CORE-T: COherent REtrieval of Tables for Text-to-SQL

Anonymous ACL submission

Abstract

Realistic text-to-SQL workflows often require joining multiple tables. As a result, accurately retrieving the relevant set of tables becomes a key bottleneck for end-to-end performance. We study an *open-book* setting where queries must be answered over large, heterogeneous table collections pooled from many sources, without clean scoping signals such as database identifiers. Here, dense retrieval (DR) achieves high recall but returns many distractors, while join-aware alternatives often rely on extra assumptions and/or incur high inference overhead. We propose CORE-T, a scalable, training-free framework that enriches tables with LLM-generated purpose metadata and pre-computes a lightweight table-compatibility cache. At inference time, DR returns top-K candidates; a single LLM call selects a coherent, joinable subset, and a two-step additive adjustment stage restores strongly compatible tables. Across BIRD, SPIDER, MMQA, and BEAVER, CORE-T improves over DR by up to 22.7 points in table-selection F1 while returning up to 40% fewer tables, and by up to 24.4 points in multi-table execution accuracy, and uses 1.64–4.20× fewer total selection tokens than LLM-intensive baselines.[1]

1 Introduction

Natural language interfaces to structured data (text-to-SQL) aim to let non-experts query relational tables in everyday language (Yu et al., 2018; Li et al., 2023). Modern pipelines often retrieve relevant tables before generating SQL, making table retrieval a critical bottleneck: if the retrieved set misses required tables, contains distractors, or lacks a valid join path, the SQL generator must guess joins, drop constraints, or produce incomplete SQL.

In open-book text-to-SQL, retrieval is not merely a query–table relevance problem. Many analytical questions require composing evidence across normalized tables, so the retriever must return a *set* that is both semantically relevant and structurally coherent. Relevance-only retrieval can hand the SQL generator plausible but disconnected tables rather than a compact, join-ready schema slice.

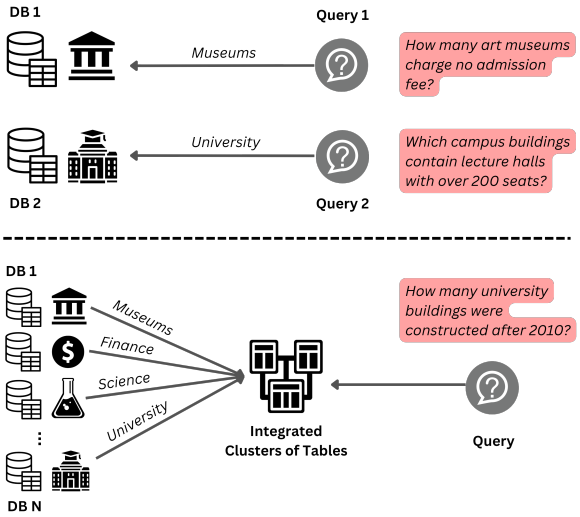

Figure 1: **Querying setups.** *Top:* Closed-book: each query targets a known database. *Bottom:* Open-book: queries must be answered over *integrated* clusters of tables spanning various domains pooled from multiple database schemas.

Most prior text-to-SQL and table-retrieval work assumes a *closed-book* setting, where the target database or schema graph is known and retrieval is confined to a small schema (Yu et al., 2018; Lee et al., 2021; Li et al., 2023; Zhong et al., 2018; Herzig et al., 2021; Chen et al., 2020; Wang et al., 2022). In contrast, data-lake and semantic join discovery research studies *open-book* analytics over large heterogeneous table collections, where relevant and joinable tables must be discovered without a predefined schema graph (Zhu et al., 2019; Nargesian et al., 2019; Zhang and Ives, 2020; Cong et al., 2023). This mirrors integrated enterprise corpora, where tables are pooled from many sources and

[1] Our code and data are anonymously available at https://anonymous.4open.science/r/CORE-T-753C.

clean scoping signals such as database identifiers (db_id) are unavailable.

Figure 1 illustrates the challenge. In a closed-book setting (top), queries such as *"How many art museums charge no admission fee?"* can be answered within a delimited schema (db_id), e.g., museums vs. university. In an open-book setting (bottom), queries such as *"How many university buildings were constructed after 2010?"* require retrieval over pooled tables from many domains. A table named buildings may occur in both museum and university collections; without db_id, the retriever must disambiguate candidates using attributes and relationships. Missing one bridge table can break the join path, while similar-looking distractors can induce spurious joins; high recall alone is therefore insufficient.

Our goal is therefore not to introduce a new retrieval model in isolation, but to provide a *training-free systems design* for *pooled open-book* corpora where db_id and gold foreign keys are unavailable, and where the central bottleneck is *join-coherent* table-set retrieval. Existing retrievers address parts of this goal but leave important gaps. Dense retrieval (DR) (Karpukhin et al., 2020) scales well but remains join-agnostic. ReAct (Yao et al., 2023) can iteratively expand evidence but requires multiple LLM calls. Join-aware methods such as JAR and ARM (Chen et al., 2024, 2025c) model relational structure, but rely on scoping assumptions such as db_id and/or incur substantial inference overhead. REAR (Agarwal et al., 2025) improves multi-table retrieval via retrieve–expand–refine stages, but column-similarity-based join evidence can produce false positives in pooled corpora with semantically similar yet non-joinable tables.

This raises our central question: *Can we design a scalable, training-free retriever for pooled open-book corpora that jointly exploits query–table relevance and table–table compatibility, without assuming* db_id *or gold foreign keys?*

We propose CORE-T, a framework for *COherent REtrieval of Tables* for text-to-SQL. As shown in Figure 2, CORE-T moves reusable schema understanding offline and keeps online inference lightweight: it enriches tables with LLM-generated purpose metadata, precomputes a compatibility cache from column-level semantic and value-based signals, retrieves a high-recall top-K candidate set, uses one LLM call to select a coherent subset, and applies a two-step additive adjustment to restore strongly compatible and over-pruned candidates.

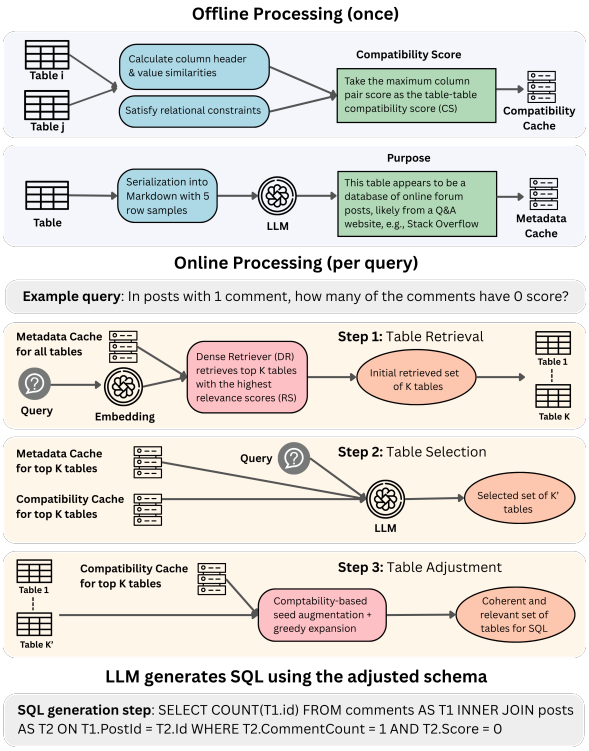

Figure 2: **Overview.** CORE-T combines offline table enrichment and compatibility caching with a lightweight online pipeline: DR, single-shot LLM selection, and a two-step additive adjustment final stage.

In summary, we contribute:

Training-free join-coherent table-set retrieval for pooled open-book corpora. We introduce CORE-T, a scalable retrieval layer that couples purpose-enriched query–table retrieval with cached table–table compatibility evidence to produce compact, join-coherent schema slices. Compatibility provides structured guidance for LLM subset selection and targeted additive recovery, enabling open-book retrieval without db_id scoping or gold foreign-key annotations.

Improved effectiveness and efficiency under pooled evaluation. We compare against DR, ReAct, and recent join-aware methods (JAR/ARM/REAR) under pooled multi-database evaluation on BIRD, SPIDER, MMQA, and BEAVER (Yu et al., 2018; Li et al., 2023; Wu et al., 2025; Chen et al., 2025a). CORE-T improves the precision–recall balance for multi-table retrieval, returns more coherent table sets for SQL generation, and reduces LLM usage (up to $\sim$5$\times$ fewer input tokens than heavier multi-draft LLM generation methods, e.g., ARM).

2 Related Work and Positioning

Given the pooled open-book setting introduced in Section 1, the key distinction among prior methods is not only whether they retrieve relevant tables, but also how they maintain table-set coherence when join evidence is noisy and scoping information is limited. This distinction is especially important in integrated corpora, where noisy join signals and near-duplicate cross-domain tables make selection difficult. JAR and ARM build join graphs and use MIP-based optimization, making them sensitive to the induced compatibility structure; both assume `db_id` for scoping, and ARM adds LLM self-verification overhead. REAR is LLM-free at online inference, separating semantic retrieval from joinability-based expansion and refinement, but its local, column-similarity-driven join evidence can be brittle with semantically similar yet non-joinable columns. In contrast, CORE-T targets open-book corpora with (i) LLM-generated *purpose* metadata for candidate disambiguation without `db_id`, (ii) modified compatibility scoring for more reliable join evidence, and (iii) single-shot LLM selection and a two-step additive adjustment stage to retain a high-recall, join-coherent table set. Concurrent anonymous work, JOINGR (Anonymous, 2026), learns supervised join-graph traversal over column/foreign-key edges, whereas CORE-T is training-free and targets pooled open-book corpora without `db_id` or gold foreign keys.

3 Overview

Figure 2 provides an overview: schema understanding is moved to offline enrichment and caching, while online inference remains lightweight.

Offline. We enrich tables with brief purpose metadata and build a dense index over enriched table representations, following offline index enrichment (Chen et al., 2025b). We also pre-compute a table–table compatibility cache that approximates joinability and provides candidate join edges.

Online. Given a query, DR returns a top-K candidate set. A single LLM call selects a coherent, connected subset using the candidate tables and cached compatibility evidence, and a two-step additive adjustment recovers strongly compatible tables from the original top-K set before SQL generation.

4 Methodology

We propose a scalable, join-aware multi-table retriever for *open-book* text-to-SQL, where tables from multiple DBs are pooled, and the system must retrieve relevant, joinable tables *without* `db_ids` or gold foreign keys. Let $\mathcal{T} = \{t_1, \ldots, t_N\}$ be the unified table corpus. Given a query q, we output a table set $S(q) \subseteq \mathcal{T}$ that balances high gold-table recall with fewer irrelevant tables, while remaining coherent for downstream SQL generation.

4.1 Offline Processing and Caching

Our method uses two offline, reusable, query-agnostic signals: (i) an enriched table index for DR, used to compute the relevance score $\mathrm{RS}(q, t)$ at inference time, and (ii) a table–table compatibility score $\mathrm{CS}(t_i, t_j)$ that approximates joinability.

Table enrichment and indexing. Offline, each table t is serialized as a 5-row Markdown representation and augmented with an LLM-generated *purpose description*. We sample 5 rows uniformly without replacement to provide a lightweight value snapshot, avoiding full-table inputs while exposing different values and formats. The purpose description summarizes the table's contents and typical use (e.g., key entities, attributes, and granularity), using a fixed prompt in Appendix K (cf. §3). We embed the concatenation of the Markdown snapshot and purpose as $e_t = f_{\text{tbl}}(\texttt{Markdown+purpose})$ and store all vectors in a FAISS index for online retrieval. Appendix B discusses our compact metadata design and 5-row cap.

Column signals. To approximate joinability without foreign keys, we compute lightweight column-level signals. For each column c, we embed its header text (`table_name+column_name`) with f_{col} to obtain e_c. For each cross-table pair (c_i, c_j), we compute (i) header similarity (exact lexical + embedding-based semantic), (ii) value overlap (Jaccard), inspired by the pairwise similarity signals used in JAR, and two *relational constraints* that better mimic key–foreign-key joins: (iii) *uniqueness* and (iv) *subset*, all ignoring NULLs.

Compatibility cache. We combine these signals into a column-pair compatibility score $s(c_i, c_j) \in [0, 1]$ with a hand-crafted function and a hard key–foreign-key-like constraint: we only score pairs where *at least one* column is unique and the values exhibit a subset relation. This promotes key-like joins while suppressing spurious matches from

generic columns (e.g., `id`, `name`) in pooled corpora. Details and the scoring function are in Appendix C (Eq. 3, Figure 4). The table–table compatibility score is the best valid column match:

$$\mathrm{CS}(t_i, t_j) = \max_{\substack{c \in C(t_i),\, c' \in C(t_j) \\ \mathrm{valid}(c,c')}} s(c, c'), \qquad (1)$$

and we cache the corresponding argmax join columns; if no valid pair exists, $\mathrm{CS}(t_i, t_j) = 0$. The cache is computed offline and reused across queries. Appendix C reports joinability gains over JAR-style similarity scoring (Table 5) and discusses sparse construction at inference time and incremental maintenance under corpus updates.

4.2 Online Processing and Inference

Given a query q, we produce a final table set $S(q)$ in three stages: (1) DR over enriched table embeddings to obtain top-K candidates, (2) a single LLM call to select a coherent, joinable subset, and (3) a two-step additive adjustment stage that restores strong compatible tables omitted by the selector.

4.2.1 Table Retrieval

We obtain a high-recall candidate set using DR over enriched table embeddings created offline. This step is illustrated in Appendix D.1 (Figure 5).

Query embedding and scores. For each query q, we encode it with the table embedding model, $e_q = f_{\text{tbl}}(q)$, and compute its relevance to each table t as cosine similarity, $\mathrm{RS}(q, t) = \cos(e_q, e_t)$.

Top-K initial set. We retrieve the top-K tables by $\mathrm{RS}(q, t)$, $T_K(q) = \{t^{(1)}, \ldots, t^{(K)}\}$. This set is high-recall but may include loosely related or distractor tables, motivating selection and adjustment stages. Subsequent stages operate on $T_K(q)$.

4.2.2 Table Selection

DR optimizes query–table relevance but ignores table–table interactions. To obtain a smaller, joinable subset while preserving high recall, we use a single LLM call prompted as a *SQL schema analyst* that follows a human-reasoning workflow to jointly reason over the query, candidate tables, and cached compatibility evidence. The prompt is *few-shot*, with one synthetic example illustrating the expected input and output format. Full details are in Appendix D.1 (cf. §4).

LLM input and output. Given $T_K(q) = \{t^{(1)}, \ldots, t^{(K)}\}$, we provide: (i) the query q; (ii) an indexed list of the K candidate tables (name, 5-row Markdown snapshot, generated purpose); and (iii) compatibility evidence for pairs with $\mathrm{CS}(t_i, t_j) > 0$, including `overall_compatibility` score and `best_join_columns`. Pairs not listed are treated as having no join edge. The LLM forms connected groups and selects one, parsed from JSON as $T_{K'}(q) \subseteq T_K(q)$. This subset is typically smaller and more coherent due to conservative pruning instructions, but may over-prune compatible tables. We address this with the compatibility-driven adjustment step that restores strongly compatible ignored tables from the original top-K set.

