REAR: Retrieve, Expand and Refine for Effective Multitable Retrieval

Anonymous ACL submission

Abstract

Answering natural language queries over relational data often requires retrieving and reasoning over multiple tables, yet most retrievers optimize only for query–table relevance and ignore table–table compatibility. We introduce REAR (Retrieve, Expand and Refine), a three-stage, LLM-free framework that separates semantic relevance from structural joinability for efficient, high-fidelity multi-table retrieval. REAR (i) retrieves query-aligned tables, (ii) expands these with structurally joinable tables via fast, precomputed column-embedding comparisons, and (iii) refines them by pruning noisy or weakly related candidates. Empirically, REAR is retriever-agnostic and consistently improves dense/ sparse retrievers on complex table QA datasets (BIRD, MMQA, and Spider) by improving both multi-table retrieval quality and downstream SQL execution. Despite being LLM-free, it delivers performance competitive with state-of-the-art LLM-augmented retrieval systems (e.g., ARM) while achieving much lower latency and cost. Ablations confirm complementary gains from expansion and refinement, underscoring REAR as a practical, scalable building block for table-based downstream tasks (e.g., Text-to-SQL).

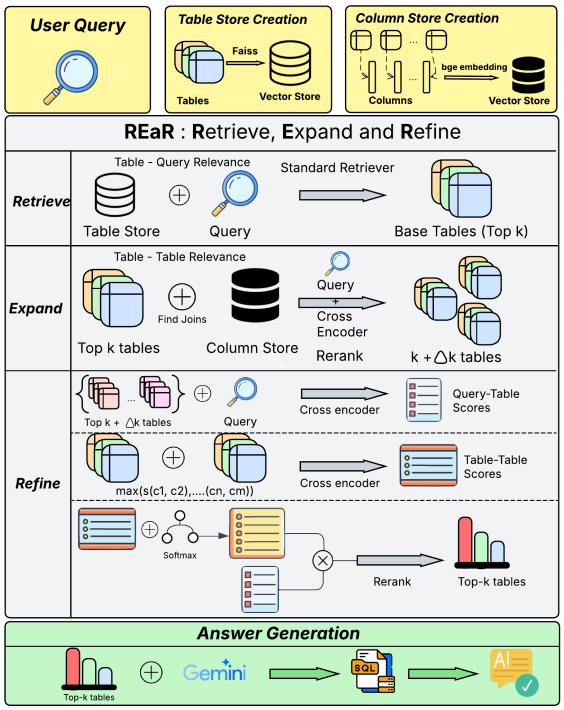

Figure 1: Overview of the REAR framework. (1) **Retrieve**: Select top-k relevant tables. (2) **Expand:** Augment with joinable tables using FAISS column search and cross-encoder reranking ($k \rightarrow k' + \Delta k'$ candidates). (3) **Refine:** Score candidates by query relevance and table joinability, rerank to final top-k. Top boxes: offline preprocessing

1 Introduction

Retrieval in current Text-to-SQL pipelines is fundamentally myopic: it optimizes for query–table relevance while ignoring table–table compatibility. This mismatch creates a brittle handoff, retrievers surface tables that look topically aligned, but lack the join paths, key constraints, or schema alignment needed for integration. Downstream, the semantic parser inherits an incoherent candidate set, forcing it to guess joins, drop constraints, or hallucinate links, errors that propagate into invalid or incomplete SQL. In short, relevance-only retrieval starves the parser of relational context; without explicit reasoning over joinability and connectivity, even a "relevant" set of tables cannot be composed into a correct query. The remedy is retrieval that jointly scores (i) topical fit and (ii) relational fit, so Text-to-SQL starts from a schema-aware, join-ready subset rather than a bag of loosely related tables.

