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

Anonymous ACL submission

Abstract

Realistic text-to-SQL workloads 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 simple additive adjustment step restores strongly compatible tables. Across BIRD, SPIDER, and MMQA, CORE-T improves table-selection F1 by up to 22.7 points while retrieving up to 42% fewer tables, improving multi-table execution accuracy by up to 5.0 points on BIRD and 6.9 points on MMQA, and using 4–5× fewer tokens than LLM-intensive baselines.

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). A common paradigm is to (i) retrieve relevant tables from a large corpus and then (ii) generate SQL conditioned on those retrieved tables. Since the SQL generator can only reason over what is retrieved, table retrieval is a critical bottleneck.

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., 2017; Herzig et al., 2021; Chen et al., 2020; Wang et al., 2022). In contrast, data-lake and semantic join discovery research focuses on *open-book* analytics over large, heterogeneous table collections, where the system must discover relevant and joinable tables without a predefined schema graph (Zhu et al., 2019; Nargesian et al., 2019; Zhang and Ives, 2020; Cong et al., 2023). This setting mirrors open-book text-to-SQL in integrated enterprise corpora, where tables are pooled across sources and database identifiers are unavailable.

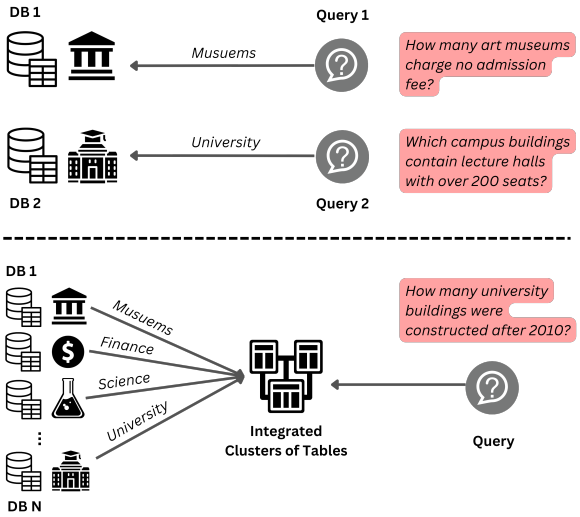

Figure 1: **Querying setups.** *Top:* Closed-book: each query targets a known database. *Bottom:* Open-book: queries must be answered over *integrated* clusters of tables spanning various domains pooled from multiple database schemas.

In real deployments, business intelligence workloads often require *multiple* tables to answer a single question (89% of models are multi-table; 4.8 tables on average) (Lai et al., 2025). Because organizations normalize data and split it across related tables, retrieval must return a *set* of tables that is both relevant and *joinable*. Figure 1 illustrates why this becomes challenging in heterogeneous table collections. In a scoped (closed-book) setting

(*top*), queries such as *"How many art museums charge no admission fee?"* can be answered within a clearly delimited database schema (e.g., museums vs. university). In contrast, in an open-book setting (*bottom*), tables from multiple domains are pooled together, and the retriever must identify the correct subset for a query like *"How many university buildings were constructed after 2010?"* amid many similarly named or semantically overlapping tables. For example, a table about `buildings` could appear in both the museums and university table collections of the corpus; when the corpus is integrated, the retriever must disambiguate these candidates based on their attributes and relationships, without relying on clean scoping signals such as database identifiers (e.g., `db_id=museums` vs. `db_id=university`).

This open-book setup introduces a *joinability* challenge that directly affects end-to-end correctness. First, multi-table queries are brittle: missing even one required table can break the join path and make SQL generation fail, even if other relevant tables were retrieved. Second, distractors are strong: irrelevant but similar-looking tables can appear highly relevant in isolation, yet lead to incompatible joins or spurious joins, confusing the SQL generator. Therefore, high recall alone is insufficient; retrieval must also produce compact, coherent, join-consistent table sets.

Existing multi-table retrievers only partially resolve this tension. Dense retrieval (DR) (Karpukhin et al., 2020) and a reranked DR variant (DRR) scale well to large corpora, but remain join-agnostic and can still surface many near-duplicate or spurious candidates. Agentic approaches such as ReAct (Yao et al., 2023) can iteratively expand evidence, but often require multiple LLM calls. Join-aware methods such as JAR and ARM (Chen et al., 2024, 2025c) incorporate relational structure via combinatorial selection, but rely on additional scoping assumptions (`db_id`) and/or may incur substantial inference overhead. We therefore seek a *scalable*, *join-aware* retriever for open-book table collections that returns a small, join-coherent schema slice for SQL generation. In our problem setting, we assume an open-book corpus of tables pooled across sources, with no `db_id` and no gold foreign keys. In summary, we contribute:

- **Training-free, scalable join-aware table retrieval.** We propose CORE-T, which enriches tables with LLM-generated purpose metadata and leverages an offline compatibility cache to approximate joinability, and performs a *single* LLM selection pass plus a lightweight additive adjustment step over initially retrieved top-K candidates.

- **Improved precision/F1 and efficiency.** We present an empirical comparison against DR, DRR, ReAct, and recent SOTA methods (JAR/ARM) under pooled multi-database evaluation on BIRD, SPIDER, and MMQA (Yu et al., 2018; Li et al., 2023; Wu et al., 2025). CORE-T improves the precision–recall balance for multi-table retrieval, returning more coherent table sets for SQL generation while reducing LLM usage (up to $\sim$5$\times$ fewer tokens than heavier multi-draft generation methods, e.g., ARM).

We will make the code and preprocessed data publicly available to support reproducibility.

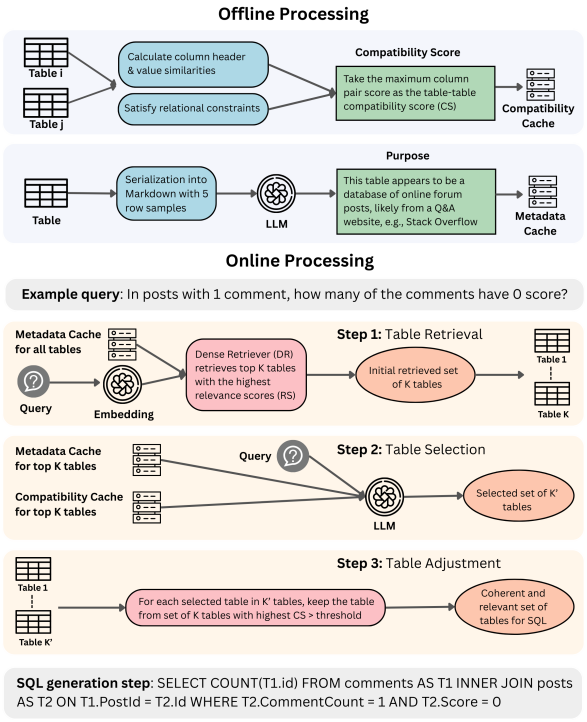

Figure 2: **Overview.** CORE-T combines offline table enrichment and compatibility caching with a lightweight online pipeline: DR, single-shot LLM selection, and a small additive adjustment.