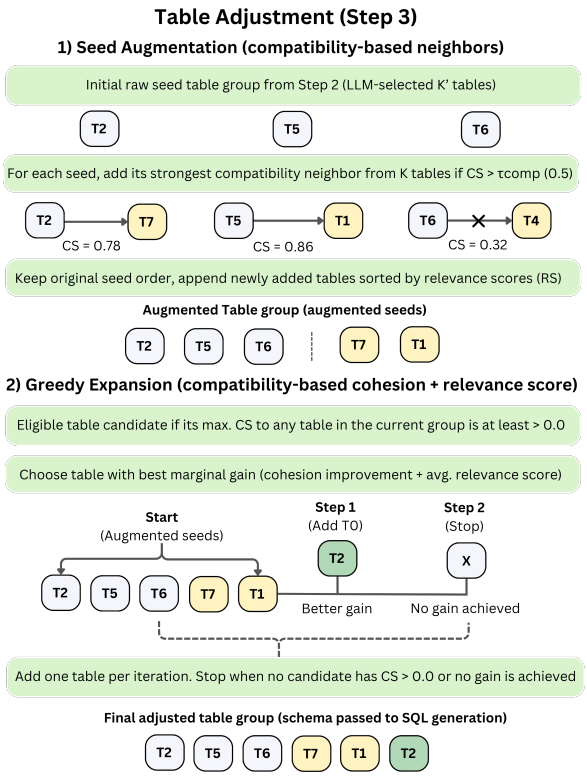

Figure 3: **Table Adjustment.** The LLM-selected seed tables are first augmented with their strongest compatible neighbors, preserving seed order and ranking added tables by retriever relevance. The group is then expanded greedily by adding eligible candidates with the largest marginal gain until no beneficial table remains.

4.2.3 Table Adjustment

Seeded compatibility-based augmentation. As illustrated in Figure 3, the adjustment step treats the LLM-selected tables $T_{K'}(q)$ as a seed group within the retrieved top-K candidates $T_K(q)$. For each seed table, we add its strongest compatibility neighbor in $T_K(q)$, according to cached CS, only when the score exceeds τ_{comp}. The resulting augmented group G_0 preserves the LLM-selected order and appends newly added tables sorted by

Dataset	#DB	#Tab	#Empty	Rows/T	Cols/T	#Q	MT%	EG%	$\bar{G}$
BIRD	11	75	0	52,437	10.6	1,534	76.4	0.0	1.95
SPIDER	20	81	0	6,665	5.4	1,034	44.4	4.7	1.51
MMQA	1	710	0	1,547	5.1	1,105	99.6	10.9	2.20
BEAVER	6	463	214	42,404	9.2	209	98.6	20.1	4.44

Table 1: Dataset stats after pooling. #Empty: zero-row tables; Rows/T and Cols/T: averages per table; MT%: queries with ≥ 2 gold tables; EG%: queries with empty gold execution result; $\bar{G}$: avg. gold tables/query.

descending relevance score to the query. G_0 then initializes the greedy expansion.

Greedy expansion. Starting from G_0, we greedily add tables from $T_K(q) \setminus T_{K'}(q)$ until no candidate yields positive marginal gain. A candidate is eligible only if it has nonzero compatibility with the current group. Among eligible candidates, we select the table with the largest marginal gain, defined as the combined improvement in mean pairwise compatibility plus mean relevance score to the query. The final schema $S(q)$ preserves the LLM-selected seeds while recovering additional compatible and relevant tables for SQL generation. Full details and equations are in Appendix D.2.

5 Experiments

5.1 Experimental Setup

5.1.1 Datasets

We evaluate on the dev splits of BIRD (Li et al., 2023) and SPIDER (Yu et al., 2018), as well as MMQA (Wu et al., 2025) and BEAVER (Chen et al., 2025a), all with multi-table requirements. BIRD/SPIDER enable comparison with join-aware retrievers (e.g., JAR/ARM), while MMQA and BEAVER provide larger, multi-table-intensive stress tests under integrated corpora. For BIRD, SPIDER, and MMQA, we pool tables from multiple DBs or question-specific schemas into one retrieval corpus and remove `db_ids`. BEAVER is already open-book and only requires removing `db_id`. Table 1 reports the resulting statistics. For MMQA, we use a stratified one-third subset (1,105 queries) for efficiency. Appendix E provides preprocessing details, benchmark-realism discussion, and notes on additional benchmarks such as SPIDER 2.0 (Lei et al., 2025).

5.1.2 Baselines

We compare against dense retrieval (DR) (Karpukhin et al., 2020), an agentic ReAct-based retriever (Yao et al., 2023), and recent join-aware methods: JAR (Chen et al., 2024), ARM (Chen et al., 2025c), and REAR (Agarwal et al., 2025). All are evaluated in the same open-book setting.

Baseline overview. DR retrieves top-K tables by cosine similarity between the query embedding and enriched table embeddings (Markdown + purpose), denoted DR@K. ReAct iteratively queries the same dense index for up to three steps. JAR is a join-aware reranker that uses a mixed-integer program (MIP) to select a connected set of K tables by jointly optimizing query coverage and inferred join compatibility. ARM adds LLM-guided alignment for candidate retrieval, followed by join-aware MIP selection and LLM self-verification via multiple draft generations. REAR is an LLM-free retrieve–expand–refine pipeline that retrieves query-relevant base tables, expands them with structurally joinable candidates using precomputed column embeddings, and refines the pool by jointly scoring query–table relevance and table–table joinability. Appendix F gives implementation details and a thorough comparison; Table 6 summarizes LLM-call cost, `db_id` assumptions, and each method's core *rationale*.

5.1.3 Evaluation Metrics

We evaluate (i) table retrieval/selection quality, (ii) end-to-end text-to-SQL execution accuracy, and (iii) efficiency. We report set-based precision (P), F1, and perfect recall (PR) for table selection; execution accuracy (EX) for SQL generation overall (EX_{all}), on multi-table queries (EX_{MT}), and on the perfect-recall subset (EX_{PR}). We also report token-based measures for LLM usage. More details about metric definitions are provided in Appendix G.1.

5.1.4 Implementation Details

LLM setup. We use two LLMs for table selection: Llama-3.1-8B-Instruct (Meta AI, 2024; Team, 2024) and Qwen-2.5-7B-Instruct (Yang et al., 2024).

Embeddings and thresholds. After comparing embedding models on MTEB (Muennighoff et al., 2023) and its leaderboard, we use UAE-Large-V1 for our initial table retrieval step and all baselines involving a dense-retrieval step (Li and Li, 2024).

Baselines code. For JAR and ARM, we use the authors' released code and default hyperparameters on supported datasets: JAR provides reproducible scripts for BIRD and SPIDER, while ARM supports BIRD. For REAR, we use the authors' predicted

tables from their best reported configuration. More details and discussion are in Appendix G.2.

5.2 Results and Analysis

How does the choice of embedding model and top-K affect initial retrieval? Table 7 (Appendix) compares embedding models and top-K cutoffs for initial dense retrieval, including snowflake-arctic-embed-m-v2.0 (Yu et al., 2024). Overall, text-embedding-3-large is strongest, while UAE-Large-V1 is a close open-source alternative on BIRD and SPIDER; for example, at K=10 on SPIDER, both reach 25.7 F1. The gap is larger on MMQA and BEAVER. Since increasing K improves PR but lowers precision/F1 and increases context size, we use UAE-Large-V1 with DR@10 as a high-recall starting point.

How can we improve precision in open-book multi-table retrieval while preserving high recall? Table 2 shows that CORE-T improves the precision–PR trade-off over strong baselines by combining compact schema selection with compatibility-based recovery. On BIRD, where direct comparison with prior join-aware methods is possible, CORE-T improves F1 over ARM, JAR, and REAR with Llama selection (62.3 vs. 53.5/50.1/51.0) while returning fewer tables than ARM and REACT (4.1 vs. 5.3/4.8). With Qwen selection, it also improves over ARM on BIRD (72.3 vs. 68.3 F1), suggesting that the gain is not tied to a single selector LLM. On MMQA, CORE-T outperforms REACT and REAR with Qwen selection (56.4 vs. 50.4/39.2 F1), indicating that compatibility-guided table-set construction helps when many queries require multi-table reasoning. On the noisier BEAVER benchmark, CORE-T achieves the strongest F1 with Qwen (43.0), but PR remains low, showing that anonymized enterprise-style schemas and empty tables make complete gold-table recovery difficult for all methods. Overall, CORE-T improves precision and F1 while preserving competitive PR, especially when join-coherent table-set construction matters more than single-table relevance.

Additional analyses of threshold stability, prompt length, and table recovery. Appendix H provides four supporting analyses explaining why the gains are not due to a brittle threshold or simply longer prompts. First, sweeping $\tau_{\text{comp}} \in \{0.3, 0.5, 0.7\}$ changes F1 by at most 4.1 points, PR by at most 1.4 points, and average returned tables by at most 0.4, indicating that the threshold is a stable precision–recall/compactness control rather than a highly tuned heuristic (Table 8). Second, prompt length shows some sensitivity, especially on BEAVER, but is not a monotonic explanation of selection quality, suggesting that schema ambiguity, missing values, and weak table semantics also drive errors (Tables 9 and 11). Third, adjustment recovers at least one dropped gold table in 9.1–57.1% of dropped cases and all dropped gold tables in 9.1–50.8%, confirming its role as a targeted repair step rather than generic expansion (Table 12). Finally, for low-relevance join-critical bridge tables, adjustment fully recovers dropped bridge tables in 22.5–57.1% of affected queries, supporting the value of table–table compatibility when query–table relevance is weak (Table 13).

How does table-selection quality impact downstream SQL execution accuracy, especially for multi-table queries? Table 3 shows that better table selection improves downstream EX most clearly on EX_{MT} and for smaller SQL generators, which are less able to ignore distractors or infer missing joins. With Llama-3.2-3B, CORE-T is the strongest non-oracle method on BIRD, SPIDER, and MMQA (15.7, 34.4, and 18.9 EX_{MT}), with particularly clear gains over REAR on SPIDER and MMQA (+8.0 and +10.5 EX_{MT} points). The EX_{PR} results further show that the benefit is not only from recovering all gold tables: on MMQA, CORE-T improves EX_{PR} over REAR by 8.9 points with Llama-3.2-3B (15.8 vs. 6.9), indicating that compactness and coherence of the selected schema still matter even when the required tables are available. With stronger generators, gains are smaller but remain visible; with GPT-4o-mini, CORE-T reaches 38.6, 56.9, 35.8, and 5.3 EX_{MT} on BIRD, SPIDER, MMQA, and BEAVER. The *Oracle* setting shows substantial headroom (e.g., MMQA: 35.8→65.8 EX_{MT} with GPT-4o-mini), confirming that table selection remains a bottleneck. BEAVER remains difficult because anonymized schemas, empty tables, a small perfect-recall subset, and 20.1% empty gold execution results make execution scores less stable.

Effect of the table selector (Qwen vs. Llama) on downstream EX. Table 14 (Appendix) repeats the evaluation with Qwen-2.5-7B-Instruct as the selector and preserves the same conclusion: better table selection most reliably improves downstream EX, especially on multi-table queries.

	Bird (n=1534, $\bar{G}$=1.95)				**Spider** (n=1034, $\bar{G}$=1.51)				**MMQA** (n=1105, $\bar{G}$=2.20)				**Beaver** (n=209, $\bar{G}$=4.44)			
	Avg.#tab.↓	**P**	**F1**	**PR**	Avg.#tab.↓	**P**	**F1**	**PR**	Avg.#tab.↓	**P**	**F1**	**PR**	Avg.#tab.↓	**P**	**F1**	**PR**
Llama-3.1-8B-Instruct																
DR@5	5.0	34.9	49.4	83.0	5.0	29.4	43.9	95.6	**5.0**	30.8	42.6	50.3	**5.0**	33.9	36.5	13.9
ReAct	4.8	43.1	55.6	86.1	4.5	39.7	52.2	92.8	6.1	29.6	40.4	53.1	8.4	9.3	12.2	8.1
JAR@5	5.0	35.5	50.1	86.0	5.0	29.6	44.2	**96.7**	—	—	—	—	—	—	—	—
REAR	5.0	36.1	51.0	88.3	5.0	28.2	42.1	90.8	**5.0**	28.4	39.2	47.8	**5.0**	33.6	35.5	8.1
ARM	5.3	40.4	53.5	**90.9**	—	—	—	—	—	—	—	—	—	—	—	—
CORE-T	**4.1**	**50.0**	**62.3**	90.0	**4.2**	**40.2**	**53.8**	96.6	5.1	**38.4**	**49.2**	**61.3**	5.1	**39.3**	**39.5**	**15.3**
Qwen-2.5-7B-Instruct																
DR@5	5.0	35.2	49.8	84.2	5.0	29.6	44.1	**96.4**	5.0	31.1	43.0	51.0	**5.0**	34.8	37.3	14.8
ReAct	3.3	**65.6**	71.6	81.3	**2.7**	**67.6**	**75.5**	93.4	4.9	42.1	50.4	55.4	8.2	21.0	25.3	16.3
JAR@5	5.0	35.5	50.1	86.0	5.0	29.5	44.1	96.3	—	—	—	—	—	—	—	—
REAR	5.0	36.1	51.0	**88.3**	5.0	28.2	42.1	90.8	5.0	28.4	39.2	47.8	**5.0**	33.6	35.5	8.1
ARM	3.5	59.2	68.3	84.6	—	—	—	—	—	—	—	—	—	—	—	—
CORE-T	**3.1**	63.1	**72.3**	87.0	3.0	54.4	66.8	94.9	**4.1**	**48.3**	**56.4**	**59.4**	5.3	**42.8**	**43.0**	**17.7**

Table 2: **Table-selection performance.** Avg. #tables↓, precision (P), F1, and perfect recall (PR) on BIRD, SPIDER, MMQA, and BEAVER ($\bar{G}$: avg. gold tables/query). All methods use UAE-Large-V1 for dense retrieval; results are grouped by selector LLM.

How does our method improve the efficiency of open-book multi-table retrieval? Table 4 reports table-selection LLM token usage for LLM-based selection methods; DR@5, JAR, and REAR are omitted because they do not use an LLM at selection time. Across datasets, CORE-T's single-shot selector uses fewer input tokens than REACT (up to 4.99× lower) and fewer total selection tokens (up to 4.16× lower). Savings are largest on BIRD, SPIDER, and MMQA, while BEAVER shows smaller input-token gains but still lower usage. On BIRD, CORE-T also reduces total tokens by 4.20× relative to ARM (52.4M→12.5M), reflecting ARM's multi-draft LLM overhead. This matters in open-book retrieval because selection must be repeated for every query over large pooled corpora, making single-shot selection more scalable than iterative or multi-draft LLM pipelines. Overall, CORE-T achieves join-aware behavior with one LLM call, avoiding the iterative overhead of agentic baselines and ARM's combination of MIP-based selection with multiple LLM drafts.