Prior LLM-based efforts span schema linking (E-SQL (Caferoğlu and Ulusoy, 2024), RSL-SQL (Cao et al., 2024)), column-level joinability (DeepJoin (Dong et al., 2023)), and multi-hop retrieval (MURRE (Zhang et al., 2025a)), yet they often falter when relationships are implicit or distant. Despite these advances, gaps remain: many techniques require heavy, dataset-specific training that weakens generalization to unseen databases;

multi-hop strategies can be computationally expensive; and overall, they still trade off recall (capturing all necessary tables) against precision (excluding distractors), limiting real-world utility. Meanwhile, state-of-the-art reasoning-augmented retrieval methods (e.g., ARM (Chen et al., 2025b)) rely on costly LLM calls, trading efficiency for stronger performance. This raises a critical question: *Can we design a **simple and efficient retriever** that scores both **query–table relevance and table–table** compatibility?*

To address this question, we introduce REAR (**R**etrieve, **E**xpand **a**nd **R**efine), a three-stage framework that cleanly separates query–table from table–table reasoning to deliver efficient, high-fidelity multi-table retrieval without any LLM calls. As illustrated in Figure 1, REAR (i) retrieves tables that are semantically aligned with the query, (ii) expands this set by adding tables that are structurally joinable with the retrieved ones, and (iii) refines the pool by pruning noisy or redundant items. This decomposition jointly captures what information is needed (query–table relevance) and how it connects (table–table compatibility), addressing core limitations of prior methods.

Concretely, the *Expansion* stage boosts recall by scanning the database for tables that are semantically joinable with this set via precomputed column-embedding comparisons that are far more efficient than LLM-based checks to uncover latent relationships required for complex joins. Furthermore, the *Refinement* stage removes irrelevant or weakly related tables, producing a final set that is both precise and complete. Overall, REAR forms a lightweight pipeline that unites semantic and structural reasoning for efficient, LLM-free multi-table retrieval. Across multiple complex table QA datasets, REAR reliably improves standard retrievers, reaching performance close to state-of-the-art LLM-based retrieval systems (e.g., JAR (Chen et al., 2024) and ARM) while remaining compute- and cost-efficient. By better selecting and refining tables, REAR also boosts downstream SQL execution performance.

Our contributions are as follows:

- We introduce REAR, which separates query–table relevance from table–table joinability, using precomputed column embeddings to identify join-ready tables before effective precision filtering.
- REAR eliminates LLM calls from retrieval, enhancing retrieval performance with substantially lower latency and cost, making it practical for large-scale real-world usage.
- REAR acts as a plug-and-play layer over standard retrievers, boosting recall (through expansion) and precision (through refinement) without modifying the base retriever.
- Extensive experiments on MMQA, BIRD, and Spider show that REAR also improves downstream SQL execution accuracy, with ablations validating the impact of each stage.

2 Methodology

Inspired by (Chen et al., 2024, 2025b; Dong et al., 2023), an effective multi-table retriever should jointly optimize query–table relevance and table–table joinability. The former ensures that retrieved tables contain information germane to the user query, whereas the latter guarantees that the tables can be connected so that their evidence can be composed to answer the query. As discussed in Section 1, existing approaches frequently employ LLMs during retrieval, which is often impractical due to latency and cost. To mitigate these limitations, we propose a top-k multi-table retrieval pipeline that avoids online LLM usage by explicitly models both objectives using offline embeddings of tables, queries, and columns.

Given a query q, REAR comprises three stages: retrieval, expansion, and refinement. (1) **Retrieval:** a standard dense, sparse, or hybrid retriever selects k' base tables that are semantically relevant to q, establishing query–table relevance. (2) **Expansion:** $\Delta k'$ additional tables that are joinable with the base set are introduced, promoting table–table joinability. (3) **Refinement:** the $(k' + \Delta k')$ candidates are reduced to k by jointly scoring relevance and joinability, and the resulting set is returned as the final top-k tables. Detailed specifications of these stages are provided in Section 2.1, Section 2.2, and Section 2.3.

2.1 Retrieval

For a user query q, the retrieval stage first identifies a set of *base tables* $\mathcal{T}_{\text{base}} \subset \mathcal{T}$ that are semantically relevant to q to ensure query-table relevance, where $\mathcal{T}$ denotes the full table corpus. These base tables provide the foundation for subsequent stages, which further account for table-table joinability.