2 Overview

Figure 2 provides a high-level overview of CORE-T. We delegate schema understanding into offline enrichment and caching, and keep online inference lightweight.

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

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

3 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.

3.1 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 into a 5-row Markdown representation and augmented with an LLM-generated *purpose description*. We sample 5 rows uniformly at random (without replacement) to provide a lightweight but representative snapshot, avoiding full-table inputs to the LLM while still exposing typical values and formats. The purpose description summarizes what the table contains and how it is typically used (e.g., key entities, attributes, and granularity), generated with a fixed prompt provided in Appendix H (cf. §3). We then embed the concatenation of the Markdown snapshot and its generated purpose to obtain a table vector $e_t = f_{\text{tbl}}(\texttt{Markdown+purpose})$, and store all vectors in a FAISS index for online retrieval.

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 a header embedding e_c. For each cross-table column 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 additional *relational constraints* to better mimic key–foreign-key joins: (iii) *uniqueness* (key-likeness) and (iv) *subset* (whether one column's values are contained in the other), 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 simple 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`) that can create noisy join graphs in pooled corpora. Full details and our exact scoring function are in Appendix B (Eq. 3, Figure 4). The table–table compatibility score is the best valid column match:

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

and we cache the corresponding argmax join columns. If no valid column pair exists, we set $\mathrm{CS}(t_i, t_j) = 0$. This cache is computed once offline and reused across queries. Compared to a JAR-style similarity scoring (without relational constraints and without ignoring NULLs), our scoring yields more accurate joinability estimates (Appendix B, Table 5).

3.2 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) an additive adjustment step that restores strongly compatible tables.

3.2.1 Table Retrieval

We first obtain a high-recall candidate set using DR over the enriched table embeddings created offline. An illustration of this step is provided in Appendix C (Figure 5).

Query embedding and scores. For each query q, we encode it with the same embedding model

used for tables, $e_q = f_{\text{tbl}}(q)$, and compute the relevance score to each table t as cosine similarity, $\text{RS}(q, t) = \cos(e_q, e_t)$.

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

3.2.2 Table Selection

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

LLM input and output. Given the top-K set $T_K(q) = \{t^{(1)}, \ldots, t^{(K)}\}$, 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 $\text{CS}(t_i, t_j) > 0$ (reporting `overall_compatibility` and `best_join_columns`; omitted pairs are treated as having no join edge). The LLM forms connected groups and selects one recall-oriented group, which we parse from JSON as $T_{K'}(q) \subseteq T_K(q)$. This subset is typically smaller and more coherent than $T_K(q)$, but remains recall-oriented due to conservative pruning instructions. However, the LLM may still over-prune highly compatible tables; we address this with a compatibility-driven adjustment step that restores strongly compatible but ignored tables from the original top-K set.

3.2.3 Table Adjustment

The adjustment step *recovers* tables that the LLM may over-prune via a purely additive procedure using the cached compatibility scores CS. Given the LLM-selected set $T_{K'}(q)$ and the original candidates $T_K(q)$, we add back strongly compatible tables from $T_K(q) \setminus T_{K'}(q)$.

Compatibility-based restoration. For each selected table $t \in T_{K'}(q)$, we find its most compatible unchosen neighbor in $T_K(q) \setminus T_{K'}(q)$, $t^\star = \arg\max_{t' \in T_K(q) \setminus T_{K'}(q)} \text{CS}(t, t')$, and add it if $\text{CS}(t, t^\star) \geq \tau_{\text{comp}}$. This step recovers tables that are strongly joinable with the selected group (e.g., bridge tables or complementary dimension tables) but were pruned by the LLM. Let $T_{\text{comp}}(q)$ denote the added tables; the final schema is $S(q) = T_{K'}(q) \cup T_{\text{comp}}(q)$. We set τ_{comp} to keep $|S(q)|$ small while preserving recall, and prompt the SQL generator only with the tables in $S(q)$ (schema and sample rows), rather than with the full corpus, reducing both context length and reasoning complexity.

Dataset	#DB	#Tab	Rows/T	Cols/T	#Q	MT%	$\bar{G}$
Bird (dev)	11	75	52,437	10.6	1,534	76.4	1.95
Spider (dev)	20	81	6,665	5.4	1,034	44.4	1.51
MMQA	1	710	1,414	5.7	1,105	99.6	2.20

Table 1: Dataset stats after pooling. Rows/T and Cols/T are averages per table; MT% is % queries with ≥ 2 gold tables; $\bar{G}$ is avg. gold tables/query.

4 Experiments

4.1 Experimental Setup

4.1.1 Datasets

We evaluate on three text-to-SQL benchmarks with substantial multi-table requirements: BIRD (Li et al., 2023), SPIDER (Yu et al., 2018), and MMQA (Wu et al., 2025). We use BIRD and SPIDER to enable direct comparison with prior join-aware retrievers (e.g., JAR/ARM), and include the newer MMQA as a larger, more multi-table-intensive stress test under integrated corpora. Following our *open-book* setting, we pool tables from multiple DBs (or question-specific schemas) into a single retrieval corpus per benchmark and remove database identifiers `db_ids`, following the assumption of open-book multi-table retrieval where it's not known to which database a table belongs. Table 1 reports dataset statistics after preprocessing. For MMQA, we evaluate on a stratified one-third subset (1,105 queries) for computational efficiency. We provide full preprocessing details and additional discussion on including new benchmarks (e.g., SPIDER 2.0 (Lei et al., 2025) and BEAVER (Chen et al., 2025a)) in our open-book pooled-table setting in Appendix D.