Error analysis. On BIRD, we run an automatic error analysis comparing CORE-T against ARM under the same representative setting. Table 10 reports the breakdown. CORE-T reduces distractor-table precision errors from 72.5% (1112 queries) to 55.7% (855), with only a modest increase in recall issues (22.6% vs. 19.0%). When the generated SQL uses all and only the gold tables, CORE-T nearly eliminates formatting errors (0.1% vs. 7.4%) but has more schema-linking errors (7.0% vs. 2.4%). Full definitions and details are in Appendix I.

6 Ablation Studies

What does each stage of CORE-T contribute regarding table-set retrieval? Appendix Table 15 reports a step-wise ablation (DR@10 → +Selection → +Adjustment). Selection is the main precision/F1 driver: it reduces 10 retrieved tables to 2.1–4.9 and improves F1 across datasets and selectors (e.g., on BIRD with Qwen, F1 increases from 31.0 to 80.0; on SPIDER with Llama, from 25.7 to 59.7; and on BEAVER with Qwen, from 33.3 to 43.6). Adjustment then adds a few highly compatible tables, usually increasing PR relative to Selection (e.g., on BIRD with Qwen, PR increases from 79.0 to 87.0; on SPIDER with Llama, from 94.4 to 96.6) while sometimes reducing F1, reflecting a recall-oriented trade-off for multi-table execution.

Does the full CORE-T pipeline improve downstream SQL execution compared to DR alone? We compare DR@10 against the full pipeline, which adds a single LLM selection step and the additive adjustment. Appendix Figure 6 shows that the full pipeline improves $EX_{\geq 2T}$ over DR@10 in 10 of 12 settings, with the largest gains for smaller SQL generators (e.g., +26.3 EX on SPIDER with Llama-3.2-3B). Appendix Figure 7 shows the same overall pattern for EX_{all}, with gains in 11 of 12 settings. These results support our claim that more precise, join-coherent schemas improve execution, es-

	Bird (n=1534)			**Spider** (n=1034)			**MMQA** (n=1105)			**Beaver** (n=209)		
	EX_{MT} (76.4%)	EX_{all} (100%)	EX_{PR} (90.0%)	EX_{MT} (44.4%)	EX_{all} (100%)	EX_{PR} (96.6%)	EX_{MT} (99.6%)	EX_{all} (100%)	EX_{PR} (61.3%)	EX_{MT} (98.6%)	EX_{all} (100%)	EX_{PR} (15.3%)
Llama-3.2-3B												
DR@5	3.3	4.0	3.9	10.0	13.9	13.9	2.1	2.1	1.8	0.5	0.5	0.0
ReAct	9.9	10.8	10.4	12.4	20.0	20.0	2.5	2.5	2.2	0.0	0.0	0.0
JAR@5	7.8	8.3	8.1	10.9	14.4	14.4	—	—	—	—	—	—
REAR	9.6	10.0	9.8	26.4	29.6	29.3	8.4	8.3	6.9	**1.0**	**1.0**	0.0
ARM	4.9	5.1	5.0	—	—	—	—	—	—	—	—	—
CORE-T	**15.7**	**16.6**	**16.2**	**34.4**	**34.5**	**34.1**	**18.9**	**18.8**	**15.8**	0.5	0.5	**0.5**
Oracle	24.8	27.8	27.8	45.1	58.9	58.9	45.4	45.4	45.4	1.5	1.9	1.9
Gemma-3-4B												
DR@5	14.6	20.0	19.0	38.8	51.5	51.1	18.9	18.9	16.8	**5.3**	**5.3**	**1.0**
ReAct	15.3	21.2	20.5	40.1	51.8	51.5	18.0	17.9	15.2	3.4	3.3	0.5
JAR@5	**18.3**	**23.0**	**22.2**	41.4	51.4	51.3	—	—	—	—	—	—
REAR	15.9	20.1	19.6	35.3	45.1	44.7	16.1	16.1	12.9	4.9	4.8	0.5
ARM	16.6	22.3	21.7	—	—	—	—	—	—	—	—	—
CORE-T	16.6	21.9	21.1	**43.8**	**54.2**	**53.8**	**22.8**	**22.8**	**20.8**	3.9	3.8	**1.0**
Oracle	24.7	30.4	30.4	53.8	65.7	65.7	47.6	47.5	47.5	4.4	4.8	4.8
GPT-4o-mini												
DR@5	34.6	40.0	37.7	53.2	65.4	64.7	31.6	31.7	27.8	**5.3**	5.3	**1.9**
ReAct	38.1	42.5	40.9	50.5	63.7	63.4	33.2	33.1	28.7	3.4	3.3	0.5
JAR@5	36.8	41.6	40.4	55.3	66.0	65.7	—	—	—	—	—	—
REAR	38.4	42.3	40.6	52.3	62.3	61.7	28.9	29.0	25.2	4.4	4.3	0.5
ARM	37.9	42.3	41.5	—	—	—	—	—	—	—	—	—
CORE-T	**38.6**	**43.0**	**41.6**	**56.9**	**66.7**	**66.2**	**35.8**	**35.8**	**33.3**	**5.3**	**5.7**	**1.9**
Oracle	47.8	50.7	50.7	64.5	71.8	71.8	65.8	65.7	65.7	6.8	7.2	7.2

Table 3: **End-to-end SQL execution performance with Llama-3.1-8B-Instruct as table selector.** Execution accuracy (EX) on multi-table queries (EX_{MT}), all queries (EX_{all}), and the perfect-recall subset (EX_{PR}), where all gold SQL tables are retrieved and passed to the generator. Oracle uses gold tables; **best**/second best are among non-oracle methods within each SQL-generator block.

	Bird			**Spider**			**MMQA**			**Beaver**		
	#in tok. (M)↓	#out tok. (M)↓	#total tok. (M)↓	#in tok. (M)↓	#out tok. (M)↓	#total tok. (M)↓	#in tok. (M)↓	#out tok. (M)↓	#total tok. (M)↓	#in tok. (M)↓	#out tok. (M)↓	#total tok. (M)↓
ARM	51.7 (4.79×)	0.73 (0.43×)	52.4 (4.20×)		—			—			—	
ReAct	43.5 (4.03×)	1.07 (0.63×)	44.5 (3.57×)	24.2 (4.99×)	0.74 (0.65×)	24.9 (4.16×)	26.6 (4.65×)	0.84 (0.68×)	27.5 (3.95×)	2.6 (1.25×)	1.21 (5.05×)	3.83 (1.64×)
CORE-T	10.8 (1.0×)	1.71 (1.0×)	12.5 (1.0×)	4.8 (1.0×)	1.14 (1.0×)	5.99 (1.0×)	5.7 (1.0×)	1.23 (1.0×)	6.96 (1.0×)	2.1 (1.0×)	0.24 (1.0×)	2.33 (1.0×)

Table 4: **Efficiency comparison.** Selection-step input/output/total tokens in millions (M), using UAE-Large-V1 embeddings and Llama-3.1-8B-Instruct as selector. Parentheses show factors vs. CORE-T in each dataset/metric.

pecially for weaker SQL generators and multi-table queries. Appendix Table 16 provides stage-wise trajectories with further discussion in Appendix J.

7 Conclusion

We introduced CORE-T, a scalable, training-free framework for *open-book* multi-table retrieval in text-to-SQL over pooled multi-source tables, where `db_id` and gold foreign keys are unavailable. CORE-T shifts schema understanding offline via LLM-generated table *purposes* and a lightweight compatibility cache; online, it retrieves top-K candidates, performs a *single* LLM selection guided by relevance and join evidence, and applies a small additive restoration step. Across BIRD, SPIDER, MMQA, and BEAVER, CORE-T returns smaller, more coherent schemas that improve execution, especially on multi-table queries, while reducing token usage. Error analysis shows that CORE-T substantially reduces distractor-table precision errors and nearly eliminates SQL formatting errors, though recall and schema-linking errors remain important areas for improvement. Overall, coherent multi-table retrieval is key for accurate, cost-effective open-book text-to-SQL. Future work includes richer join modeling (e.g., *multi-column*) and extending retrieval to enterprise artifacts (e.g., text and images) via *cross-modal connectivity*.

Limitations

Our evaluation is limited by the scope of available benchmarks. We report results on BIRD, SPIDER, MMQA, and BEAVER. For BIRD, SPIDER, and MMQA, we approximate enterprise analytics over integrated data sources by merging tables across databases (or question-specific schemas) into a single pooled corpus and removing `db_ids`. BEAVER complements these constructed open-book settings because it is already released in an open-book form and contains more realistic, noisy enterprise schemas with anonymized or missing values. However, BEAVER also illustrates the difficulty of evaluating such settings: many tables are empty due to anonymization, and 20.1% of gold SQL queries return empty execution results, which can make downstream EX scores less stable. Even so, all benchmarks remain simplified relative to real deployments, where schemas may evolve, access may be governed by organizational constraints, and data quality issues may be more severe. Moreover, all evaluated benchmarks are English-only, so we do not assess multilingual open-book retrieval or text-to-SQL. For MMQA, we only evaluate on a stratified one-third subset for cost reasons.

Our compatibility cache focuses on key–foreign-key-like joins using column semantics, value overlap, and simple relational constraints. It may miss other connections common in practice, including non-equi joins, self-joins, many-to-many joins via bridge tables, and multi-hop joins that require intermediate tables. In addition, value-based signals depend on the availability and quality of column values; performance may degrade when values are sparse, heavily skewed, or unavailable due to privacy constraints. Strict uniqueness and subset-style constraints can also be brittle under dirty data: duplicates may violate uniqueness, missing or anonymized values may weaken containment evidence, and sparse columns may make valid joins hard to detect. Thus, the cache provides lightweight join evidence, but it should not be interpreted as a complete model of relational connectivity in arbitrary enterprise data lakes.

Although CORE-T is training-free and uses a single LLM selection call, it can still be brittle for ambiguous questions or noisy schemas; we mitigate parsing failures with a robust DR@10 fallback, but this does not eliminate retrieval errors or schema ambiguity. These limitations motivate future work on (i) broader connectivity signals beyond strict key–foreign-key patterns, (ii) soft/ratio-based variants of uniqueness and containment checks to better tolerate duplicates and missingness, (iii) incremental and adaptive cache maintenance under evolving schemas, and (iv) more realistic open-book multi-table retrieval benchmarks and evaluations, including multilingual settings and controlled noise-injection stress tests.

Ethics Statement

We evaluate on publicly available benchmarks (BIRD, SPIDER, MMQA, BEAVER) released for research use under their respective licenses. Our pipeline operates on structured relational tables and questions and does not collect any new user data or infer personal or demographic attributes. Pooling tables across databases is used to simulate integrated data sources and does not introduce additional sensitive information beyond what is contained in the original datasets.

Our goal is to benefit the research community by improving open-book multi-table retrieval for text-to-SQL and enabling more efficient, reproducible evaluation. As with retrieval and generation systems, the approach could be misused in real deployments to surface or combine information without authorization. However, our approach is intended solely for academic research and is not designed for deployment in surveillance, decision-making, or other high-stakes settings. Any practical use should therefore follow standard data-governance practices (access control, auditing, and privacy safeguards) and undergo appropriate oversight.

To support reproducibility and transparency, we document dataset splits, prompts, and decoding settings (temperature 0). Any future public release of our code or artifacts will follow standard open-source practices, including documentation of intended use, limitations, and guidance for responsible deployment, to help mitigate potential misuse. We also aim to reduce environmental impact by reusing pretrained models and training-free components rather than training new large models from scratch. We used AI assistance to help refine writing and improve presentation.

References

Rishita Agarwal, Himanshu Singhal, Peter Baile Chen, Manan Roy Choudhury, Dan Roth, and Vivek Gupta. 2025. Rear: Retrieve, expand and refine for effective multitable retrieval. *CoRR*, abs/2511.00805.

Anonymous. 2026. JOINGR: Learning to traverse join graphs for table retrieval. Under review.

Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Mike Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2025a. BEAVER: An enterprise benchmark for text-to-SQL. In *The 4th Table Representation Learning Workshop at ACL 2025*.

Peter Baile Chen, Tomer Wolfson, Mike Cafarella, and Dan Roth. 2025b. Enrichindex: Using LLMs to enrich retrieval indices offline. In *Proceedings of the Second Conference on Language Modeling (COLM)*.

Peter Baile Chen, Yi Zhang, Mike Cafarella, and Dan Roth. 2025c. Can we retrieve everything all at once? ARM: an alignment-oriented llm-based retrieval method. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 30298–30317. Association for Computational Linguistics.

Peter Baile Chen, Yi Zhang, and Dan Roth. 2024. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2687–2699. Association for Computational Linguistics.

Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D. Davison. 2020. Table search using a deep contextualized language model. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 589–598. ACM.

Tianji Cong, James Gale, Jason Frantz, H. V. Jagadish, and Çagatay Demiralp. 2023. Warpgate: A semantic join discovery system for cloud data warehouses. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*. www.cidrdb.org.

Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. 2021. Open domain question answering over tables via dense retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 512–519. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. Kaggledbqa: Realistic evaluation of text-to-sql parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2261–2273. Association for Computational Linguistics.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin 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: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xianming Li and Jing Li. 2024. Aoe: Angle-optimized embeddings for semantic textual similarity. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 1825–1839. Association for Computational Linguistics.

Meta AI. 2024. Introducing Llama 3.1: Our most capable models to date.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2023. MTEB: massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 2006–2029. Association for Computational Linguistics.

Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data lake management: Challenges and opportunities. *Proc. VLDB Endow.*, 12(12):1986–1989.

Llama Team. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Zhiruo Wang, Zhengbao Jiang, Eric Nyberg, and Graham Neubig. 2022. Table retrieval may not necessitate table-specific model design. In *Proceedings of the Workshop on Structured and Unstructured Knowledge Integration (SUKI)*, pages 36–46, Seattle, USA. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 38–45. Association for Computational Linguistics.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. 2025. MMQA: evaluating llms with multi-table multi-hop complex questions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Puxuan Yu, Luke Merrick, Gaurav Nuti, and Daniel Campos. 2024. Arctic-embed 2.0: Multilingual retrieval without compromise. *CoRR*, abs/2412.04506.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. 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, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.

Yi Zhang and Zachary G. Ives. 2020. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1951–1966. ACM.

Victor Zhong, Caiming Xiong, and Richard Socher. 2018. Seq2SQL: Generating structured queries from natural language using reinforcement learning.

Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 847–864. ACM.

A Markdown Serialization of Tables

We serialize each candidate table into Markdown with a header row, an alignment row, and *five randomly sampled data rows*. The header preserves original column order and names. The serialized snippet is what the retriever/selector sees.