Specifically, we compute a relevance score $s(q, t_i)$ for each table $t_i \in \mathcal{T}$, rank all tables by

their scores in descending order, and select top-k' tables: $\mathcal{T}_{\text{base}} = \{t_{(1)}, t_{(2)}, \ldots, t_{(k')}\}$, where $t_{(i)}$ denotes the i-th highest-scoring table. The relevance scoring function $s(q, t_i)$ varies by retriever type:

1. **Sparse retrievers** compute relevance using TF-IDF score:
$$s_{\text{sparse}}(q, t_i) = \sum_{w \in q \cap t_i} \text{TF}(w, t_i) \cdot \text{IDF}(w)$$
where w denotes a word that appears in both the query q and the table t_i.
2. **Dense retrievers** compute relevance as cosine similarity:
$$s_{\text{dense}}(q, t_i) = \frac{\mathbf{e}_q \cdot \mathbf{e}_{t_i}}{\|\mathbf{e}_q\| \|\mathbf{e}_{t_i}\|}$$
where $\mathbf{e}_q$ and $\mathbf{e}_{t_i}$ are the learned embeddings of the query and table, respectively.
3. **Hybrid retrievers** combine both approaches i.e. $s_{\text{hybrid}}(q, t_i) =$:
$$\alpha \cdot s_{\text{sparse}}(q, t_i) + (1 - \alpha) \cdot s_{\text{dense}}(q, t_i)$$
where $\alpha \in [0, 1]$ is a weighting hyperparameter.

Further details on query-table relevance computation are provided in Appendix A.1.

2.2 Expansion

Although the base tables $\mathcal{T}_{\text{base}}$ described in Section 2.1 are semantically relevant to the user query q, they may not be mutually joinable, which limits their ability to compose information to address q. To address this, we introduce an expansion stage that explicitly adds tables joinable with the base set, ensuring table-table joinability and thereby increasing the likelihood of answering q. Concretely, the expansion stage consists of two steps: detection and reranking. We first identify tables from the corpus $\mathcal{T}$ that are joinable with the base tables, and then rerank these candidates by their query-table relevance, yielding tables that are both joinable and semantically relevant to the query.

Detection. Inspired by DeepJoin (Dong et al., 2023), we identify joinable tables by measuring the similarity between column embeddings from pairs of tables. Formally, let $\mathcal{C}(t)$ denote the set of columns in table t. For two tables $t_i, t_j \in \mathcal{T}$, we define them as joinable if:
$$\text{joinable}(t_i, t_j) = \max_{c_i \in \mathcal{C}(t_i), c_j \in \mathcal{C}(t_j)} \text{sim}(c_i, c_j) \geq \tau$$
where $\text{sim}(c_i, c_j)$ is the cosine similarity between column embeddings and τ is a threshold (set to 0.7 in our experiments). Unlike DeepJoin, which relies on dataset-specific embedding models, we use general-purpose pre-trained embeddings to ensure cross-domain generalizability.

To construct a column embedding $\mathbf{e}_c$ for column c, we first serialize it by concatenating its name and values into a text representation $\text{serialize}(c)$, which is then encoded using a dense embedding model f_{enc}:
$$\mathbf{e}_c = f_{\text{enc}}(\text{serialize}(c))$$

Since exhaustively comparing all column pairs requires $\mathcal{O}(|\mathcal{T}|^2 \cdot |\mathcal{C}_{\text{avg}}|^2)$ operations (where $|\mathcal{C}_{\text{avg}}|$ is the average number of columns per table), we leverage approximate nearest neighbor search algorithms (Douze et al., 2024), enabling sublinear-time retrieval of candidate column matches.