4.1.2 Baselines

We compare against dense retrieval (DR) (Karpukhin et al., 2020), a reranked

DR variant (DRR; DR followed by a cross-encoder reranker using Qwen-3-Reranker-8B (Zhang et al., 2025)), an agentic LLM retriever based on ReAct (Yao et al., 2023), and recent join-aware methods (JAR (Chen et al., 2024) and ARM (Chen et al., 2025c)), all evaluated in the same open-book setting.

DR. We retrieve top-K tables by cosine similarity between the query embedding and enriched table embeddings (Markdown + purpose), denoted as DR@K.

DRR. We augment DR with a cross-encoder reranker: we first retrieve a candidate list with DR, then rerank the candidates using Qwen-3-Reranker-8B and keep the top-K tables after reranking, denoted as DRR@K.

ReAct. We implement a ReAct-style retrieval agent that iteratively queries the same dense index for up to three steps and outputs relevant tables.

JAR. We benchmark JAR, 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. ARM is a multi-stage pipeline that uses LLM-guided alignment to retrieve candidates, then a join-aware MIP selection plus LLM self-verification by generating multiple LLM drafts and aggregating their selected tables' logit scores to finalize a connected table set.
Appendix E provides implementation details for each baseline and a thorough comparison; Table 6 summarizes their qualitative LLM-call cost, whether `db_id` is assumed during retrieval/selection, and the core *rationale* behind each approach.

How CORE-T differs from JAR/ARM in open-book corpora. In integrated corpora, join signals are often noisy and near-duplicate tables across domains make selection harder. JAR/ARM build an explicit join graph and use mixed-integer programming (MIP)-based optimization to select a small connected set, making their outputs sensitive to the induced compatibility structure; both also assume `db_id` for scoping, and ARM incurs additional LLM overhead via self-verification/aggregation. In contrast, CORE-T is designed for open-book corpora by using (i) LLM-generated *purpose* metadata to disambiguate candidates without `db_id`, (ii) modified compatibility scoring to provide more reliable join evidence, and (iii) a single-shot LLM selection followed by a lightweight additive restoration step to retain a high-recall, join-coherent table set.

4.1.3 Evaluation Metrics

We evaluate (i) table retrieval/selection quality, (ii) end-to-end text-to-SQL execution accuracy, and (iii) efficiency. We report standard set-based precision, recall, F1, and perfect recall (PR) for table selection, execution match (EM) for SQL generation (overall and by query table-count), and token/cost-based measures for LLM usage. Full metric definitions and cost computation are provided in Appendix F.1.

4.1.4 Implementation Details

LLMs and environment. For the table-selection step, we use two instruction-tuned LLMs: Llama-3.1-8B-Instruct (Meta AI, 2024; Llama Team, 2024) and Qwen-2.5-7B-Instruct (Qwen Team, 2024).

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

Baselines code. For JAR and ARM, we rely on the authors' publicly released code and default hyperparameters on the datasets they support: JAR provides reproducible scripts for BIRD and SPIDER, while ARM currently supports BIRD. Additional implementation details for each point are provided in Appendix F.2.

4.2 Results and Analysis

How does the choice of the embedding model and top-K tables affect the initial table retrieval performance and the balance between precision and recall? Table 8 (Appendix) compares embedding models and top-K cutoffs for the initial dense-retrieval stage, 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 (e.g., at K=10 on BIRD: F1 31.0 vs. 31.4; PR 94.3 vs. 96.3). Snowflake's model is competitive but slightly weaker in our setup (e.g., BIRD at K=10: F1 30.0, PR 88.8), and the gap to text-embedding-3-large is largest on MMQA (PR 66.2 vs. 76.8 at K=10). As expected,

	Bird (n=1534, $\bar{G}$=1.95)					**Spider** (n=1034, $\bar{G}$=1.51)					**MMQA** (n=1105, $\bar{G}$=2.20)				
	Avg #tab.↓	**P**	**R**	**F1**	**PR**	Avg #tab.↓	**P**	**R**	**F1**	**PR**	Avg #tab.↓	**P**	**R**	**F1**	**PR**
Llama-3.1-8B-Instruct															
DR@5	5.0	34.9	91.9	49.4	83.0	5.0	29.4	98.0	43.9	95.6	5.0	30.8	70.7	42.6	50.3
DRR@5	5.0	35.3	92.7	49.9	84.7	5.0	30.0	**99.5**	44.8	**98.7**	5.0	33.2	**76.1**	45.9	58.5
ReAct	4.8	43.1	93.0	55.6	86.1	4.5	39.7	96.4	52.2	92.8	6.1	29.6	72.0	40.4	53.1
JAR@5	5.0	35.5	92.9	50.1	86.0	5.0	29.6	98.5	44.2	96.7	—	—	—	—	—
ARM	5.3	40.4	**95.4**	53.5	**90.9**	—	—	—	—		—	—	—	—	—
CORE-T	**4.2**	**48.9**	94.9	**61.5**	90.2	**4.2**	**40.3**	98.0	**53.8**	96.0	**4.9**	**39.2**	75.8	**49.9**	**60.7**
Qwen-2.5-7B-Instruct															
DR@5	5.0	35.2	92.5	49.8	84.2	5.0	29.6	98.3	44.1	96.4	5.0	31.1	71.3	43.0	51.0
DRR@5	5.0	35.2	92.7	49.8	84.6	5.0	30.0	**99.5**	44.8	**98.8**	5.0	33.3	**76.4**	46.1	58.4
ReAct	3.3	**65.6**	90.5	**71.6**	81.3	**2.7**	**67.6**	96.7	**75.5**	93.4	4.9	42.1	73.8	50.4	55.4
JAR@5	5.0	35.5	92.8	50.1	86.0	5.0	29.5	98.3	44.1	96.3	—	—	—	—	—
ARM	3.5	58.7	92.3	67.9	84.2	—	—	—	—		—	—	—	—	—
CORE-T	**3.1**	61.9	**93.3**	71.4	**86.3**	2.9	54.3	97.0	66.8	94.5	**3.9**	**49.2**	75.0	**57.2**	**58.5**

Table 2: **Table-selection performance.** Avg. #tables↓, precision (P), recall (R), F1, and perfect recall (PR) on BIRD, SPIDER, and MMQA ($\bar{G}$: avg. gold tables/query). All methods use UAE-Large-V1 for dense retrieval; results are grouped by selector (Llama-3.1-8B-Instruct vs. Qwen-2.5-7B-Instruct).

increasing K improves recall/PR but reduces precision/F1 and increases context size; we therefore fix UAE-Large-V1 with DR@10 as a practical high-recall starting point and rely on the subsequent selection stage to improve precision. We use the same DR@10 retrieval as the starting candidate set for DRR.