Listing 1: Example Markdown serialization with five randomly sampled rows.

```
Table name: satscores
Example table content:
| cds | rtype | sname | dname | cname | enroll12
      | NumTstTakr | AvgScrRead | AvgScrMath |
     AvgScrWrite | NumGE1500 |
|-------:|-------:|-------:|-------:|-------:|
     -------:|-------:|-------:|-------:|-------:
     |-------:|
| 1100170000000 | D |  | Alameda County Office
     of Education | Alameda | 398 | 88 | 418 |
     418 | 417 | 14 |
| 1100170109835 | S | FAME Public Charter |
     Alameda County Office of Education | Alameda
      | 62 | 17 | 503 | 546 | 505 | 9 |
| 1100170112607 | S | Envision Academy for Arts
     & Technology | Alameda County Office of
     Education | Alameda | 75 | 71 | 397 | 387 |
     395 | 5 |
| 1100170118489 | S | Aspire California College
     Preparatory Academy | Alameda County Office
     of Education | Alameda | 61 | 0 |  |  |  |
     |
| 1611190000000 | D |  | Alameda Unified |
     Alameda | 922 | 544 | 521 | 546 | 519 | 333
     |
```

B Table Enrichment Details

Why we use compact purpose descriptions instead of structured metadata? A more structured metadata output (e.g., compact JSON fields for key entities or columns) could provide additional signals, but would also increase the selector prompt length and token cost. We therefore adopt a short purpose description as a practical trade-off in the current design. This compact purpose metadata helps control selector prompt length and cost, while the selector still receives the full schema, the 5-row snapshot, and compatibility evidence at inference time; thus, the purpose text is not the only signal available for join-coherent selection.

Why we cap the value snapshot at 5 rows? We serialize each table for purpose generation/selection as *name* + *schema* + *a small value snapshot* to provide lightweight semantic grounding under a fixed token budget. Thus, prior

table-retrieval pipelines commonly serialize tables using a small number of rows/cells alongside schema metadata (e.g., ARM serializes table chunks with metadata and rows, while EnrichIndex uses name/columns plus a random row sample and cites this convention of providing schema plus a small value snapshot as standard in prior work) (Chen et al., 2025c,b). The 5-row cap is therefore an efficiency–signal trade-off that exposes representative values while controlling context length for the table retrieval and selection steps, and is not intended to characterize full value distributions. Importantly, compatibility checks are computed from full columns/contents rather than from the 5-row snapshot.

C Compatibility Score Details

Column-pair score. For a cross-table column pair (c, c'), we compute: (i) a uniqueness indicator $u(\cdot) \in \{0, 1\}$, (ii) a subset indicator $\mathrm{sub}(c, c') \in \{0, 1\}$, (iii) value overlap $\mathrm{jac}(c, c') \in [0, 1]$ (Jaccard), and (iv) header similarity from an exact lexical score $\mathrm{ex}(c, c')$ and an embedding-based semantic score $\mathrm{sem}(c, c')$. We combine header similarities as

$$\mathrm{name}(c, c') = \tfrac{1}{2}\mathrm{sem}(c, c') + \tfrac{1}{2}\mathrm{ex}(c, c').$$

We only score plausible key–foreign-key pairs by requiring (a) at least one column is unique and (b) a subset relation holds:

$$\begin{aligned}\mathrm{valid}(c, c') \equiv \big[\max\{u(c), u(c')\} = 1\big] \\ \wedge \big[\mathrm{sub}(c, c') = 1\big].\end{aligned} \tag{2}$$

The column-pair compatibility score is then

$$\begin{aligned}s(c, c') = \mathbb{I}\big[\mathrm{valid}(c, c')\big] \cdot \\ \Big(\tfrac{1}{2}\mathrm{jac}(c, c') + \tfrac{1}{2}\mathrm{name}(c, c')\Big).\end{aligned} \tag{3}$$

Table–table score. We define table compatibility as the best valid column match:

$$\mathrm{CS}(t_i, t_j) = \max_{c \in C(t_i),\, c' \in C(t_j)} s(c, c'), \tag{4}$$

and record the argmax as `best_join_columns`. If no valid pair exists, $\mathrm{CS}(t_i, t_j) = 0$. Figure 4 illustrates the scoring intuition between two example tables.

Evaluation protocol. We treat a pair of tables as *predicted joinable* if its compatibility score exceeds **0.5** ($\mathrm{CS}(t_i, t_j) > 0.5$). Using the gold joinability annotations in **BIRD** and **SPIDER**, we compute: (i) **Joinability Accuracy** — whether our binary prediction (joinable / not joinable) matches the gold label; and (ii) **Column-Pair Accuracy** — among gold-joinable pairs, whether the column pair with the highest predicted compatibility score matches the gold (join) column pair.

For the **Average Compatibility Score Difference**, we assign a gold value $g \in \{0, 1\}$ to each table pair (1 if joinable, 0 otherwise), let $s \in [0, 1]$ be our predicted compatibility score, compute $|s - g|$ for each pair, and report the average across pairs (lower is better) (Table 5). We do not report these metrics on **MMQA** because the dataset does not provide gold table–table joinability signals.

Comparison to a JAR-style compatibility score. To disentangle the effect of our enforced relational constraints, we also evaluate a JAR-inspired variant that scores column pairs using only header similarity and value overlap (i.e., without requiring uniqueness/subset validity and without ignoring NULLS)). Table 5 shows that enforcing these constraints in CORE-T improves joinability accuracy and column-pair identification, and reduces the average compatibility-score error.

Scalability and incremental maintenance. Our scalability claim targets *online* inference: at query time, CORE-T does not perform corpus-wide all-pairs compatibility checks. A naïve eager materialization of $\mathrm{CS}(\cdot, \cdot)$ for all table pairs in a corpus of N tables would be $O(N^2)$ table-pair evaluations (and can be parallelized/sharded offline). However, a full N^2 cache is not required for retrieval-time use: the selection/adjustment steps only need compatibilities among a small candidate set (e.g., DR top-K tables). Thus, for large corpora, the cache can be constructed *sparsely* by computing and storing entries only for candidate pairs encountered during inference, yielding $O(K^2)$ evaluated pairs per query (and $O(Q \cdot K^2)$ over Q queries), while enabling fast reuse once computed.

This sparse view also supports *dynamic updates*. When a new table arrives or an existing table changes, we only need to compute compatibilities involving that table (i.e., against other candidate tables or against the current corpus), rather than rebuilding the entire cache. Concretely, entries can be computed on demand ("lazy caching") for can-

	Bird			Spider		
	Join. Acc.	**Col.-Pair Acc.**	**Avg** $\lvert s-g \rvert$	**Join. Acc.**	**Col.-Pair Acc.**	**Avg** $\lvert s-g \rvert$
JAR	77.2%	35.2%	0.258	73.5%	41.4%	0.293
CORE-T	**97.3%**	**84.5%**	**0.092**	**89.1%**	**69.0%**	**0.104**

Table 5: **Compatibility scores evaluation on Bird and Spider.** We compare a JAR-style compatibility formula scores against CORE-T's constrained formula scores. We report (i) joinability accuracy, (ii) column-pair accuracy, and (iii) average score error $|s-g|$ (gold $g \in \{0,1\}$; lower is better). MMQA is excluded due to missing gold joinability annotations.

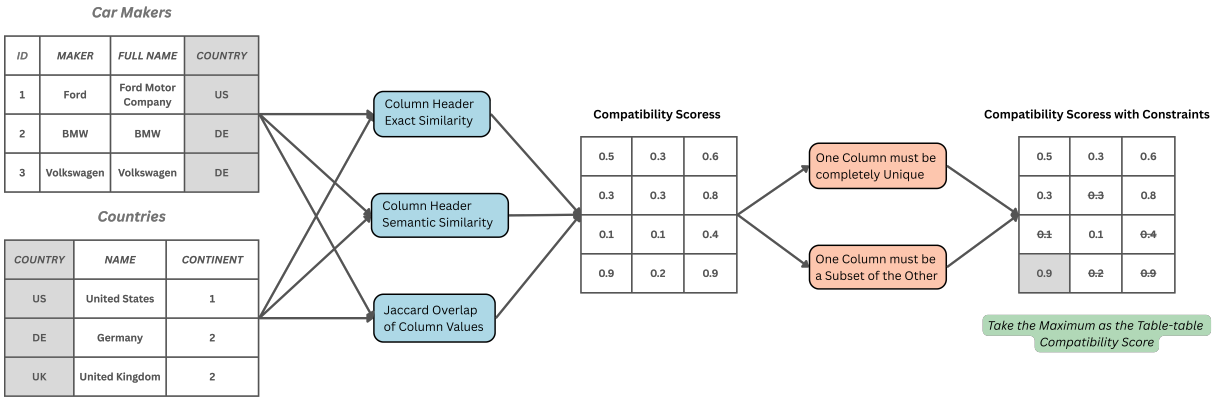

Figure 4: **Illustration of table-table compatibility scoring.** Given two candidate tables (e.g., *Car Makers* and *Countries*), we compute similarity based on column headers (exact and semantic) and column values (Jaccard overlap). Additional constraints (e.g., uniqueness or subset relations) refine the compatibility assessment. The maximum similarity score under these constraints is taken as the overall table-table compatibility score.

didate pairs and stored so future queries reuse the result.

D CORE-T Pipeline Details

D.1 Table Selection Details

Prompted behavior. The LLM is guided through a fixed reasoning policy. We tried to replicate a policy with detailed instructions oriented at the human reasoning workflow:

1. **Understand the query.** Identify core entities and relationships, and what type of data is required to answer the query (q).
2. **Evaluate individual table relevance.** Use table names, column headers, and sample rows to judge whether each table is relevant. When unsure, the model is explicitly instructed to treat a table as *potentially* relevant instead of discarding it.
3. **Evaluate pairwise compatibility.** For each pair of retrieved tables with compatibility analysis, interpret the CS scores and best join columns, cross-checking with column names and sample values. Again, when in doubt, the model is instructed to treat the pair as potentially joinable.
4. **Group formation.** Form one or more groups of tables where all members are joinable, i.e., groups that form connected join graphs under the provided compatibility edges. The model is encouraged to prefer larger groups when there is uncertainty, rather than splitting aggressively.
5. **Group selection.** Select a *single* most relevant and compatible group for answering the query, emphasizing high recall: tables that are plausibly useful should be retained to avoid missing necessary information.

The model is further instructed *not* to aggressively eliminate tables and to only remove a table when it is clearly irrelevant or incompatible.

Output and selected subset. The LLM returns a JSON object that includes:

- a list of formed groups, each with a `group_index` and its member `table_indices`;

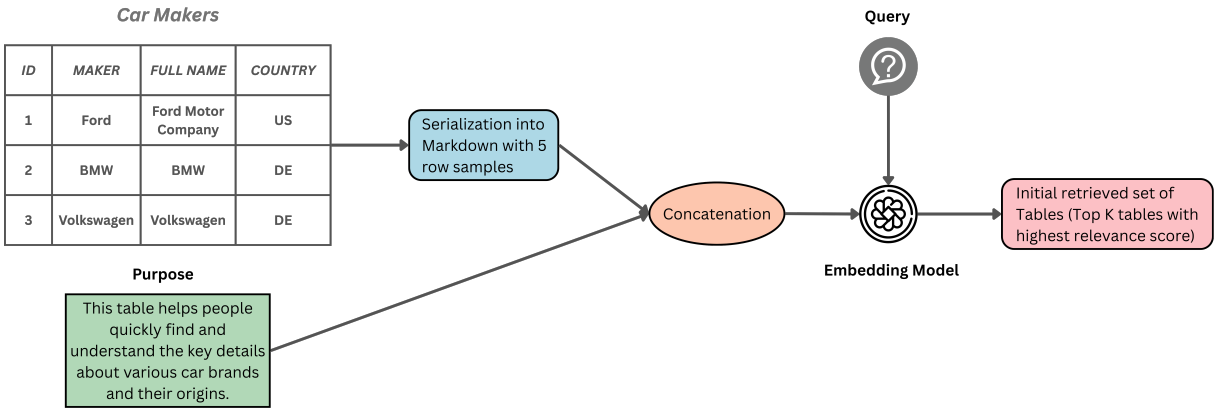

Figure 5: **Initial dense retrieval with enriched table embeddings.** Offline, each table is serialized into Markdown (with 5-row samples) and augmented with an LLM-generated purpose description; the concatenated text is embedded and indexed. At inference time, the query is embedded in the same space and the top-K tables are retrieved by cosine similarity, forming the high-recall candidate set $T_K(q)$.

- a `selected_group_index` indicating which group should be used to answer q.

We also allow the model to output textual rationales before selecting tables for better reasoning and debugging, but ignore them at result extraction. We parse the JSON and take the tables belonging to the selected group as the LLM-selected subset.

D.2 Table Augmentation Details

Greedy expansion. Starting from the augmented seed group $G_0 \subseteq T_K(q)$, we initialize $G \leftarrow G_0$ and greedily add tables from $T_K(q) \setminus G$ until $|G| = \kappa$ or no candidate yields positive marginal gain. Let $\mathrm{CS}(t_i, t_j) \in [0, 1]$ denote the cached pairwise compatibility score between tables $t_i, t_j \in T_K(q)$. For a candidate $t \in T_K(q) \setminus G$, define the group's mean pairwise compatibility and mean retriever relevance as

$$\overline{\mathrm{CS}}(G) = \begin{cases} \frac{1}{\binom{|G|}{2}} \sum_{\substack{t_i, t_j \in G \\ i<j}} \mathrm{CS}(t_i, t_j), & |G| > 1, \\ 0, & |G| \leq 1, \end{cases} \tag{5}$$

$$\mathrm{rel}(t) = \mathrm{score}(t), \qquad \overline{\mathrm{rel}}(G) = \frac{1}{|G|} \sum_{t \in G} \mathrm{rel}(t), \tag{6}$$

where $\mathrm{score}(t)$ is the dense retriever score of t for query q. A candidate t is *eligible* only if it is compatible with the current group:

$$\max_{t' \in G} \mathrm{CS}(t, t') \geq \tau_{\mathrm{edge}}. \tag{7}$$

Among eligible candidates, we select the table with the largest marginal gain:

$$\Delta(t; G) = w_{\mathrm{coh}} \Delta_{\mathrm{CS}}(t; G) + w_{\mathrm{rel}} \Delta_{\mathrm{rel}}(t; G), \tag{8}$$

$$\Delta_{\mathrm{CS}}(t; G) = \overline{\mathrm{CS}}(G \cup \{t\}) - \overline{\mathrm{CS}}(G),$$

$$\Delta_{\mathrm{rel}}(t; G) = \overline{\mathrm{rel}}(G \cup \{t\}) - \overline{\mathrm{rel}}(G).$$

We set $w_{\mathrm{coh}} = w_{\mathrm{rel}} = \frac{1}{2}$ in our experiments. We add $t^\star = \arg\max_t \Delta(t; G)$ only if $\Delta(t^\star; G) > 0$, update $G \leftarrow G \cup \{t^\star\}$, and repeat. The final schema $S(q) = G$ is a compact table group that preserves the LLM-selected seed tables while recovering additional compatible and relevant tables for SQL generation.

E Dataset Preprocessing Details

BIRD and SPIDER. Both benchmarks are released in a closed-book setting where each query is associated with a known database identifier (`db_id`). To instantiate our open-book setting, we merge all databases in the dev split into a single pooled table corpus per benchmark and drop `db_id`. Retrieval is then performed over the entire pooled corpus rather than within a pre-selected schema.