Finally, for each base table $t_b \in \mathcal{T}_{\text{base}}$ obtained in Section 2.1, we enumerate its joinable tables to form the candidate joinable set:
$$\mathcal{T}_c = \bigcup_{t_b \in \mathcal{T}_{\text{base}}} \{t \in \mathcal{T} \mid \text{joinable}(t_b, t)\}$$

Reranking. To filter tables that are semantically distant from the user query, we rerank the candidate joinable tables $\mathcal{T}_c$ using a cross-encoder reranker $r(q, t)$ that computes joint query-table representations. We rank all tables in $\mathcal{T}_c$ by their reranker scores and select the top-$(\Delta k')$:
$$\mathcal{T}_{\text{join}} = \{t_{(1)}, t_{(2)}, \ldots, t_{(\Delta k')}\}$$
where $t_{(i)}$ denotes the i-th highest-scoring table under $r(q, \cdot)$. These filtered joinable tables are combined with the base tables to form the expanded set:
$$\mathcal{T}_{\text{expanded}} = \mathcal{T}_{\text{base}} \cup \mathcal{T}_{\text{join}}$$

2.3 Refinement

Finally, we refine the expanded set $\mathcal{T}_{\text{expanded}}$ (described in Section 2.2) by selecting the final subset of tables to return, *jointly* considering query–table relevance and table–table joinability.

For each table T_i in the expanded set, we compute a score $S(T_i)$, which is then used to rank all tables. The top-k tables with highest scores $S(T_i)$ are returned, where $S(T_i)$ combines query-table relevance and table-table joinability:
$$S(T_i) = C(q, T_i) \cdot A(T_i)$$
where $C(q, T_i)$ and $A(T_i)$ denote the query-table relevance and table-table joinability, respectively, and q is the user query.

Query-table relevance. We first compute the similarity between the query q and T_i using a cross-encoder model (detailed in Section 3.1), yielding the score $C(q, T_i)$. We employ a cross-encoder rather than embedding-based similarity for two key

reasons. First, cross-encoders enable fine-grained token-level interactions between the query and table schema through full self-attention, allowing the model to identify subtle semantic relationships that bi-encoders, which encode inputs independently, cannot capture. Second, at this stage, we operate on a small candidate set, making the computational overhead of cross-encoders tractable while prioritizing scoring accuracy over inference speed. This mirrors standard practice: bi-encoders for retrieval, cross-encoders for reranking.

Table-table joinability. We then assess the relational coherence of T_i with respect to other tables in the retrieved set.

$$A(T_i) = \max(\text{softmax}(C(T_i, T_j)) \cdot C(T_i, T_j))$$

For each table pair (T_i, T_j) where T_j is in the neighborhood of T_i, we compute column-level similarities using the same cross-encoder. Specifically, for all column pairs between T_i and T_j, we retain the maximum similarity score as the table-pair score $C(T_i, T_j)$. This design leverages the observation that joinable columns sharing foreign key relationships exhibit high semantic similarity.

We compute the attention score $A(T_i)$ by applying softmax normalization over all table-table similarity scores $C(T_i, T_j)$, selecting the maximum normalized score, and scaling it by the corresponding similarity value. This max operation suppresses contributions from weakly-related tables.

3 Experiments

Our goal is to evaluate the effectiveness of the proposed multi-table retrieval pipeline, REAR, which introduces a novel retrieve–expand–refine process. Compared to *standard retrievers* that lack the expansion and refinement stages, we aim to understand the added benefits of these components in our pipeline. Unlike *LLM-based retrieval* methods, our approach avoids online LLM calls entirely.

For evaluation, we use complex open-domain Text-to-SQL datasets as they require reasoning over multiple tables to produce correct answers. These datasets allow us to assess performance along two axes: (1) whether gold tables are retrieved, capturing retrieval performance, and (2) whether the correct final answer is generated, capturing end-to-end performance.

3.1 Experimental Setup

Datasets. We evaluate all methods on complex text-to-SQL datasets that require reasoning over multiple tables: MMQA (Wu et al., 2025), Spider (Yu et al., 2018), and Bird (Li et al., 2023a). Since most databases in Spider and Bird contain at most 10 tables, retrieval in these cases is trivial: for example, retrieving all 10 tables from a 10-table database guarantees 100% recall. To create a more challenging evaluation setting, we merge all databases into a single corpus (statistics in Table 1).

Table 1: Dataset statistics and characteristics

	BIRD	SPIDER	MMQA
Avg. Rows per Table	52436.46	6742.46	1732.24
Avg. Columns per Table	10.64	5.51	5.77
Avg. Tables per Query	1.96	1.91	2.22
Total Databases	11	20	–
Total Tables	75	139	695
Total Queries	1534	1034	3312

Baselines. We compare our method with multiple standard and LLM-based retrieval methods.