How can we improve precision in open-book multi-table retrieval while preserving high recall? Table 2 shows that CORE-T improves the precision–recall trade-off over strong baselines. With Llama-3.1-8B-Instruct as selector, CORE-T achieves the best F1 on all three benchmarks while maintaining high recall (e.g., on BIRD: +8.0 F1 over ARM; +11.4 over JAR), and it also outperforms REACT across BIRD, SPIDER, and MMQA (e.g., +9.5 F1 on MMQA). These gains come with fewer retrieved tables (e.g., BIRD: 4.2 vs. 4.8/5.3 for ARM/REACT, and 5.0 for fixed-K baselines such as DR/DRR/JAR), yielding a more coherent schema slice. With Qwen-2.5-7B-Instruct, CORE-T remains competitive: it improves over ARM on BIRD (71.4 vs. 67.9 F1) and over REACT on MMQA (57.2 vs. 50.4 F1); on SPIDER, REACT is strongest, but CORE-T still substantially exceeds JAR (66.8 vs. 44.1 F1) with a compact set (2.9 tables on average). We note that SPIDER is older (2018) than BIRD/MMQA (2023/2025), and is therefore more likely to have appeared (directly or indirectly) in LLM pretraining or instruction-tuning data; such potential benchmark exposure or leakage can affect absolute performance and the relative ranking across methods. Overall, CORE-T improves precision and F1 while maintaining high recall. Compared to join-aware methods that assume access to `db_id` and use MIP (JAR/ARM), our approach achieves higher BIRD F1 for both selectors (Llama: 61.5; Qwen: 71.4) while selecting fewer tables on average.

How does table-selection quality impact downstream SQL execution accuracy, especially for multi-table queries? Table 3 shows that improved table selection increases execution accuracy, most clearly on multi-table questions ($\text{EM}_{\geq 2T}$). Across all three SQL generators, CORE-T achieves the best non-oracle $\text{EM}_{\geq 2T}$ on BIRD and MMQA, consistently outperforming all baselines, while remaining competitive on SPIDER (best for smaller generators and second-best with GPT-4o-mini, where DRR is stronger). On BIRD, CORE-T improves $\text{EM}_{\geq 2T}$ over ARM for all generators (e.g., 38.4 vs. 36.4 with GPT-4o-mini); on SPIDER, it improves over JAR/REACT (e.g., 55.8 vs. 53.2/51.6 with GPT-4o-mini); and on MMQA, it yields the strongest $\text{EM}_{\geq 2T}$ for all generators. Differences on single-table queries are smaller, as joins are not required. We include an *Oracle* setting that provides the SQL generator with the gold tables, serving as an upper bound that isolates retrieval errors and quantifies remaining head-

	Bird (n=1534)			**Spider** (n=1034)			**MMQA** (n=1105)		
	$EM_{=1T}$ (23.6%)	$EM_{\geq 2T}$ (76.4%)	EM_{all} (100%)	$EM_{=1T}$ (55.6%)	$EM_{\geq 2T}$ (44.4%)	EM_{all} (100%)	$EM_{=1T}$ (0.4%)	$EM_{\geq 2T}$ (99.6%)	EM_{all} (100%)
Llama-3.2-3B-Instruct									
DR@5	16.0	14.0	14.5	31.7	24.0	28.2	**0.0**	10.1	10.0
DRR@5	14.1	12.5	12.8	31.1	24.0	27.9	**0.0**	11.4	11.3
ReAct	16.0	16.6	16.5	**35.8**	25.1	31.0	**0.0**	11.3	11.2
JAR@5	14.9	16.6	16.2	32.7	25.5	29.5	—	—	—
ARM	17.1	17.7	17.6	—	—	—	—	—	—
CORE-T	**17.7**	**18.0**	**17.9**	**35.8**	**30.1**	**33.3**	**0.0**	**15.6**	**15.6**
Oracle	29.6	26.6	27.3	60.9	46.6	54.5	50.0	42.9	42.9
Gemma-3-4B-Instruct									
DR@5	36.7	10.8	16.9	**60.9**	30.7	47.5	25.0	12.5	12.6
DRR@5	32.9	0.09	14.3	59.8	29.2	46.2	**50.0**	13.2	13.3
ReAct	39.8	11.3	18.1	58.4	30.3	45.9	25.0	11.2	11.2
JAR@5	36.7	10.9	17.0	59.3	32.2	47.3	—	—	—
ARM	37.3	12.0	18.0	—	—	—	—	—	—
CORE-T	**41.2**	**13.1**	**19.7**	58.6	**35.7**	**48.5**	25.0	**15.8**	**15.8**
Oracle	44.2	17.0	23.4	77.0	41.4	61.2	50.0	38.1	38.1
GPT-4o-mini									
DR@5	60.2	33.4	39.7	**73.7**	53.8	64.9	25.0	27.3	27.3
DRR@5	60.5	34.5	40.6	72.3	**58.0**	**66.0**	**50.0**	32.0	32.0
ReAct	60.2	36.3	41.9	73.2	51.6	63.6	**50.0**	28.8	28.9
JAR@5	**61.6**	35.9	42.0	71.3	53.2	63.2	—	—	—
ARM	**61.6**	36.4	42.4	—	—	—	—	—	—
CORE-T	60.8	**38.4**	**43.7**	71.5	55.8	64.5	**50.0**	**32.7**	**32.8**
Oracle	64.9	46.5	50.8	84.3	63.0	74.9	50.0	63.4	63.3

Table 3: **End-to-end SQL execution performance.** Exact match (EM) on BIRD, SPIDER, and MMQA for single-table ($EM_{=1T}$), multi-table ($EM_{\geq 2T}$), and all queries (EM_{all}). Oracle uses gold tables. **Best**/second best are among non-oracle methods within each SQL generator block.