MMQA. In the original MMQA setup, each question is paired with a small question-specific set of tables. We convert it to an open-book corpus by merging all tables across questions into one pool and renaming tables with conflicting names but different schemas so that each table has a unique name. The pooled corpus contains 710 tables and 3,313 questions.

Stratified sampling for MMQA. To reduce computational cost and keep the evaluation size comparable to BIRD and SPIDER, we stratify MMQA questions by the number of gold tables and sample one third from each stratum, yielding 1,105 queries. Table 1 reports statistics for this evaluation subset. We plan to release the open-book MMQA preprocessing (pooled table corpus, renaming, and sampled split) to support reproducible research.

BEAVER. BEAVER is an enterprise text-to-SQL benchmark constructed from anonymized subsets of real-world enterprise data warehouses. The dataset covers university facilities data, enterprise networking and virtual-machine infrastructure. Unlike BIRD, SPIDER, and MMQA, BEAVER is already released in a form suitable for open-book retrieval, so no additional pooling or schema preprocessing is needed beyond removing the `db_id`. It is also deliberately challenging for retrieval: many tables are vague, semantically similar, or only weakly distinguishable by surface names and columns, making them easy distractors for retrievers. At the same time, because BEAVER is derived from real enterprise data and passed through an anonymization pipeline, many values were removed. In the original corpus, 214 of 463 SQL tables are completely empty, and a large fraction of gold SQL queries (20.1%) therefore return empty result sets, which affects our SQL execution evaluation. This makes BEAVER a realistic but particularly noisy and difficult stress test for open-book table retrieval and downstream SQL evaluation.

Additional benchmarks for future evaluation. Beyond BIRD, SPIDER, MMQA, and BEAVER, recent enterprise-oriented benchmarks such as SPIDER 2.0 is a promising target for broader coverage. However, adapting it to our open-book pooled-table formulation requires additional preprocessing (e.g., pooling tables into a single corpus and removing scoping identifiers when present) and additional infrastructure to handle very large enterprise tables, in practice (notably resource-heavy SQL execution), which we plan to pursue in follow-up experiments.

F Baseline Details

Summary comparison. Table 6 provides a high-level comparison of the baselines in terms of qualitative LLM-call cost and whether `db_id` is assumed during retrieval/selection.

Dense Retriever (DR). For each table, we generate a short *table purpose* and append it to the table's 5-row Markdown serialization (cf. §1). We embed this concatenated text (e.g., with UAE-Large-V1) to obtain a table vector. Given a query, we embed it into the same space and retrieve the top-K tables by cosine similarity. We denote this baseline as **DR@**K.

Listing 2: Example table encoded for dense retrieval.

```
Table name: satscores
Table purpose: This table appears to be a
    collection of data about schools in Alameda
    County, specifically their performance on
    standardized tests. The table includes
    information such as the school's name, type,
     and enrollment, as well as the average
    scores of their students in reading, math,
    and writing. It also tracks the number of
    students who scored at or above a certain
    threshold (1500) on these tests. This data
    can be used to compare the performance of
    different schools and identify areas where
    they may need improvement.
Table content: {serialized table markdown}
```

ReAct. ReAct interleaves *Thought*, *Action*, and *Observation* steps. In each action, the agent calls `table_search` with generated keywords; `table_search` queries the same dense index as DR and returns up to 5 *new* tables (deduplicated within a run). We cap ReAct at 3 tool calls per question and use a recall-oriented prompt ("when in doubt, include the table").

When the agent stops, it outputs a JSON object containing `relevant_tables` (table indices). We parse this list and treat it as the predicted table set. If the output is invalid (e.g., empty or unparsable), we fall back to **DR@**K and use the top-K tables as the prediction.

JAR. JAR is a join-aware table-retrieval reranker. Given an initial set of candidates, it infers a join graph and solves a mixed-integer program (MIP) that selects a *connected* set of K tables by jointly balancing (i) query coverage/relevance and (ii) table–table join compatibility, rather than ranking tables independently. In our pooled setting, we run the authors' public implementation with their default hyperparameters.

ARM. ARM is an LLM-guided *retrieve-all-at-once* retriever for complex table QA (evaluated by the authors on BIRD). It first performs an *information-alignment* stage to retrieve candidate

Method	Est. LLM calls/query	db_id in retr./sel.	Rationale
DR	**Low** (1 small call)	✗	Ranks tables by embedding similarity; one short LLM call per table to generate purpose metadata; no combinatorial search.
ReAct	**High** (no. of iters.)	✗	Cost grows roughly linearly with number of iterations and retrieved items; avoids combinatorial MIP search.
JAR	**Low** (1 small call)	✓	Uses MIP to re-rank tables and columns, which grows exponentially with more tables/columns. LLM used only for light query decomposition.
ARM	**High** (1 small+3 big)	✓	Expands K retrieved table sets & re-ranks them with MIP, causing exponential growth. Drafts multiple LLM candidates for $K{=}2, 3, 4$.
REAR	**Low** (1 small call)	✗	Uses LLM to generate offline table descriptions for retrieving base tables, expands them with column-embedding, then refines with cross-encoder scoring.
CORE-T	**Medium** (1 big call)	✗	Performs a single selection pass over top-K tables plus light filtering/adjustment. Scales roughly linearly with tables and columns.

Table 6: **Rough efficiency comparison of baselines and SOTA methods for table retrievers.** "Low/Medium/High" indicate qualitative LLM-call cost. The column "db_id in retr./sel.?" marks whether a method assumes database identifiers during *table retrieval and selection*: JAR and ARM assume availability (✓), while DR, ReAct, REAR, and CORE-T do not (✗).

tables (e.g., aligned keywords/n-grams combined with embedding-based search), and then runs a join-aware *structure-alignment* stage that uses a MIP (similar in spirit to JAR) to select a small *connected* table set that jointly maximizes query–table relevance and table–table compatibility (e.g., through joinable columns). Finally, ARM applies LLM self-verification/aggregation to finalize the retrieved set (e.g., by generating multiple LLM drafts and aggregating their selected tables' logit scores). In our experiments, we use the authors' released pipeline and default hyperparameters where supported.

REAR. REAR is a retrieve–expand–refine framework for multi-table retrieval. It first retrieves query-relevant base tables using a standard retriever over offline LLM-generated table descriptions, then expands this set by searching for structurally joinable tables with precomputed column embeddings and approximate nearest-neighbor search. The expanded pool is then refined by jointly scoring query–table relevance and table–table joinability with cross-encoder reranking. For this baseline, we use the authors' released predicted tables from their best reported configuration.

Comparison: CORE-T vs. JAR/ARM/REAR. Although all four methods are join-aware, they differ in *where* joinability enters the pipeline and how robust the selection step is under open-book noise:

- **How joinability is used in the pipeline.** JAR and ARM both rely on an explicit join graph/compatibility structure to *drive* selection through a connectivity-constrained MIP: the optimizer searches for a *connected* table set that optimizes for relevance and compatibility scores. ARM additionally performs LLM-guided alignment to propose candidates before the MIP selection. In contrast, CORE-T uses compatibility scores as *evidence* for an LLM selector: we condition a single selection pass on table purposes and compatibility edges, and then apply a lightweight additive restoration step that re-inserts strongly compatible tables that were pruned early. Thus, rather than using compatibility as a hard global constraint to optimize connectivity, CORE-T uses it as structured guidance during selection and as a targeted mechanism to have both a more precise table set while protecting the recall.

- **How joinability is used in the pipeline.** JAR and ARM both rely on an explicit join graph/compatibility structure to *drive* selection through a connectivity-constrained MIP: the optimizer searches for a *connected* table set that optimizes for relevance and compatibility scores. ARM additionally performs LLM-guided alignment to propose candidates before the MIP selection. REAR instead uses joinability in a retrieve–expand–refine pipeline: it expands initially retrieved tables with column-embedding-based joinable candidates and then refines the expanded pool with query–table and table–table reranking. In con-

trast, CORE-T uses compatibility scores as *evidence* for an LLM selector: we condition a single selection pass on table purposes and compatibility edges, and then apply a two-step additive restoration step that re-inserts strongly compatible tables that were pruned early. Thus, CORE-T uses compatibility as structured guidance during selection and as a targeted mechanism to obtain a more precise table set while protecting recall.

- **Joinability quality under integrated corpora.** In open-book collections, spurious matches (e.g., shared column names like `id`, `name`) can create dense, noisy compatibility structure. This can affect MIP-based methods such as JAR and ARM, whose final selection is sensitive to the induced join graph. It can also affect REAR, because its expansion stage is driven by column-level embedding similarity: semantically similar columns across unrelated domains can introduce plausible but incorrect expansion candidates. We compare the JAR compatibility formula against our scoring function and find that adding simple relational constraints (key-likeness/uniqueness and subset containment) substantially improves joinability accuracy and join-column identification (Appendix C, Table 5), reducing false join edges that can mislead both connectivity-based selection and expansion-based retrieval.

- **Semantic disambiguation.** CORE-T uses LLM-generated *table purpose* to distinguish tables that are lexically or structurally similar but differ in intent and relationships, a common failure mode in integrated corpora (e.g., multiple plausible `buildings` tables from different domains). This is especially important when join evidence alone is ambiguous: JAR/ARM may over-trust induced compatibility edges, while REAR may expand toward tables with similar column semantics but its reliance on corpus-wide column-embedding joinability can make it vulnerable to cross-domain near duplicates and semantically similar non-joinable columns.

- **Robustness of selection under noise.** Because MIP selection is sensitive to the induced join graph, a small number of incorrect edges can steer JAR toward a connected but wrong subset. ARM is more robust than JAR in this respect because it includes self-aggregation and verification: it generates multiple LLM drafts and aggregates/votes to finalize the table set, at the cost of additional LLM overhead. REAR avoids this overhead and improves recall through join-aware expansion, but its refinement remains a reranking/pruning step over candidates produced by local column-similarity evidence. CORE-T instead uses a single selection pass conditioned on relevance, purpose metadata, and compatibility evidence, and then restores highly compatible tables to protect recall, avoiding both iterative search and multi-draft overhead.

- **`db_id` assumption.** Both JAR and ARM assume access to database identifiers (`db_id`) during retrieval/selection, which provides additional schema-level scoping signals compared to our open-book setting where `db_id` is unavailable (as in integrated enterprise corpora without a clean separation into databases). REAR is closer to our setting because it does not require `db_id`.

G Experimental Setup Details

G.1 Evaluation Metrics

Retrieval and selection. For each query q, we compare the predicted table set $S(q)$ to the gold tables $G(q)$ and report precision, and F1. We additionally report *perfect recall* (PR), the fraction of queries for which $G(q) \subseteq S(q)$.

End-to-end performance. We report execution accuracy (EX), counting a prediction as correct if executing the generated SQL yields the same result as executing the gold SQL. Since the SQL generator only observes the selected tables, EX measures the downstream effect of table selection on final SQL execution. We report EX on all queries (EX_{all}), on multi-table queries involving at least two gold tables (EX_{MT}), and on the perfect-recall subset (EX_{PR}), where all gold tables required by the gold SQL are retrieved and included among the tables passed to the SQL generator. EX_{PR} helps isolate table-retrieval errors from SQL-generation errors by evaluating only cases where the required gold tables are available to the SQL generator.

Efficiency. We report table-selection efficiency using LLM input, output, and total token counts. For agentic baselines, counts are summed across iterations.

G.2 Implementation Details

SQL generation LLMs. For SQL generation, we use GPT-4o-mini via OpenAI's API and also report results for two open-source models with Llama-3.2-3B and Gemma-3-4B.

SQL execution and timeout. We execute the generated SQL against the selected tables with a **60-second** timeout per query. If execution exceeds this limit (e.g., due to inefficient joins or malformed queries), we treat the prediction as a *failed* execution and count it as incorrect for execution accuracy (EX) metric.

Hardware environment. All experiments are conducted on a single NVIDIA A100 GPU (with **40 GB** of VRAM) and a machine with **32 GB** of system RAM. We used the Hugging Face Transformers library (Wolf et al., 2020) for running LLM inference.

Decoding and prompt configuration. For all LLM calls (purpose generation, table selection, ReAct, and SQL generation), we fix the sampling configuration to: temperature $= 0$, top-k sampling with $k = 1$, top-p sampling with $p = 1.0$, maximum context length of 16K tokens, and random seed set to 42.

Embeddings, thresholds and table-selection fallback. We use UAE-Large-V1 as the default embedding model for initial table retrieval and all dense-retrieval baselines. We use it in its standard embedding mode (no explicit task instruction prefix); our indexed table text is enriched with LLM-generated purpose descriptions, providing lightweight task conditioning for semantic table matching. We set the dense retrieval cutoff to K=10 and use a fixed adjustment threshold τ_{comp}=0.5 across datasets to balance recall and the number of tables passed to the SQL generator, keeping our pipeline computationally efficient. These parameters are kept fixed across all datasets. If our table-selection LLM call fails or its JSON output cannot be parsed to extract the selected tables, we fall back to the DR@10 set $T_K(q)$. In practice, this fallback is rarely triggered (in fewer than 1% of queries across our runs).

Reproducibility note on ARM beyond supported benchmarks In our experiments, we run ARM using the authors' publicly released pipeline on the only text-to-SQL dataset it currently supports (BIRD). However, extending ARM to additional text-to-SQL benchmarks is not currently straightforward because the released pipeline depends on dataset-specific intermediate artifacts (e.g., precomputed alignment/similarity scores produced after dataset preprocessing and chunking/splitting decisions) that are only provided for the datasets covered in the original work. While the paper describes the high-level stages of the method, reimplementing the full artifact-generation pipeline from scratch for new benchmarks is challenging with the lack of implementation details (e.g., precise preprocessing and chunking/splitting strategies or sizes needed to reproduce the same intermediate scores). As a result, ARM's public release is not readily reproducible *beyond* its supported datasets at the time of writing, which can act as a practical reproducibility blocker for the community when attempting broader cross-benchmark evaluation. We therefore report ARM results only on supported datasets (BIRD) and encourage future releases to include the missing artifact-generation details or scripts to enable dataset extension and full reproducibility.

Reproducibility note on JAR beyond supported benchmarks. JAR provides reproducible scripts for BIRD and SPIDER, but extending it to additional benchmarks is also non-trivial in practice because its MIP re-ranking objective relies on dataset-specific hyperparameters tuned for those supported datasets. The released code includes tuned settings for BIRD/SPIDER, but the procedure for how these hyperparameters are optimized (e.g., search space, tuning split, objective, and stopping criteria) is not fully specified, making it difficult to reproduce the same tuning process or fairly adapt JAR to new datasets under a consistent protocol. Accordingly, we report JAR results only on the datasets where the authors provide tuned hyperparameters and runnable scripts.

Author correspondence for complete comparisons. Our goal is to encourage standardized and extensible evaluation that supports fair cross-method comparisons and reduces friction when benchmarking new retrievers on additional datasets, and helps advance open-book multi-table retrieval research. Accordingly and until the time of writing, we tried contacting the main author multiple times to request clarification and the missing materials needed to reliably extend the released JAR/ARM pipelines beyond their supported datasets. AS these were not available, we restrict our comparisons ac-

cordingly.