- For **standard retrievers**, we consider three methods: sparse retrievers, dense retrievers, and hybrid methods combining both. For sparse retrievers, we employ the BM25 implementation from (Robertson and Zaragoza, 2009) and Splade (Lassance and Clinchant, 2022). For dense retrievers, we select lightweight models that rank highly on the MTEB leaderboard, including UAE-Large-V1 (Li and Li, 2024), GTE-large (Li et al., 2023b), e5-mistral (Wang et al., 2023), and BGE-large-en (Xiao et al., 2023). For hybrid retrievers, we used Splade retriever's hybrid version for a vast coverage of different types of retrievers.

- For **LLM-based retrieval** methods, we evaluate two approaches: JAR (Chen et al., 2024) and ARM (Chen et al., 2025b). These methods dynamically determine the number of tables for SQL generation, rather than fixing the retrieval count a priori.

Evaluation Metrics. We assess the performance of different methods along two dimensions: retrieval performance and end-to-end performance.

- For **retrieval performance**, we report standard precision and recall, along with a stricter metric we call full recall, by comparing the retrieved tables and the gold tables. Full Recall is a binary measure indicating whether all gold tables appear in the retrieved set, a necessary condition for generating the correct final answer.

- For **end-to-end performance**, we use execution accuracy, which evaluates query correctness by comparing the execution results of SQL statements generated using the retrieved tables against those

Table 2: Retrieval performance of standard retrievers and our method. The expansion stage of our method expands the set of candidate tables to 8, and the refinement stage reduces the set of tables to 5. **Bold** and underline numbers denote best and second-best, respectively.

	BIRD			MMQA			Spider		
	Recall	Precision	Full Recall	Recall	Precision	Full Recall	Recall	Precision	Full Recall
	Dense Retrievers								
BGE (Top 8)	96.11	23.88	91.53	79.29	21.94	60.18	97.92	18.54	96.62
Retrieval (Top 5)	90.70	35.64	81.75	72.03	31.78	48.53	96.69	29.20	94.19
+Expansion	**96.51**	24.66	**93.16**	**87.40**	24.31	**74.71**	**98.84**	18.76	**98.55**
+Refining (REAR)	93.32	**35.90**	86.42	82.78	**36.70**	65.83	96.69	**29.26**	95.94
UAE (Top 8)	94.72	23.92	89.37	81.28	22.49	64.29	99.10	18.77	98.45
Retrieval (Top 5)	90.68	36.09	82.79	73.92	32.61	52.27	98.06	**29.75**	96.61
+Expansion	**96.81**	25.26	**93.42**	**87.91**	24.44	**75.64**	**99.32**	18.88	**99.13**
+Refining (REAR)	93.96	**36.33**	87.29	83.19	**36.86**	66.61	98.50	29.61	97.68
GTE (Top 8)	92.54	22.82	85.40	73.25	20.17	53.34	98.03	18.59	97.00
Retrieval (Top 5)	87.46	34.05	76.60	64.95	28.52	41.93	94.25	28.22	89.85
+Expansion	**96.17**	24.24	**92.13**	**85.72**	23.80	**72.53**	**98.79**	18.75	**98.45**
+Refining (REAR)	92.85	**35.70**	84.95	81.65	**36.17**	64.32	97.10	**29.28**	96.03
e5 (Top 8)	95.17	22.90	89.57	77.75	21.68	58.59	97.59	18.38	96.22
Retrieval (Top 5)	90.49	34.41	81.49	69.96	31.09	45.94	94.96	28.45	91.50
+Expansion	**95.38**	23.25	**90.66**	**87.40**	24.31	**74.74**	**98.59**	18.68	**98.16**
+Refining (REAR)	92.18	**35.40**	83.82	83.57	**37.08**	67.37	96.94	**29.23**	95.84
	Sparse Retrievers								
BM25 (Top 8)	88.28	21.41	77.25	79.32	22.09	58.90	95.08	17.84	91.10
Retrieval (Top 5)	81.10	31.21	65.65	73.16	32.48	48.42	91.02	27.16	84.82
+Expansion	**93.71**	23.08	**88.40**	**87.62**	24.40	**75.03**	**98.55**	18.70	**98.07**
+Refining (REAR)	91.04	**34.86**	82.72	83.69	**37.16**	67.43	96.76	**29.21**	95.74
SPLADE (Top 8)	96.02	23.83	91.4	82.18	22.82	64.26	91.04	17.14	86.84
Retrieval (Top 5)	90.97	35.69	81.75	75.91	33.57	53.9	81.14	24.31	71.76
+Expansion	**97.67**	24.07	**94.58**	**88.99**	24.76	**77.66**	**99.17**	18.65	**98.83**
+Refining (REAR)	94.15	**36.08**	87.16	83.86	**37.16**	67.79	97.22	**29.34**	96.32
	Hybrid Retrievers								
$\text{SPLADE}_{\text{Hybrid}}$ (Top 8)	96.08	23.83	91.05	84.82	23.5	68.97	95.48	24.97	93.35
Retrieval (Top 5)	92.49	35.96	84.06	78.4	34.59	57.13	85.46	**29.46**	75.04
+Expansion	**97.27**	23.88	**93.81**	**89.19**	24.79	**77.69**	**96.74**	18.24	**95.55**
+Refining (REAR)	93.88	**36.43**	86.64	83.71	**37.11**	67.2	96.47	29.07	94.97