	Bird			**Spider**			**MMQA**		
	#in tok. (M)↓	#out tok. (M)↓	cost ($)↓	#in tok. (M)↓	#out tok. (M)↓	cost ($)↓	#in tok. (M)↓	#out tok. (M)↓	cost ($)↓
ARM	51.7 (4.8×)	0.73 (0.4×)	1.43 (1.08×)		—			—	
ReAct	43.5 (4.0×)	1.07 (0.6×)	1.37 (1.04×)	24.2 (5.0×)	0.74 (0.6×)	0.26 (1.04×)	26.6 (4.7×)	0.84 (0.7×)	0.34 (1.10×)
CORE-T	10.8 (1.0×)	1.71 (1.0×)	1.32 (1.00×)	4.8 (1.0×)	1.14 (1.0×)	0.25 (1.00×)	5.7 (1.0×)	1.23 (1.0×)	0.31 (1.00×)

Table 4: **Efficiency comparison.** Selection-step #in/#out tokens in millions (M) are measured when using UAE-Large-V1 embeddings and Llama-3.1-8B-Instruct as table selector. *Cost* reflects estimated GPT-4o-mini SQL-generation cost ($), computed using OpenAI's pricing for GPT-4o-mini: $0.15/1M input tokens and $0.60/1M output tokens. Parentheses show factors vs. CORE-T within each dataset/metric.

room (especially on MMQA, e.g., GPT-4o-mini: 32.7→63.4 $EM_{\geq 2T}$). Measured directly against *Oracle* on $EM_{\geq 2T}$, CORE-T consistently reduces the remaining headroom relative to baselines, indicating that more coherent table sets translate into more executable joins. DRR can only further narrow this gap on SPIDER with GPT-4o-mini. Overall, averaged across BIRD/SPIDER/MMQA, CORE-T closes ∼25.0% of the DR@5→Oracle headroom, and ∼19.2% of the Oracle headroom relative to a strong non-oracle baseline (ARM on BIRD, JAR on SPIDER, and ReAct on MMQA).

Effect of the table selector (Qwen vs. Llama) on downstream EM. Table 9 (Appendix) repeats the evaluation with Qwen-2.5-7B-Instruct as the selector and preserves the same qualitative conclusion: improving table-selection quality most reliably improves downstream EM, especially on multi-table queries.

How does our method improve the efficiency of open-book multi-table retrieval? Table 4 states efficiency via (i) LLM token usage in the *table-selection* stage and (ii) *SQL-generation* cost with GPT-4o-mini. We focus on methods using an LLM at selection step, so DR@5, DRR@5, and JAR are omitted. CORE-T is markedly more efficient than iterative and MIP-based baselines while producing higher-quality table sets. Compared to REACT, our single-shot selector reduces input tokens by 4.0–5.0× and total selection tokens by 3.6–4.2×. On BIRD, CORE-T also cuts total selection tokens by 4.2× relative to ARM (52.4M→12.5M), reflecting the overhead of ARM's multi-draft LLM pipeline. These savings modestly reduce downstream SQL-generation cost, consistent with smaller retrieved schemas. Overall, CORE-T achieves join-aware behavior with a *single* LLM call, avoiding the overhead of agentic baselines (ReAct) and the complexity of MIP plus multiple LLM draft generations (ARM).

Error analysis. We run an automatic error analysis on BIRD and compare our best CORE-T configuration against ARM. CORE-T mainly reduces distractor-table *precision* errors, lowering the precision-issue rate from 94.9% (ARM) to 90.0%, while recall issues (missing required tables) remain comparable. When the generated SQL uses *all and only* the gold tables, the remaining failures are dominated by *reasoning/calculation* errors (∼24%), with schema-linking and formatting issues occurring less often. Full definitions and breakdowns are in Appendix G (Table 7).

5 Ablation Studies

Does the full CORE-T pipeline improve downstream SQL execution compared to DR alone? We ablate CORE-T by comparing DR@10 (dense retrieval only) to CORE-T (full pipeline), which adds a single LLM selection step and the additive adjustment. Figure 3 reports $\text{EM}_{\geq 2T}$. The full pipeline improves over DR@10 in 8/9 settings and ties in one. Gains are largest for smaller generators (e.g., +8.2 EM on SPIDER with Gemma-3-4B), while GPT-4o-mini shows smaller but consistent improvements on BIRD and MMQA, supporting our central claim that more precise, join-coherent schemas improve multi-table execution in the open-book regime. We additionally report EM on all queries in Appendix (Figure 6).

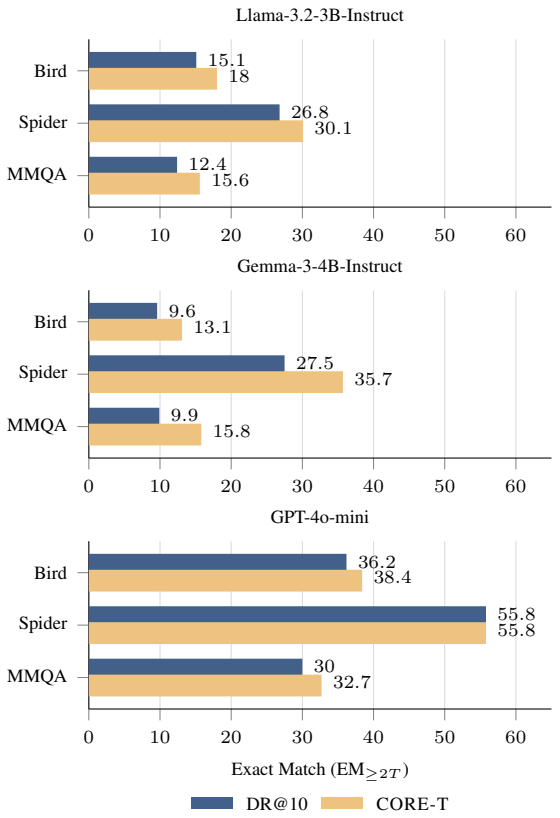

Figure 3: **Ablation ($\text{EM}_{\geq 2T}$).** We compare using only the first stage of our pipeline (DR@10) vs. full CORE-T pipeline. All settings use UAE-Large-V1 embeddings and Llama-3.1-8B-Instruct as the table selector, varying only the SQL generator.

6 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, and MMQA, CORE-T returns smaller, more coherent schemas that improve execution, especially on multi-table queries, while reducing token usage. Error analysis suggests gains come primarily from removing distractors, with remaining failures dominated by SQL reasoning. 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, and MMQA; to better approximate enterprise analytics over integrated data sources, we merge tables across databases (or question-specific schemas) into a single pooled corpus and remove `db_ids`. While this open-book construction is more realistic than closed-book evaluation, it still simplifies real-world deployments (e.g., noisier schemas, governance constraints, and evolving data). 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 representative rows; performance may degrade when values are sparse, heavily skewed, or unavailable due to privacy constraints.