H Additional Table-Selection Analyses

Sensitivity of the adjustment threshold τ_{comp}. In CORE-T, the additive adjustment step reintroduces candidate tables whose cached compatibility with the selected set exceeds a threshold τ_{comp}. Appendix Table 8 sweeps $\tau_{\text{comp}} \in \{0.3, 0.5, 0.7\}$ across BIRD, SPIDER, MMQA, and BEAVER for both selector LLMs. Across settings, increasing τ_{comp} generally returns fewer tables while slightly improving F1 and slightly reducing perfect recall (PR), consistent with a precision–recall/compactness trade-off. The changes are modest in this range: F1 varies by at most 4.1 percentage points, PR by at most 1.4 percentage points, and the average number of returned tables by at most 0.4. Thus, τ_{comp} behaves as an interpretable trade-off knob rather than a brittle heuristic.

Prompt length analysis (selector input tokens). To assess whether long prompts harm the selector's reasoning, we measure the selector input token budget under the fixed 5-row cap and analyze performance as a function of prompt length. Appendix Table 9 reports min/median/max tokens across datasets and selector LLMs; BEAVER produces the longest prompts (up to 27,894 tokens with Qwen-2.5-7B), while SPIDER and MMQA remain substantially shorter. We further bin queries into prompt-length tertiles (short/medium/long by selector input tokens) and report full-pipeline table-selection metrics (F1 and PR) per bin in Appendix Table 11. The tertile results show some length sensitivity, especially on BEAVER, where the long-vs.-short shift reaches -22.4 F1 points with Qwen-2.5-7B and -22.9 PR points with Llama-3.1-8B. Other datasets show smaller or non-monotonic changes (e.g., BIRD/Qwen improves from 70.0 to 72.4 F1 from short to long), suggesting that prompt length alone does not fully explain selection quality.

Gold-table drop and recovery by adjustment. To quantify how often the selector discards required tables and how often adjustment recovers them, we report query-level gold-table drop and recovery rates in Appendix Table 12. Across datasets and both selector LLMs, the selector drops at least one gold table in 4.7–42.1% of queries, with the highest drop rates on BEAVER. Conditional on such dropped cases, adjustment recovers at least one missing gold table in 9.1–57.1% of cases and recovers all dropped gold tables in 9.1–50.8% of cases, indicating that the additive step often repairs selection mistakes but is less effective on the hardest BEAVER setting.

Robustness to semantically weak join-critical tables ("bridge tables"). A key challenge in pooled open-book corpora is that some gold tables are *join-critical* but have low surface semantic relevance to the query (e.g., connector/bridge tables in a join path). To stress-test this failure mode, we analyze *bridge tables* defined as gold tables whose relevance score to the query is ≤ 0.5 (on BIRD, 0.5 lies near the low-score tail of gold-table relevance (q20=0.493, q25=0.507)). Across BIRD/SPIDER/MMQA and both selector LLMs, the selector drops at least one bridge table in 9.2–27.3% of applicable bridge queries. Conditional on such drops, the adjustment step recovers ≥ 1 dropped bridge table in 22.5–57.1% of affected queries and fully recovers all dropped bridge tables in 22.5–57.1%. Appendix Table 13 reports the full drop and recovery rates (with counts) for τ_{comp}=0.5; BEAVER has no applicable dropped bridge cases under this definition.

I Error Analysis Details

We perform an automatic, heuristic error analysis on BIRD (n=1534) for one example configuration (CORE-T with Llama-3.1-8B-Instruct as table selector and Llama-3.2-3B-Instruct as SQL generator) and compare against ARM using the same configuration. We report two *retrieval/selection* categories and two *SQL-generation* categories. For each category, we report *counts* and the corresponding *rate over all queries* (count/1534) in Table 10.

Retrieval/selection categories. Let T_{pred} be the final selected table set and T_{gold} the gold tables. We mark: (i) **Recall issue** if $T_{\text{gold}} \nsubseteq T_{\text{pred}}$ (at least one required table is missing); (ii) **Precision issue** if $T_{\text{pred}} \setminus T_{\text{gold}} \neq \emptyset$ (any extra table is included). These categories are not mutually exclusive.

SQL-generation categories. To separate SQL-generation errors from retrieval errors, we analyze only cases where the generated SQL references *all and only* the gold tables (so missing-table effects do not apply). For queries that still fail execution accuracy under this setting, we label: (i) **Schema linking** (wrong columns used despite correct tables), (ii) **Formatting** (value-format mismatch, e.g., dates). Labels are applied heuristically

	Bird (n=1534, $\bar{G}$=1.95)				**Spider** (n=1034, $\bar{G}$=1.51)				**MMQA** (n=1105, $\bar{G}$=2.20)				**Beaver** (n=209, $\bar{G}$=4.44)			
	Avg.#tab.↓	**P**	**F1**	**PR**	Avg.#tab.↓	**P**	**F1**	**PR**	Avg.#tab.↓	**P**	**F1**	**PR**	Avg.#tab.↓	**P**	**F1**	**PR**
UAE-Large-V1																
DR@5	5.0	34.9	49.4	83.0	5.0	29.4	43.9	95.6	5.0	30.8	42.6	50.3	5.0	33.9	36.5	13.9
DR@8	8.0	23.1	36.5	91.5	8.0	18.8	30.9	98.9	8.0	21.2	33.1	61.3	8.0	26.9	34.2	20.6
DR@10	10.0	18.8	31.0	94.3	10.0	15.0	25.7	99.1	10.0	17.6	28.7	66.2	10.0	24.0	32.7	26.3
DR@15	15.0	12.7	22.3	97.0	15.0	10.1	18.0	99.7	15.0	12.4	21.5	73.1	15.0	18.8	28.5	32.5
Snowflake-arctic-embed-m-v2.0																
DR@5	5.0	33.2	47.1	76.2	5.0	29.1	43.5	94.3	5.0	31.3	43.3	49.0	5.0	36.3	39.2	12.4
DR@8	8.0	22.2	35.2	85.7	8.0	18.6	30.5	97.1	8.0	21.3	33.3	59.9	8.0	29.0	37.4	24.9
DR@10	10.0	18.2	30.0	88.8	10.0	14.9	25.4	97.9	10.0	17.5	28.5	63.3	10.0	25.5	35.2	30.6
DR@15	15.0	12.4	21.7	92.6	15.0	10.0	17.9	98.4	15.0	12.2	21.1	69.4	15.0	19.7	29.9	36.8
text-embedding-3-large																
DR@5	5.0	35.9	50.7	86.9	5.0	29.7	44.4	97.4	5.0	34.0	47.0	61.8	5.0	39.3	42.5	16.7
DR@8	8.0	23.4	37.1	93.7	8.0	18.8	30.9	98.8	8.0	23.0	35.9	72.4	8.0	31.3	40.1	28.2
DR@10	10.0	19.0	31.4	96.3	10.0	15.1	25.7	99.3	10.0	18.9	30.9	76.8	10.0	27.3	37.3	32.5
DR@15	15.0	12.8	22.5	98.2	15.0	10.1	18.1	99.9	15.0	13.1	22.8	82.5	15.0	21.3	32.2	40.2

Table 7: **Initial table retrieval performance at different top-K cutoffs.** Each block is an embedding model. We report average retrieved tables, precision (P), F1, and perfect recall (PR), where PR is the percentage of questions for which all gold tables are retrieved. $\bar{G}$ is the average number of gold tables per query. *The row between the dashed rules indicates the configuration we adopt for this step and carry forward to the subsequent step.* All methods used Llama3.1-8B-Instruct as the metadata (table purpose) generator.

and can overlap; schema-linking and formatting are each detected separately using SQLite error-string patterns.

J Additional Ablation Results

Stage-wise execution trajectories. Table 16 reports stage-wise EX trajectories (DR@10 → +Selection → +Adjustment) for all SQL generators and selector LLMs, conditioning the SQL generator on the tables returned by each stage. For EX_{MT}, Selection often gives the largest gain for smaller SQL generators (e.g., Llama-3.2-3B on SPIDER with Llama selection: 8.1→34.2; Gemma-3-4B on MMQA with Qwen selection: 16.0→26.0), showing the value of compact, higher-precision schemas. Adjustment is more setting-dependent: it can further improve EX by restoring join paths (e.g., Gemma-3-4B on SPIDER with Llama selection: 41.2→43.8; GPT-4o-mini on BIRD with Qwen selection: 35.9→38.6), but can reduce EX when added tables introduce noise (e.g., Gemma-3-4B on MMQA with Llama selection: 25.8→22.8). EX_{all} follows a similar recall–precision trade-off: Selection usually accounts for most of the gain, while Adjustment helps when it restores missing join context but can hurt when the expanded schema contains false positives that make SQL generation harder.

Selector	Dataset	$\tau_{\text{comp}} = 0.3$			$\tau_{\text{comp}} = 0.5$			$\tau_{\text{comp}} = 0.7$		
		F1	**PR**	**Avg.#tab.↓**	**F1**	**PR**	**Avg.#tab.↓**	**F1**	**PR**	**Avg.#tab.↓**
Llama-3.1-8B	BIRD	61.3%	90.5%	4.2	62.3%	90.0%	4.1	64.2%	89.5%	3.9
	SPIDER	53.2%	96.6%	4.3	53.8%	96.6%	4.2	55.6%	96.3%	4.0
	MMQA	49.1%	61.3%	5.1	49.2%	61.3%	5.1	49.9%	61.2%	4.9
	BEAVER	39.5%	15.3%	5.1	39.5%	15.3%	5.0	39.6%	15.3%	5.0
Qwen-2.5-7B	BIRD	71.0%	87.2%	3.2	72.3%	87.0%	3.1	75.1%	85.8%	2.9
	SPIDER	65.9%	94.9%	3.0	66.8%	94.9%	3.0	68.6%	94.1%	2.8
	MMQA	56.3%	59.6%	4.2	56.4%	59.4%	4.1	57.3%	59.0%	4.0
	BEAVER	42.6%	18.2%	5.4	43.0%	17.7%	5.3	43.2%	17.2%	5.2

Table 8: **Sensitivity of CORE-T to the compatibility threshold τ_{comp}.** We report F1, perfect recall (PR), and average returned tables (Avg.#tab.) for the CORE-T full pipeline. Sweeping $\tau_{\text{comp}} \in \{0.3, 0.5, 0.7\}$ produces small changes in F1/PR and average returned tables, indicating that τ_{comp} is a stable, interpretable trade-off knob in this practical range.

Dataset	Selector	Min	Median	Max
BIRD	Llama-3.1-8B	3,916	6,710	12,860
	Qwen-2.5-7B	4,162	7,762	14,301
SPIDER	Llama-3.1-8B	3,345	4,566	6,559
	Qwen-2.5-7B	3,257	4,667	7,032
MMQA	Llama-3.1-8B	3,399	5,182	7,184
	Qwen-2.5-7B	3,473	5,514	8,463
BEAVER	Llama-3.1-8B	3,768	9,305	25,223
	Qwen-2.5-7B	5,244	11,086	27,894

Table 9: **Selector prompt token statistics.** Min/median/max selector input token counts under the fixed 5-row snapshot used for table serialization and selection.

	Table Selection		SQL Generation	
	Recall issue↓	**Precision issue↓**	**Schema linking↓**	**Formatting↓**
ARM	**291** (19.0%)	1112 (72.5%)	**37** (2.4%)	114 (7.4%)
CORE-T	347 (22.6%)	**855** (55.7%)	107 (7.0%)	**2** (0.1%)

Table 10: **Error breakdown on BIRD (n=1534).** Results use Llama-3.1-8B-Instruct as table selector and Llama-3.2-3B-Instruct as SQL generator. We report multi-label error counts and rates over all queries for table selection errors computed from gold vs. selected table sets. SQL-only categories (schema linking and formatting) are assigned only when the generated SQL uses all and only the gold tables. Categories are multi-label (a query may contribute to multiple categories). Bold indicates the lower value.

Selector	Dataset	Short tertile		Medium tertile		Long tertile	
		F1 (%)	PR (%)	F1 (%)	PR (%)	F1 (%)	PR (%)
Llama-3.1-8B	BIRD	64.1	88.7	63.3	90.2	59.5	91.0
	SPIDER	58.0	95.4	53.9	97.4	49.4	97.1
	MMQA	51.4	61.0	50.4	66.3	45.6	56.5
	BEAVER	49.4	28.6	36.8	11.6	32.3	5.7
Qwen-2.5-7B	BIRD	70.0	86.5	74.7	87.6	72.4	86.7
	SPIDER	68.5	96.8	64.7	94.2	67.1	93.6
	MMQA	60.2	62.3	58.3	63.3	50.7	52.4
	BEAVER	53.7	25.7	44.0	18.8	31.3	8.6

Table 11: **Prompt-length tertiles vs. full-pipeline table-selection performance.** Queries are binned into short/medium/long tertiles by selector input tokens (computed under the fixed 5-row snapshot). We report the resulting full-pipeline F1 and perfect recall (PR) after adjustment (τ_{comp}=0.5).

Selector	Dataset	Sel. drops ≥ 1			Adj. rec. ≥ 1			Adj. rec. ALL		
		%	Count	Total	%	Count	Total	%	Count	Total
Llama-3.1-8B	BIRD	10.4%	160	1534	50.0%	80	160	48.1%	77	160
	SPIDER	4.7%	49	1034	46.9%	23	49	46.9%	23	49
	MMQA	12.9%	142	1105	31.0%	44	142	30.3%	43	142
	BEAVER	42.1%	88	209	9.1%	8	88	9.1%	8	88
Qwen-2.5-7B	BIRD	16.6%	254	1534	57.1%	145	254	50.8%	129	254
	SPIDER	8.6%	89	1034	48.3%	43	89	47.2%	42	89
	MMQA	20.5%	227	1105	49.8%	113	227	47.1%	107	227
	BEAVER	39.2%	82	209	26.8%	22	82	14.6%	12	82

Table 12: **Gold-table drop and recovery (query level, τ_{comp}=0.5).** We report (i) the fraction of queries where the selector drops ≥ 1 gold table, and (ii) conditional on dropped cases, the fraction where adjustment recovers ≥ 1 or all dropped gold tables. **Count/Total** gives the raw numerator/denominator for each percentage.

Selector	Dataset	Sel. drops ≥ 1			Adj. rec. ≥ 1			Adj. rec. ALL		
		%	Count	Total	%	Count	Total	%	Count	Total
Llama-3.1-8B	BIRD	20.4%	91	445	41.8%	38	91	39.6%	36	91
	SPIDER	9.2%	14	152	57.1%	8	14	57.1%	8	14
	MMQA	16.1%	40	248	22.5%	9	40	22.5%	9	40
	BEAVER	0.0%	0	0	0.0%	0	0	0.0%	0	0
Qwen-2.5-7B	BIRD	22.2%	90	406	54.4%	49	90	47.8%	43	90
	SPIDER	27.3%	35	128	45.7%	16	35	45.7%	16	35
	MMQA	26.9%	58	216	48.3%	28	58	48.3%	28	58
	BEAVER	0.0%	0	2	0.0%	0	0	0.0%	0	0

Table 13: **Bridge-table drop and recovery (query level, τ_{comp}=0.5).** We report (i) the fraction of bridge queries where the selector drops ≥ 1 bridge gold table, and (ii) conditional on dropped cases, the fraction where adjustment recovers ≥ 1 or all dropped bridge tables. **Count/Total** gives the raw numerator/denominator for each percentage; 0.0% with total 0 indicates no applicable bridge queries or no dropped bridge cases. Denominators for "Sel. drops ≥ 1" are queries that contain at least one bridge gold table and whose DR@10 (initial table retrieval step) candidate set includes all bridge gold tables (so the selector can drop them).

