",
  "answer": "<short literal value(s)>"
}

User:
Original Question: {question}

DB SCHEMA:
/*
{schema}
*/

History:
{history}

Collected passages (cumulative, in attach order):
{collected}

Based on above information, select right action.
```

Figure 11: SQL execution over the completed state prompt (Step iv of §4.2). Each turn the agent emits either a self-contained SQLite SELECT against $\mathcal{D}_q$ or a final answer; passages attached to rows returned by previous queries are appended cumulatively to collected.

System:
You are an impartial judge evaluating whether a candidate answer to a question is semantically equivalent to a reference answer.

A candidate answer is CORRECT iff:

- It refers to the same set of entities, quantities, dates, or facts as the reference answer.
- Differences are only in surface form -- abbreviations ("Lakers" = "Los Angeles Lakers"), numerals vs. words ("5" = "five"), unit conversions ("1.50 m" = "150 cm"), articles, or paraphrases.

A candidate answer is INCORRECT if any of the following hold:

- It refers to a different entity, quantity, date, or fact.
- For multi-answer questions: it is missing any reference item, or it contains any extra item not in the reference set. Order does not matter.
- It is vague or noncommittal (e.g., "unknown", "cannot determine") while the reference is specific.
- It contains the right item together with contradictory information.

First, briefly justify your decision in one sentence.
Then output your final verdict on a new line, strictly in the format:
Verdict: [[CORRECT]] or Verdict: [[INCORRECT]]

User:
[Question]
{question}

[Reference Answer]
{gold_list}

[Candidate Answer]
{prediction_list}

Figure 12: LLM-AS-JUDGE prompt (§5.1). Both gold_list and prediction_list are JSON arrays; singleton-answer questions use a list of length one.