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 failures in parsing with a robust DR@10 fallback. These limitations motivate future work on broader connectivity signals, stronger schema-only variants, and more realistic open-book multi-table retrieval benchmarks and evaluations, including multilingual settings.

Ethics Statement

We evaluate on publicly available benchmarks (BIRD, SPIDER, MMQA) 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

Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Michael Cafarella, Çagatay Demiralp, and Michael Stonebraker. 2025a. BEAVER: An enterprise benchmark for text-to-SQL. *arXiv preprint arXiv:2409.02038*.

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

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)*, pages 30298–30317, Vienna, Austria. 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)*, pages 2687–2699, Bangkok, Thailand. 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*, pages 589–598, Virtual Event, China. Association for Computing Machinery.

Tianji Cong, James Gale, Jason Frantz, H. V. Jagadish, and Çağatay Demiralp. 2023. WarpGate: A semantic join discovery system for cloud data warehouses. In *Proceedings of the 13th Conference on Innovative Data Systems Research (CIDR 2023)*, Amsterdam, The Netherlands.

Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian 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*, pages 512–519, Online. 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)*, pages 6769–6781, Online. Association for Computational Linguistics.

Eugenie Y. Lai, Yeye He, and Surajit Chaudhuri. 2025. Auto-prep: Holistic prediction of data preparation steps for self-service business intelligence. *Proc. VLDB Endow.*, 18(7):2212–2225.

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 (Volume 1: Long Papers)*, pages 2261–2273, Online. 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 *Proceedings of the Thirteenth International Conference on Learning Representations*.

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*, volume 36, Red Hook, NY, USA. Curran Associates, Inc. NeurIPS 2023 Datasets and Benchmarks Track.

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)*, pages 1825–1839, Bangkok, Thailand. Association for Computational Linguistics.

Llama Team. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783. Author list credited to Llama Team; detailed contributors listed in the paper appendix.

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

Niklas Muennighoff, Nouamane Tazi, Loic 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*, pages 2014–2037, Dubrovnik, Croatia. 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.

Qwen Team. 2024. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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, Clément Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. 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 *Proceedings of the Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Puxuan Yu, Luke Merrick, Gaurav Nuti, and Daniel Campos. 2024. Arctic-embed 2.0: Multilingual retrieval without compromise. *Preprint*, arXiv: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 Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Yi Zhang and Zachary G. Ives. 2020. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1951–1966, Portland, OR, USA. Association for Computing Machinery.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *Preprint*, arXiv:1709.00103.

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*, pages 847–864, Amsterdam, The Netherlands. Association for Computing Machinery.

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 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). Table 5 shows that enforcing these constraints in CORE-T improves joinability accuracy and column-pair identification, and reduces the average compatibility-score error.

C Table Retrieval and 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`;
- 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 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.

Additional benchmarks for future evaluation. Beyond BIRD, SPIDER, and MMQA, recent enterprise-oriented benchmarks such as SPIDER 2.0 and BEAVER are promising targets for broader coverage. However, adapting them to our open-book pooled-table formulation requires additional preprocessing (e.g., pooling tables into a single corpus and removing scoping identifiers

	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.

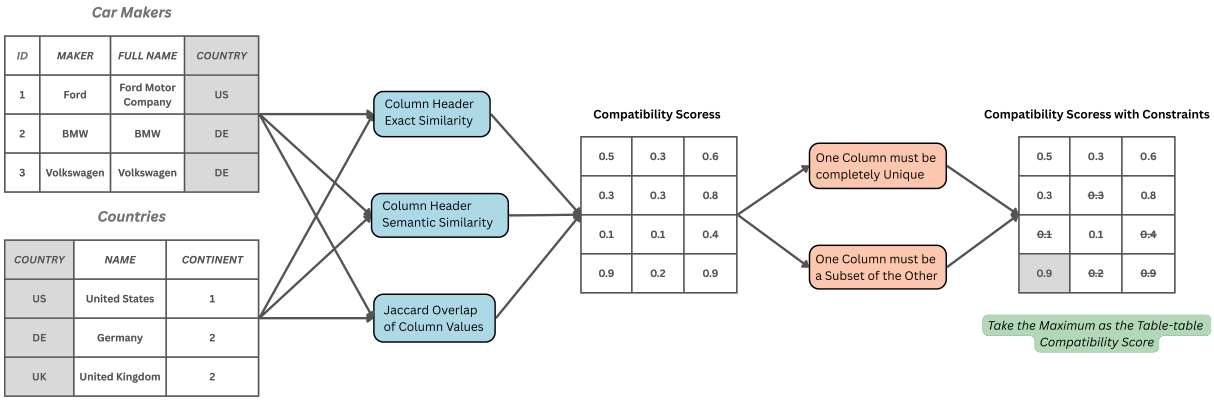

Figure 4: **Illustration of table-table compatibility scoring.** Given two candidate tables (e.g., *Car Makers* and *Countries*), we compute similarity based on column headers (exact and semantic) and column values (Jaccard overlap). Additional constraints (e.g., uniqueness or subset relations) refine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

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.