of the gold SQL statements. The accuracy is 1 if the results are the same and 0 otherwise.

Implementation details. We generate table descriptions to augment the original table schemas for all methods, inspired by (Chen et al., 2025a).

For column encoding in the expansion stage, we evaluated several state-of-the-art embedding models from the MTEB leaderboard, including bge-large-en, Qwen-Embedding (Zhang et al., 2025b), and Gemini-Embeddings (Lee et al., 2025). Based on our experiments, we adopted **bge-large-en**, which achieves performance comparable to larger models while maintaining computational efficiency. (More details in appendix A)

We fixed the number of joinable tables to 3 for all main experiments. We do this since most queries in our target datasets require 3 or less than 3 tables to answer the questions and to maintain the computational efficiency of our pipeline. We employ Jina-reranker as the reranker, a state-of-the-art and computationally efficient reranking model. In the refinement stage of our retrieval pipeline, we used Jina-reranker as the cross-encoder. For SQL generation, we employ both small and large language models: Gemini-2.0-Flash, Llama-3.2-3B, and Gemma-3-4B.

3.2 Results and analysis

How do the *Expansion* and *Refinement* stages contribute to improvements in retrieval performance over standard top-*k* baselines? To isolate the contributions of our pipeline's core components, we first analyze their impact on retrieval metrics. As shown in Table 2, both stages provide substantial benefits over a standard *"Retrieval (Top 8)"* baseline. Focusing on the BIRD dataset with the UAE dense retriever, the **Expansion** stage significantly enhances the chance of retrieving the complete set of gold-standard tables, boosting the "Full Recall" metric from 82.79% to 93.42%, an absolute improvement of 10.63 points. While this stage axiomatically lowers precision by increasing the candidate set size, the subsequent **Refinement** stage effectively re-optimizes the precision-recall balance. The complete *"+Refining"* pipeline achieves a Full Recall of 87.29%, which remains nearly 5 points higher than the baseline, while con-

Table 3: SQL execution accuracy at each stage of the REAR pipeline (*"Retrieval"*, *"+Expansion"*, *"+Refining"*) compared to baseline retrieval methods. UAE-V1-Large is used as the dense retriever for optimal performance, with Splade-Sparse and Splade-Hybrid as the sparse and hybrid retrievers, respectively. **Bold** and underline numbers denote best and second-best, respectively, excluding Oracle.