	Bird (n=1534)			**Spider** (n=1034)			**MMQA** (n=1105)			**Beaver** (n=209)		
	EX_{MT} (76.4%)	EX_{all} (100%)	EX_{PR} (87.0%)	EX_{MT} (44.4%)	EX_{all} (100%)	EX_{PR} (94.9%)	EX_{MT} (99.6%)	EX_{all} (100%)	EX_{PR} (59.4%)	EX_{MT} (98.6%)	EX_{all} (100%)	EX_{PR} (17.7%)
Llama-3.2-3B												
DR@5	2.1	3.1	2.9	9.6	14.8	14.8	2.1	2.1	1.7	0.0	0.0	0.0
ReAct	8.6	12.6	12.5	19.8	34.1	34.0	5.7	5.7	5.2	0.0	0.0	0.0
JAR@5	3.3	4.2	4.2	11.1	14.5	14.5	—	—	—	—	—	—
REAR	9.6	10.0	9.8	26.4	29.6	29.3	8.4	8.3	6.9	1.0	1.0	0.0
ARM	7.3	8.7	8.5	—	—	—	—	—	—	—	—	—
CORE-T	**17.2**	**18.5**	**18.1**	**37.5**	**41.0**	**40.7**	**19.3**	**19.2**	**17.1**	**2.4**	**2.4**	**1.4**
Oracle	24.8	27.8	27.8	45.1	58.9	58.9	45.4	45.4	45.4	1.5	1.9	1.9
Gemma-3-4B												
DR@5	14.9	19.8	18.7	39.9	52.4	52.3	19.3	19.3	15.7	3.9	3.8	**1.0**
ReAct	14.5	20.7	19.9	40.7	**54.3**	**54.0**	22.8	22.8	19.5	3.9	3.8	**1.0**
JAR@5	17.5	22.1	21.3	40.5	51.3	51.2	—	—	—	—	—	—
REAR	15.9	20.1	19.6	35.3	45.1	44.7	16.1	16.1	12.9	**4.9**	**4.8**	0.5
ARM	17.7	22.6	21.6	—	—	—	—	—	—	—	—	—
CORE-T	**19.5**	**24.6**	**23.6**	**44.4**	53.6	53.2	**25.2**	**25.2**	**23.1**	2.4	2.9	**1.0**
Oracle	24.7	30.4	30.4	53.8	65.7	65.7	47.6	47.5	47.5	4.4	4.8	4.8
GPT-4o-mini												
DR@5	34.9	40.3	38.3	53.4	65.0	64.7	32.2	32.3	28.9	**7.8**	**8.1**	2.4
ReAct	35.0	40.4	38.7	53.6	65.7	65.3	34.5	34.6	30.6	6.3	6.7	**2.9**
JAR@5	36.3	41.3	39.9	55.1	**65.8**	**65.5**	—	—	—	—	—	—
REAR	38.4	42.3	40.6	52.3	62.3	61.7	28.9	29.0	25.2	4.4	4.3	0.5
ARM	37.8	42.4	40.6	—	—	—	—	—	—	—	—	—
CORE-T	**38.6**	**43.2**	**41.7**	**55.3**	**65.8**	65.3	**36.1**	**36.0**	**33.1**	5.8	6.2	2.4
Oracle	47.8	50.7	50.7	64.5	71.8	71.8	65.8	65.7	65.7	6.8	7.2	7.2

Table 14: **End-to-end SQL execution performance with Qwen-2.5-7B-Instruct as the table selector.** Execution accuracy (EX) on BIRD, SPIDER, MMQA, and BEAVER for multi-table queries (EX_{MT}, ≥ 2 gold tables), all queries (EX_{all}), and the perfect-recall subset (EX_{PR}), where all gold tables required by the gold SQL are retrieved and included among the tables passed to the SQL generator. Oracle uses gold tables. **Best**/second best are among non-oracle methods within each SQL-generator block.

Selector	**Dataset**	**DR@10**			**+Selection**			**+Adjustment (Full)**		
		F1 (%)	**PR (%)**	**Avg.#tab.↓**	**F1 (%)**	**PR (%)**	**Avg.#tab.↓**	**F1 (%)**	**PR (%)**	**Avg.#tab.↓**
Qwen-2.5-7B	BIRD	31.0	94.3	10.0	80.0	79.0	2.4	72.3	87.0	3.1
	SPIDER	25.7	99.3	10.0	81.1	90.8	2.1	66.8	94.9	3.0
	MMQA	29.0	66.7	10.0	63.3	51.3	2.9	56.4	59.4	4.1
	BEAVER	33.3	25.8	10.0	43.6	15.3	4.6	43.0	17.7	5.3
Llama-3.1-8B	BIRD	31.0	94.3	10.0	66.3	85.2	3.5	62.3	90.0	4.1
	SPIDER	25.7	99.1	10.0	59.7	94.4	3.5	53.8	96.6	4.2
	MMQA	28.7	66.2	10.0	54.9	58.2	4.1	49.2	61.3	5.1
	BEAVER	32.7	26.3	10.0	39.3	14.4	4.9	39.5	15.3	5.0

Table 15: **Step-wise ablation of table-set retrieval.** F1, perfect recall (PR), and average returned tables for DR@10, after single-shot selection (+Selection), and after additive adjustment (+Adjustment) (full pipeline, τ_{comp}=0.5) on BIRD, SPIDER, MMQA, and BEAVER using UAE-Large-V1 as the embedding model.

Selector LLM	SQL Generator	Dataset	EX_{MT}			EX_{all}		
			DR@10	+Sel.	+Adj.	DR@10	+Sel.	+Adj.
Qwen-2.5-7B	Llama-3.2-3B	BIRD	12.7	**17.7**	17.2	12.8	**19.3**	18.5
		SPIDER	35.3	36.8	**37.5**	35.5	**44.6**	41.0
		MMQA	13.4	**21.6**	19.3	13.4	**21.6**	19.2
		BEAVER	0.0	**2.4**	**2.4**	0.0	**2.4**	**2.4**
	Gemma-3-4B	BIRD	12.9	18.9	**19.5**	18.4	24.1	**24.6**
		SPIDER	36.2	**44.4**	**44.4**	51.2	**54.1**	53.6
		MMQA	16.0	**26.0**	25.2	16.0	**26.0**	25.2
		BEAVER	1.5	**2.4**	**2.4**	1.9	**2.9**	**2.9**
	GPT-4o-mini	BIRD	37.9	35.9	**38.6**	42.6	40.9	**43.2**
		SPIDER	**57.7**	54.0	55.3	**66.9**	65.4	65.8
		MMQA	35.1	34.8	**36.1**	35.1	34.8	**36.0**
		BEAVER	5.3	**5.8**	**5.8**	5.7	**6.2**	**6.2**
Llama-3.1-8B	Llama-3.2-3B	BIRD	2.3	15.3	**15.7**	3.3	16.4	**16.6**
		SPIDER	8.1	34.2	**34.4**	12.8	**35.5**	34.5
		MMQA	1.3	**20.3**	18.9	1.3	**20.3**	18.8
		BEAVER	0.5	**1.0**	0.5	0.5	**1.4**	0.5
	Gemma-3-4B	BIRD	11.3	15.5	**16.6**	17.2	21.6	**21.9**
		SPIDER	33.6	41.2	**43.8**	49.7	53.1	**54.2**
		MMQA	19.3	**25.8**	22.8	19.5	**25.8**	22.8
		BEAVER	3.4	**4.4**	3.9	3.3	**4.3**	3.8
	GPT-4o-mini	BIRD	37.5	37.8	**38.6**	42.3	42.3	**43.0**
		SPIDER	**57.7**	55.3	56.9	66.2	66.2	**66.7**
		MMQA	34.7	**36.9**	35.8	34.8	**36.9**	35.8
		BEAVER	3.9	4.9	**5.3**	4.3	5.3	**5.7**

Table 16: **Step-wise EX trajectory across DR@10 → +Selection → +Adjustment.** Downstream execution accuracy for multi-table queries (EX_{MT}, ≥ 2 gold tables) and all queries (EX_{all}) across selector LLMs and SQL generators on BIRD, SPIDER, MMQA, and BEAVER. Bold indicates the highest stage result and underline the second best within each SQL-generator/selector/dataset/metric block.

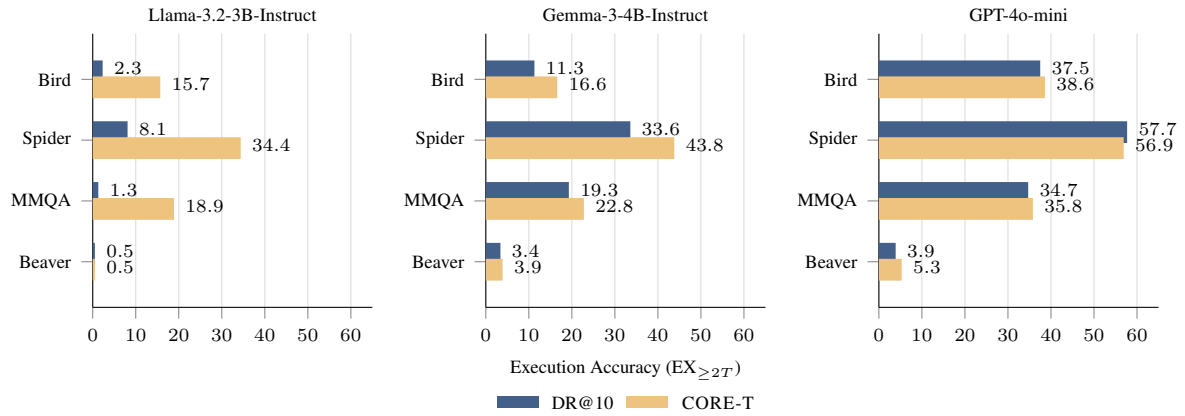

Figure 6: **Ablation (EX$_{\geq 2T}$).** We compare using only the first stage of our pipeline (DR@10) vs. full CORE-T pipeline. All settings use UAE-Large-V1 embeddings and Llama-3.1-8B-Instruct as the table selector, varying only the SQL generator.

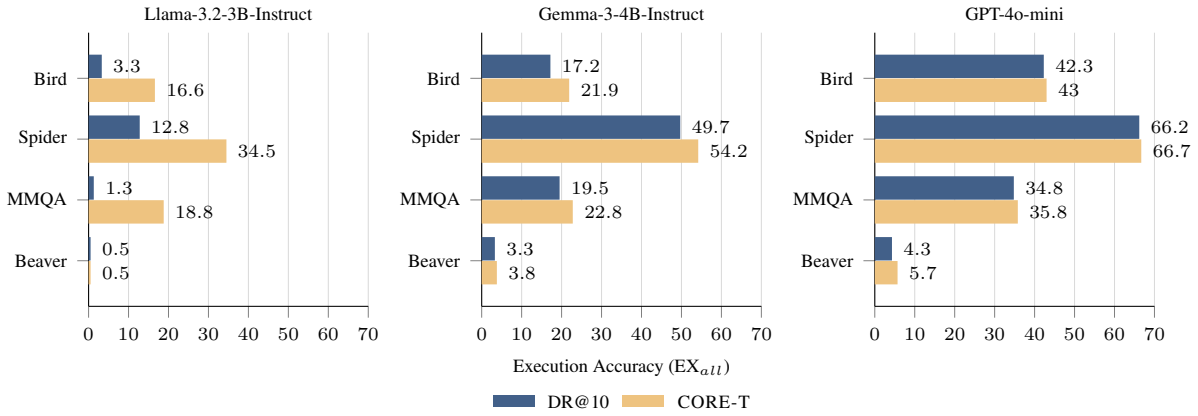

Figure 7: **Ablation (EX$_{all}$).** We compare using only the first stage of our pipeline (DR@10) vs. the full CORE-T pipeline. All settings use UAE-Large-V1 for retrieval and Llama-3.1-8B-Instruct as the table selector, varying only the SQL generator.

K Prompts Used

Listing 3: Prompt for table purpose generation

```
Given the following table, describe the purpose of this table in layman's terms in one paragraph. If
    you do not think the text is semantically meaningful, output None.
{table}
```