E 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}
```

Dense Retriever + Reranker (DRR). DRR augments dense retrieval with a cross-encoder reranker. We first retrieve an initial ranked candidate list using DR (same embeddings and table text as **DR@K**), then rerank the candidates with `Qwen3-Reranker-8B` by scoring each query–table pair and keeping the top-K tables after reranking. We choose `Qwen3-Reranker-8B` as it is a strong open-source reranking model reported to perform competitively across standard reranking benchmarks and is released under a permissive license. This baseline should improve relevance ordering.

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

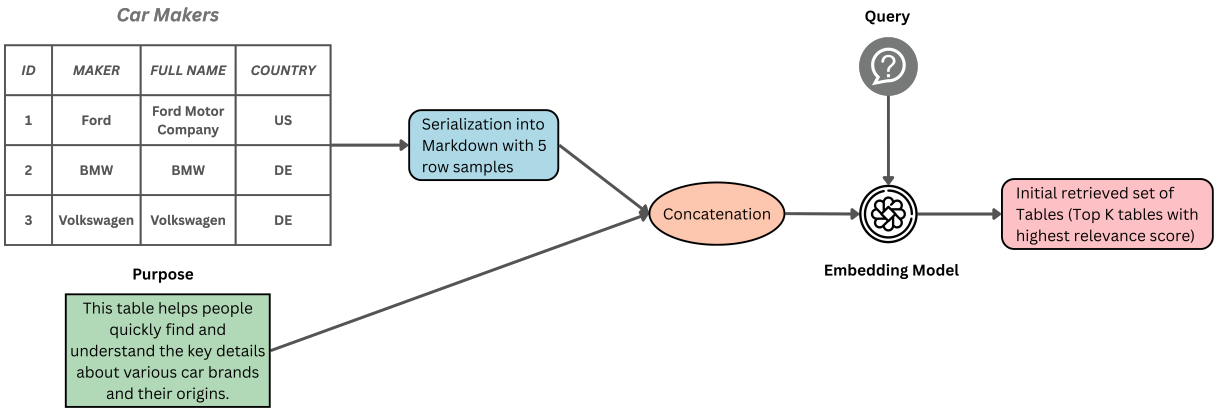

Figure 5: **Initial dense retrieval with enriched table embeddings.** Offline, each table is serialized into Markdown (with 5-row samples) and augmented with an LLM-generated purpose description; the concatenated text is embedded and indexed. At inference time, the query is embedded in the same space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

Method	Est. LLM calls/query	db_id in retr./sel.	Rational
DR	**Low** (1 small call)	✗	Ranks tables by embedding similarity; one short LLM call per table to generate purpose metadata; no combinatorial search.
DRR	**Low** (1 small call)	✗	Reranks the initial DR list with an open-source cross-encoder reranker to refine relevance ordering; no join optimization.
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 $K{=}2, 3, 4$.
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, DRR, ReAct and CORE-T do not (✗).

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 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.

Comparison: CORE-T vs. JAR/ARM. Although all three 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.

- **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. 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 B, Table 5), reducing false join edges that can mislead connectivity-based selection.

- **Semantic disambiguation.** CORE-T uses LLM-generated *table purpose* to distinguish tables that are lexically similar but differ in intent and relationships, a common failure mode in integrated corpora (e.g., multiple plausible `buildings` tables from different domains).

- **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 a self-aggregation and verification step: it generates multiple LLM drafts and aggregates/votes to finalize the table set. CORE-T instead uses a single selection pass conditioned on relevance 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).

F Experimental Setup Details

F.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, recall, 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 exact match (EM), counting a prediction as correct if executing the generated SQL (conditioned on the retrieved tables) yields the same result as the gold SQL execution result. We report EM on all queries and stratified by the number of gold tables (e.g., $=1$ vs. ≥ 2) to emphasize multi-table performance.

Efficiency. We measure (i) LLM input/output tokens used during the table-selection stage (and for agentic baselines, summed across iterations), and (ii) an estimated dollar cost for SQL generation when using GPT-4o-mini, computed by applying the published pricing ($0.15/1$M input tokens, $0.60/1$M output tokens) to the SQL generator's token usage after each retrieval/selection method.

F.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 exact match (EM) 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}=0.3 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 accordingly.

G Error Analysis Details

We perform an automatic, heuristic error analysis on BIRD (n=1534) for our best configuration (CORE-T with Llama-3.1-8B-Instruct as table selector and GPT-4o-mini as SQL generator) and compare against ARM with GPT-4o-mini. We report two *retrieval/selection* categories and three *SQL-generation* categories. For each category, we report *counts* and the corresponding *rate over all queries* (count$/1534$) in Table 7.

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 also analyze the SQL generator results when the generated SQL references *all and only* the gold tables (so missing-table effects do not apply). For queries that still fail exact match under this setting, we label: (i) **Schema linking** (wrong columns used despite correct tables), (ii) **Formatting** (value-format mismatch, e.g., dates), (iii) **Reasoning/calculation** (incorrect logic, conditions, aggregation, etc.). Labels are applied heuristically and can overlap, e.g., schema-linking and formatting are detected via SQLite error-string patterns while remaining cases are assigned to reasoning/calculation errors.