	BIRD			SPIDER			MMQA		
	Dense	**Sparse**	**Hybrid**	**Dense**	**Sparse**	**Hybrid**	**Dense**	**Sparse**	**Hybrid**
				Gemini 2.0 Flash					
Oracle Retrieval		50.39			76.98			53.9	
Retrieval (Top 8)	38.33	41	41.91	69.54	64.02	70.79	30.67	33.19	35.16
Oracle Prune	46.21	45.83	46.98	76.20	68.76	74.75	43.83	37.94	42.37
Retrieval (Top 5)	37.87	37.03	39.84	69.44	57.54	46.9	26.73	30.82	32.96
+Expansion	42.24	41.13	**42.91**	69.34	70.79	70.9	37.45	37.84	37.88
+Refining (REAR)	**43.09**	**41.54**	40.76	**69.93**	**70.8**	**71.8**	**38.87**	**37.87**	**38.15**
Oracle Prune	47.96	49.15	48.72	76.50	76.46	74.95	43.92	43.74	47.08
				Llama-3.2-3b					
Oracle Retrieval		15.71			40.03			30.12	
Retrieval (Top 8)	9.12	9.58	10.12	17.41	16.24	17.98	12.85	12.24	13.94
Oracle Prune	14.46	14.64	15.07	39.55	35.1	39.36	22.3	22.42	24.02
Retrieval (Top 5)	9.32	11.15	10.58	18.96	17.69	11.41	13.28	14.15	13.85
+Expansion	9.13	9.45	10.05	18.57	19.24	18.08	13.09	13.4	13.91
+Refining (REAR)	**11.55**	**11.91**	**11.38**	**19.34**	**19.53**	**19.05**	**14.39**	**14.51**	**14.12**
Oracle Prune	14.69	15.23	15.17	24.31	39.36	39.16	25.62	25.56	26.01
				Gemma-3-4b					
Oracle Retrieval		26.27			47.38			21.24	
Retrieval (Top 8)	13.89	12.91	14.31	17.4	17.31	15.76	8.66	8.54	8.84
Oracle Prune	24.31	24.69	25.49	47	42.06	47.29	16.4	16.39	17.11
Retrieval (Top 5)	17.6	16.62	17.24	21.76	19.92	20.11	9.8	8.99	10.26
+Expansion	15.67	13.82	17.18	16.83	18.66	17.5	8.6	8.93	9.08
+Refining (REAR)	**17.97**	**17.21**	**17.64**	**22.02**	**24.68**	**21.4**	**11.4**	**10.14**	**10.59**
Oracle Prune	24.67	25.62	25.9	47.3	46.7	47.19	18.5	18.89	18.38

currently maintaining precision, i.e., from 36.09% to 36.33%. This demonstrates that the refinement mechanism removes noisy expansion candidates while keeping the tables most vital for semantic completeness.

Within the Expansion stage, is a join-aware expansion strategy more effective than simply increasing the number of initial retrieved tables? A central hypothesis of our work is that a join-aware structural expansion is superior to a naive k-augmentation strategy. The data in Table 2 validates this assertion. Employing the BGE retriever on the BIRD dataset, a standard *"Retrieval (Top 8)"* approach yields a Full Recall of 91.53%. In contrast, our method, which initiates with 5 tables and intelligently expands the set to 8 (*"+Expansion"*), achieves a superior Full Recall of 93.16%. This pattern across retrievers and datasets shows that modeling inter-table joinability is more effective for recall than increasing the retrieval window.

To what extent do gains in retrieval performance from the REAR pipeline lead to improvements in downstream end-to-end SQL execution accuracy? Ultimately, an advanced retriever is only as valuable as its ability to produce more accurate SQL queries. Table 3 reveals a direct and positive propagation of retrieval gains to end-to-end performance. Using the UAE dense retriever and Gemini 2.0 Flash on the BIRD dataset, the baseline *"Retrieval (Top 5)"* yields an execution accuracy of 37.87%. The introduction of the **Expansion** stage elevates this accuracy to 42.24% (+4.37 points), indicating that a more complete schema context directly mitigates downstream generation errors. The full REAR pipeline with **Refinement** further improves accuracy to 43.09%, demonstrating that by increasing the signal-to-noise ratio in the retrieved table set, our pruning mechanism provides a cleaner, higher-fidelity context to the generator, culminating in the best end-to-end performance.