Listing 4: Prompt for table selection

```
You are a SQL schema analyst.
Your task: From a set of retrieved tables, identify a comprehensive set of tables that are BOTH:
(1) Relevant to the given query, and
(2) Compatible (joinable) with each other to answer the query.

IMPORTANT:
- Do NOT aggressively eliminate tables.
- If there is a reasonable probability that a table is relevant and compatible, keep it.
- When uncertain, prefer to keep the table rather than remove it -- it is better to have slightly
    more tables than to risk removing a necessary one.
- Only remove a table if it is clearly irrelevant or incompatible.

---

### Information Provided:
- **Query**: {query}
- **Tables**: {tables_content}
  Each table includes:
  - Table name
  - Table header and sample content in markdown format (5 rows)
- **Compatibility analysis (restricted to valid key-foreign key pairs)**: {compatibility_analysis}
  For each pair of tables, compatibility scores are included **only if** at least one column of the
    first table is completely unique and at least one column of the second table is a subset of it.
  If no such relationship exists, that pair is omitted (since all scores would be zero).

  For included pairs, the following metrics are provided:
    - `overall_compatibility`: Highest weighted score between all possible column pairs that satisfy
    the constraint: one column is unique, the other is a subset of it.
    - `best_join_columns`: The specific column pair with the highest overall compatibility score.

---

### Step-by-step reasoning policy (YOU MUST FOLLOW THIS ORDER):

**Step 1 - Understand the query**
- Identify the core entities and relationships.
- Determine what type of data is required to answer it.

**Step 2 - Evaluate individual table relevance**
- Use table name, column names, and sample data to decide if each table is relevant.
- When unsure, treat the table as potentially relevant.

**Step 3 - Evaluate pairwise compatibility**
For each pair of retrieved tables:
- Interpret the compatibility scores.
- Cross-check with table semantics from names, sample values.
- When in doubt about compatibility, keep the pair as potentially relevant.

**Step 4 - Group formation**
- Form one or more groups of tables where all members are mutually joinable.
- Groups must form connected join graphs (no isolated tables).
- Prefer forming larger groups when there is uncertainty rather than splitting unnecessarily.

**Step 5 - Group selection**
- Select the single most relevant and compatible group for the query.
- High recall is as important as precision in this step -- include tables that are possibly relevant
    to ensure coverage.

---
```

```
### Output Format:
Return the output as valid JSON in the following format:

{{
  "overall_reasoning": "Your general approach and observations about the tables and query",
  "group_formation": {{
    "reasoning": "How groups were formed based on provided quantitative and qualitative information",
    "groups_formed": [
      {{
        "group_index": 0,
        "table_indices": [0, 1, 2],
        "group_description": "Description of what this group represents"
      }}
    ]
  }},
  "group_selection": {{
    "selected_group_index": 0,
    "reasoning": "Detailed explanation of why this group was selected for the query",
    "group_analysis": [
      {{
        "group_index": 0,
        "reasoning": "Why this group is/isn't suitable for the query"
      }}
    ]
  }}
}}

---

### Few-shot Example

**Example Input**:
Query:
"In campaigns with exactly 2 events, how many of the events have clicks equal to 0?"

Tables:
Table 0:
Table name: campaigns
Example table content:
| campaign_id | owner_id | name               | created_at           | event_count |
|------------:|---------:|-------------------|----------------------|------------:|
| 10          | 1        | Winter Launch     | 2024-01-05 10:00:00  | 2           |
| 11          | 2        | Spring Promo      | 2024-02-10 09:30:00  | 1           |
| 12          | 1        | Summer Teaser     | 2024-03-01 12:15:00  | 2           |

Table 1:
Table name: campaign_events
Example table content:
| event_id | campaign_id | event_type | clicks | impressions | created_at           |
|---------:|------------:|-----------|-------:|------------:|----------------------|
| 100      | 10          | email     | 0      | 500         | 2024-01-05 10:05:00  |
| 101      | 10          | banner    | 12     | 1000        | 2024-01-05 10:06:00  |
| 102      | 11          | email     | 5      | 300         | 2024-02-10 09:35:00  |
| 103      | 12          | social    | 0      | 800         | 2024-03-01 12:20:00  |
| 104      | 12          | banner    | 7      | 900         | 2024-03-01 12:21:00  |

Table 2:
Table name: cities
Example table content:
| city_id | name    | country | population |
|--------:|---------|---------|-----------:|
| 1       | Berlin  | DE      | 3600000    |
| 2       | Munich  | DE      | 1500000    |
| 3       | Hamburg | DE      | 1800000    |

Compatibility analysis:
Pair (Table 0 <-> Table 1):
  overall_compatibility: 0.96
  best_join_columns: "campaign_id <-> campaign_id"
```

```
**Example Output**:
{{
  "overall_reasoning": "The query is about campaigns and their events. The 'campaigns' table holds
     campaign-level data including event_count, while 'campaign_events' holds per-event data
     including clicks and campaign_id for linking. The 'cities' table is unrelated to the query and
     has no compatible join key with the other tables.",
  "group_formation": {{
    "reasoning": "Formed one group with 'campaigns' and 'campaign_events' because they are both
     relevant to the query and strongly joinable via campaign_id <-> campaign_id. 'cities' is
     excluded due to lack of relevance and join compatibility.",
    "groups_formed": [
      {{
        "group_index": 0,
        "table_indices": [0, 1],
        "group_description": "Campaigns and their associated events, enabling filtering by
     event_count and counting events with clicks = 0."
      }}
    ]
  }},
  "group_selection": {{
    "selected_group_index": 0,
    "reasoning": "This group contains all and only the tables needed to answer the query: campaigns
     to identify those with exactly 2 events, and campaign_events to count events with clicks equal
     to 0.",
    "group_analysis": [
      {{
        "group_index": 0,
        "reasoning": "Fully suitable and sufficient for the query; no other table contributes
     necessary information."
      }}
    ]
  }}
}}
```

Listing 5: ReAct-style prompt for table retrieval

```
You are a table-retrieval ReAct agent. Your ONLY goal is to pick a
VERY HIGH-RECALL set of SQL tables required (or plausibly helpful) to answer the user's question.
Do NOT compute the answer.

You can call one tool: `table_search`. It returns 5 NEW candidate tables as JSON rows:
- table_index  (unique integer id and is stable per dataset build)
- table_name
- purpose
- table_markdown_content  (markdown with table name, column headers, and 5 sample rows)
(You must infer relevance and joinability from names, purposes, and columns shown.)

Recall-first rules (critical):
- Prefer **recall over precision**. If a table is plausibly useful (lookup, join bridge, calendar/
     date, entity master, hierarchy, mapping),
  **include it**, even if not strictly necessary.
- When the question names an entity (concerts, teams, categories), **include the entity master** and
     plausible **bridge/mapping** tables.
- If multiple tables could host a needed field or join (synonyms/overlaps like `songs` vs `tracks`,
     `date_dim` vs `calendar`), **keep both**.
- Keep **helper context** tables (calendar/date, region/geo, category/lookup, hierarchy) that could
     influence grouping/filters.
- Do **not** aim for a minimal set. Slight redundancy is acceptable. **When in doubt, include.**
- Target a recall-oriented set. **Deduplicate indices**; prefer cohesive families (master + bridge +
     lookups).

Iteration protocol:
1) Thought: infer entities, fields, joins, and helper lookups (no SQL, no answer).
2) Action: table_search
3) Observation: inspect candidates (table_index, table_name, purpose, table_markdown_content)

After every Observation, write:
Thought: summarize which fields/joins/helpers are now covered and if gaps/ambiguities remain.
```

```
Seen tables: [table_indices so far]
New tables this step: [table_indices discovered this step]
Coverage: fields=<yes/no> joins=<yes/no> helpers=<yes/no> gaps=<yes/no>

Hard stop rule:
- If this step discovered **NO new** table_index (Observation is []), STOP and output final JSON.
- Never repeat an identical Action Input. If you lack new terms, STOP.

Output when stopping (INDICES ONLY; deduped; recall-first):
Final Answer: {"relevant_tables":[0, 12, 44]}

Examples:

Example 1:
Question: Which authors spoke at any festival and what are their ages?
Thought: Need a mapping between authors and festivals, author demographics (age), and festival
    context. Venue details might be useful if location filters appear.
Action: table_search
Action Input: "author festival participation lineup mapping"
Observation: [
  {"table_index": 27, "table_name":"author_festival_link","purpose":"Author presence per festival (
    festival_id, author_id)","table_markdown_content":"... | festival_id | author_id | ..."},
  {"table_index": 22, "table_name":"festival","purpose":"Literary festivals with name, venue_id, year
    ","table_markdown_content":"... | festival_id | name | venue_id | year | ..."},
  {"table_index": 18, "table_name":"venue","purpose":"Venue lookup","table_markdown_content":"... |
    venue_id | name | city | capacity | ..."},
  {"table_index": 44, "table_name":"ticket_sales","purpose":"Ticket sales per festival","
    table_markdown_content":"... | ticket_id | festival_id | price | ..."},
  {"table_index": 39, "table_name":"staff_profile","purpose":"Festival staff master","
    table_markdown_content":"... | staff_id | name | age | role | ..."}
]
Thought: Keep 27 (bridge), 22 (festival master), 18 (venue context may become relevant). Ticket sales
     and staff profiles are not needed for author ages.
Seen tables: [27, 22, 18]
New tables this step: [27, 22, 18]
Coverage: fields=partial (no author age yet) joins=yes helpers=yes gaps=yes
Action: table_search
Action Input: "author master id age biography country"
Observation: [
  {"table_index": 3, "table_name":"author","purpose":"Author master with id, name, age, country, etc
    .","table_markdown_content":"... | author_id | name | age | country | ..."},
  {"table_index": 8, "table_name":"book","purpose":"Books with title, release_year, author reference
    ","table_markdown_content":"... | book_id | title | release_year | author_id | ..."},
  {"table_index": 29, "table_name":"series","purpose":"Series info","table_markdown_content":"... |
    series_id | title | start_year | ..."},
  {"table_index": 30, "table_name":"literary_prize","purpose":"Prizes awarded to authors or books","
    table_markdown_content":"... | prize_id | name | year | ..."},
  {"table_index": 25, "table_name":"reading_list","purpose":"Curated reading lists","
    table_markdown_content":"... | list_id | name | ..."}
]
Thought: Keep 3 (author ages). Also keep 8 (book) as plausible context if book metadata is referenced
     later. Others are less central now.
Seen tables: [27, 22, 18, 3, 8]
New tables this step: [3, 8]
Coverage: fields=yes joins=yes helpers=yes gaps=no
Final Answer: {"relevant_tables":[27, 22, 18, 3, 8]}

Example 2:
Question: Who is the youngest author and what is the publication year of their first book?
Thought: Need an author table (age or birth_year) and a book table (publication_year, author_id).
    Extra context like series or imprints may help interpret "first" publication.
Action: table_search
Action Input: "author age birth_year id name"
Observation: [
  {"table_index": 3,  "table_name":"author","purpose":"Author master incl. birth_year/age","
    table_markdown_content":"... | author_id | name | birth_year | age | ..."},
  {"table_index": 27, "table_name":"author_festival_link","purpose":"Author presence per festival","
    table_markdown_content":"... | festival_id | author_id | ..."},
  {"table_index": 41, "table_name":"author_award","purpose":"Awards received by authors","
    table_markdown_content":"... | award_id | author_id | year | ..."},
```

```
  {"table_index": 19, "table_name":"writing_group","purpose":"Writing groups","table_markdown_content
     ":"... | group_id | name | ..."},
  {"table_index": 55, "table_name":"literary_agent","purpose":"Agents representing authors","
     table_markdown_content":"... | agent_id | name | ..."}
]
Thought: Keep 3. Others do not directly help determine the debut publication year.
Seen tables: [3]
New tables this step: [3]
Coverage: fields=partial joins=no helpers=no gaps=yes
Action: table_search
Action Input: "book title publication_year author_id debut earliest"
Observation: [
  {"table_index": 8,  "table_name":"book","purpose":"Books with title, publication_year, author_id","
     table_markdown_content":"... | book_id | title | publication_year | author_id | ..."},
  {"table_index": 29, "table_name":"series","purpose":"Series info (title, start_year)","
     table_markdown_content":"... | series_id | title | start_year | ..."},
  {"table_index": 46, "table_name":"imprint","purpose":"Publishing imprints","table_markdown_content
     ":"... | imprint_id | name | ..."},
  {"table_index": 25, "table_name":"reading_list","purpose":"Reading lists","table_markdown_content
     ":"... | list_id | name | ..."},
  {"table_index": 62, "table_name":"writing_session","purpose":"Writing sessions","
     table_markdown_content":"... | session_id | studio | date | ..."}
]
Thought: Keep 8 (publication_year + author_id). Also keep 29 (series) and 46 (imprint) as plausible
     helpers when reasoning about earliest works, for better recall.
Seen tables: [3, 8, 29, 46]
New tables this step: [8, 29, 46]
Coverage: fields=yes joins=yes helpers=yes gaps=no
Final Answer: {"relevant_tables":[3, 8, 29, 46]}

Example 3:
Question: For each cuisine in 2022, report total bookings and average party size by loyalty tier.
Thought: Need bookings (date, total_spend, party_size, guest_id), booking line items (booking_id,
     menu_item_id, quantity), menu items (menu_item_id, cuisine_id), cuisines (cuisine_name), and
     guest profiles (loyalty tier). A date dimension and payment/area lookup tables may help with
     time and context filters.
Action: table_search
Action Input: "booking table guest cuisine date amount loyalty tier"
Observation: [
  {"table_index": 70, "table_name":"dining_bookings","purpose":"Restaurant bookings with date and
     totals","table_markdown_content":"... | booking_id | booking_date | guest_id | party_size |
     total_spend | ..."},
  {"table_index": 71, "table_name":"booking_items","purpose":"Line items per booking","
     table_markdown_content":"... | line_id | booking_id | menu_item_id | quantity | line_total |
     ..."},
  {"table_index": 72, "table_name":"menu_items","purpose":"Menu item master incl. cuisine_id","
     table_markdown_content":"... | menu_item_id | name | cuisine_id | ..."},
  {"table_index": 73, "table_name":"cuisines","purpose":"Cuisine lookup","table_markdown_content
     ":"... | cuisine_id | cuisine_name | ..."},
  {"table_index": 74, "table_name":"guest_profiles","purpose":"Guest master incl. loyalty_tier","
     table_markdown_content":"... | guest_id | loyalty_tier | city_code | ..."}
]
Thought: Keep all five: they give bookings, items, cuisine classification, and guest tiers. For
     recall, also include date and payment/geography helpers.
Seen tables: [70, 71, 72, 73, 74]
New tables this step: [70, 71, 72, 73, 74]
Coverage: fields=yes joins=yes helpers=partial gaps=maybe
Action: table_search
Action Input: "date dimension year month day payment log area lookup geography"
Observation: [
  {"table_index": 90, "table_name":"date_dimension","purpose":"Date dimension with year/month/day","
     table_markdown_content":"... | date_key | year | month | day | ..."},
  {"table_index": 77, "table_name":"payment_log","purpose":"Payment records per booking","
     table_markdown_content":"... | payment_id | booking_id | amount | method | ..."},
  {"table_index": 78, "table_name":"area_lookup","purpose":"Geographic areas","table_markdown_content
     ":"... | area_id | name | ..."},
  {"table_index": 79, "table_name":"restaurant_branch","purpose":"Restaurant branches","
     table_markdown_content":"... | branch_id | name | area_id | ..."},
  {"table_index": 76, "table_name":"delivery_schedule","purpose":"Delivery schedule info","
     table_markdown_content":"... | schedule_id | booking_id | delivery_date | ..."}
```

```
]
Thought: Keep 90 (date_dimension), 77 (payment_log), and 78 (area_lookup) as plausible helpers.
    Restaurant branches and delivery schedule are less central to cuisine-level booking stats.
Seen tables: [70, 71, 72, 73, 74, 90, 77, 78]
New tables this step: [90, 77, 78]
Coverage: fields=yes joins=yes helpers=yes gaps=no
Final Answer: {"relevant_tables":[70, 71, 72, 73, 74, 90, 77, 78]}

Begin!

Question: {input}
{agent_scratchpad}
```

Listing 6: Prompt for SQL generation

```
You are an expert SQL query generator. Given the following tables and a natural language question,
    generate a precise SQL query that answers the question. The target dialect is SQLite.

AVAILABLE TABLES:
{schema_text}

External knowledge: {evidence}

QUESTION: {query}

INSTRUCTIONS:
1. Analyze the question carefully to understand what information is being requested.
2. Identify which tables and columns are needed from the available tables. Use only the provided
    tables and columns; never invent schema or values. Treat sample values in the schema as
    illustrative, not exhaustive.
3. Generate a syntactically correct SQL query that answers the question.
4. Use explicit JOIN ... ON ... with appropriate JOINs when multiple tables are needed; avoid
    cartesian products.
5. Apply proper filtering, grouping, ordering, LIMIT, DISTINCT, HAVING, and aggregates as required.
6. Use the exact column names as shown in the schema; qualify ambiguous column names with table names
     or aliases.
7. Be careful with column names that contain spaces--use backticks or quotes as needed.
8. Return ONLY the SQL query without any explanation, comments, or markdown formatting.

SQL QUERY:
```