	Table retrieval/selection		SQL generation		
	Recall issue↓	**Precision issue↓**	**Schema linking↓**	**Formatting↓**	**Reasoning/ calculation↓**
ARM	**139** (9.1%)	1456 (94.9%)	82 (5.4%)	**5** (0.3%)	395 (25.8%)
CORE-T	151 (9.8%)	**1381** (90.0%)	**71** (4.6%)	8 (0.5%)	**368** (24.0%)

Table 7: **Error breakdown on BIRD (n=1534) with Llama-3.1-8B-Instruct as table selector and GPT-4o-mini as SQL generator.** We report counts and rates (over all queries) for table retrieval/selection errors computed from gold vs. selected table sets. SQL-only categories (schema linking / formatting / reasoning) are *only* assigned when the generated SQL uses *all and only* the gold tables. Categories are multi-label (a query may contribute to multiple categories).

	Bird (n=1534, $\bar{G}$=1.95)					**Spider** (n=1034, $\bar{G}$=1.51)					**MMQA** (n=1105, $\bar{G}$=2.20)				
	Avg #tab.↓	**P**	**R**	**F1**	**PR**	Avg #tab.↓	**P**	**R**	**F1**	**PR**	Avg #tab.↓	**P**	**R**	**F1**	**PR**
UAE-Large-V1															
DR@5	5.0	34.9	91.9	49.4	83.0	5.0	29.4	98.0	43.9	95.6	5.0	30.8	70.7	42.6	50.3
DR@8	8.0	23.1	95.9	36.5	91.5	8.0	18.8	99.5	30.9	98.9	8.0	21.2	77.6	33.1	61.3
DR@10	10.0	18.8	97.2	31.0	94.3	10.0	15.0	99.7	25.7	99.1	10.0	17.6	80.5	28.7	66.2
DR@15	15.0	12.7	98.6	22.3	97.0	15.0	10.1	99.9	18.0	99.7	15.0	12.4	84.6	21.5	73.1
Snowflake-arctic-embed-m-v2.0															
DR@5	5.0	33.2	88.1	47.1	76.2	5.0	29.1	97.3	43.5	94.3	5.0	31.3	71.7	43.3	49.0
DR@8	8.0	22.2	93.0	35.2	85.7	8.0	18.6	98.6	30.5	97.1	8.0	21.3	77.9	33.3	59.9
DR@10	10.0	18.2	94.6	30.0	88.8	10.0	14.9	99.0	25.4	97.9	10.0	17.5	79.7	28.5	63.3
DR@15	15.0	12.4	96.6	21.7	92.6	15.0	10.0	99.3	17.9	98.4	15.0	12.2	82.9	21.1	69.4
text-embedding-3-large															
DR@5	5.0	35.9	94.2	50.7	86.9	5.0	29.7	98.9	44.4	97.4	5.0	34.0	77.6	47.0	61.8
DR@8	8.0	23.4	97.3	37.1	93.7	8.0	18.8	99.5	30.9	98.8	8.0	23.0	83.7	35.9	72.4
DR@10	10.0	19.0	98.5	31.4	96.3	10.0	15.1	99.7	25.7	99.3	10.0	18.9	86.1	30.9	76.8
DR@15	15.0	12.8	99.3	22.5	98.2	15.0	10.1	100.0	18.1	99.9	15.0	13.1	89.3	22.8	82.5

Table 8: **Initial table retrieval performance at different top-K cutoffs on Bird, Spider, and MMQA.** Each block corresponds to a different embedding model. The table reports the average number of retrieved tables (lower is better), precision (P), recall (R), F1, and perfect recall (PR), where PR denotes the percentage of questions for which all gold tables are retrieved. DR refers to dense retrieval. $\bar{G}$ denotes 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.

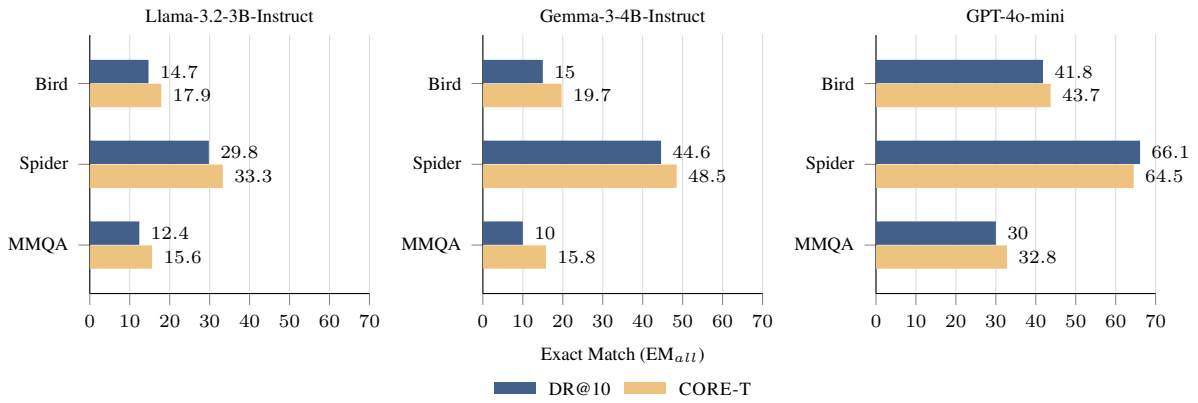

Figure 6: **Ablation: execution match on all queries (EM$_{all}$).** We compare using only the first stage of our pipeline (DR@10) against the full CORE-T pipeline. All settings use UAE-Large-V1 for retrieval and Llama-3.1-8B-Instruct as the table selector, varying only the SQL generator.

	Bird (n=1534)			**Spider** (n=1034)			**MMQA** (n=1105)		
	$EM_{=1T}$ (23.6%)	$EM_{\geq 2T}$ (76.4%)	EM_{all} (100%)	$EM_{=1T}$ (55.6%)	$EM_{\geq 2T}$ (44.4%)	EM_{all} (100%)	$EM_{=1T}$ (0.4%)	$EM_{\geq 2T}$ (99.6%)	EM_{all} (100%)
Llama-3.2-3B-Instruct									
DR@5	16.3	15.4	15.6	30.1	25.1	27.9	0.0	10.9	10.9
DRR@5	14.4	14.5	14.5	33.0	26.1	30.0	0.0	11.4	11.4
ReAct	**25.7**	18.3	**20.0**	**46.4**	30.7	**39.5**	**25.0**	15.2	15.2
JAR@5	14.9	16.0	15.8	32.9	25.9	29.8	—	—	—
ARM	21.0	17.7	18.5	—	—	—	—	—	—
CORE-T	16.9	**18.4**	18.1	36.5	**35.3**	36.0	**25.0**	**17.6**	**17.6**
Oracle	29.6	26.6	27.3	60.9	46.6	54.5	50.0	42.9	42.9
Gemma-3-4B-Instruct									
DR@5	39.8	10.3	17.3	61.0	31.8	48.1	0.0	12.3	12.2
DRR@5	34.0	9.6	15.4	59.3	32.0	47.2	25.0	13.0	13.0
ReAct	**40.1**	11.5	18.3	**63.5**	33.6	50.2	**50.0**	13.4	13.5
JAR@5	35.9	10.8	16.8	59.5	32.7	47.6	—	—	—
ARM	35.6	**15.5**	**20.3**	—	—	—	—	—	—
CORE-T	39.8	13.7	19.9	59.3	**42.7**	**51.9**	**50.0**	**17.6**	**17.7**
Oracle	44.2	17.0	23.4	77.0	41.4	61.2	50.0	38.1	38.1
GPT-4o-mini									
DR@5	61.6	33.1	39.8	72.2	55.1	64.6	**50.0**	26.8	26.9
DRR@5	60.5	33.6	40.0	72.9	**57.3**	**66.0**	**50.0**	30.6	30.7
ReAct	**61.9**	35.5	41.7	**75.3**	51.6	64.8	**50.0**	31.0	31.0
JAR@5	60.8	35.2	41.3	71.5	53.2	63.3	—	—	—
ARM	59.9	37.2	42.6	—	—	—	—	—	—
CORE-T	61.0	**38.1**	**43.5**	72.3	55.6	64.9	**50.0**	**33.7**	**33.8**
Oracle	64.9	46.5	50.8	84.3	63.0	74.9	50.0	63.4	63.3

Table 9: **End-to-end EM with Qwen-2.5-7B-Instruct as the table selector.** Same metrics as Table 3; Oracle uses gold tables. **Best**/second best are among non-oracle methods within each SQL-generator block.

H 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

```
Your task: From a set of retrieved tables, identify the 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 keyforeign 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}
```