Does the improved retrieval performance of the REAR pipeline hold consistently across different SQL generation models? To validate the model-agnostic nature of our retrieval framework, we evaluated its impact across three distinct LLMs. The results in Table 3 demonstrate that the benefits of REAR are robust and generalize consistently. For the dense retriever on BIRD, the full *"+Refining"* pipeline improved execution accuracy over the *"Retrieval (Top 5)"* baseline for all models: from 37.87% to 43.09% for Gemini 2.0 Flash, from 9.32% to 11.55% for Llama-3.2-3B, and from 17.6% to 17.97% for Gemma-3-4B. This consistent performance uplift, irrespective of the genera-

Table 4: REaR against LLM-based methods on retrieval quality and SQL execution accuracy on different datasets across multiple LLMs. "-Desc." represents performance without offline computed LLMs table descriptions. MURRE SQL accuracy is on gpt-3.5-turbo. **Bold** and underline numbers denote best and second-best, respectively; "-" denotes unavailable results.

	BIRD						Spider					
	Retrieval			SQL Execution Accuracy			Retrieval			SQL Execution Accuracy		
	Precision	Recall	$\text{Recall}_{\text{Full}}$	LLAMA	GPT	Gemini	Precision	Recall	$\text{Recall}_{\text{Full}}$	LLAMA	GPT	Gemini
ARM	**42.7**	96.5	92.7	20.6	**32.4**	30.4	-	-	-	-	-	-
JAR	40.3	89.9	77.9	20.07	25.2	29.1	**41.9**	97.8	96.23	56.8	70.0	**75.7**
ReAcT	15.0	**96.7**	**93.5**	4.7	25.4	-	-	-	-	-	-	-
MURRE	-	87.6	80.1	-	21.8	-	-	94.3	93.5	-	64.4	-
REAR	36.1	93.9	87.3	**22.1**	27.9	**34.0**	29.6	**98.5**	**97.7**	**58.3**	**73.2**	74.3
–Desc.	34.7	91.1	81.1	21.57	26.92	32.37	29.1	96.8	95.6	51.1	55.5	66.7

tor's scale or architecture, confirms that providing a high-quality, well-pruned set of tables is a fundamental improvement at the retrieval level that robustly enhances end-to-end SQL performance.

How much gap does our approach close w.r.t. to human retrieval and human pruning? We compare REAR against two oracle baselines: Oracle Retrieval, where a human perfectly selects the relevant set of tables, and Oracle Prune, where a human perfectly prunes a candidate set, either from standard retrieval or after our method expansion. Across all three models (Gemini 2.0 Flash, LLaMA 3.2-3B, and Gemma 3 4B (Team et al., 2025)) and datasets (BIRD, SPIDER, and MMQA), Oracle Retrieval consistently yields significantly higher accuracy than standard retrievers, highlighting the large gap between current retrieval quality and the upper bound. This oracle-prune benefit is especially pronounced for weaker models: LLaMA-3.2-3B improves from 9.32% to 14.69% (+5.14 points) on BIRD, confirming that retrieval noise is a critical bottleneck for smaller models that lack the implicit robustness of larger LLMs.

Our REAR pipeline closes much of this gap by combining recall-oriented expansion with refinement, which effectively prunes irrelevant candidates and brings performance close to the Oracle Prune upper bound. On MMQA with Gemini, Hybrid accuracy rises from 32.96% (Top-5) to 38.15% after refinement, compared to 47.08% for Oracle Prune after expansion recovering over 5.19% of the potential gain. On SPIDER, refinement achieves 70.9%, just 4 points below the 74.95% Oracle score. Even in smaller models, refinement consistently improves over naive expansion and captures much of the oracle-level benefit. These results demonstrate that our method maintains precision while improving recall for multi-table reasoning.

How does the REAR pipeline compare to state-of-the-art LLM-based retrieval methods in terms of both retrieval quality and end-to-end execution accuracy? A primary objective of this work is to achieve comparable performance with computationally expensive LLM-based retrieval systems. The benchmarks in Table 4 confirm REAR's success in this regard. On the BIRD dataset, our method records a SQL execution accuracy of 34.00% with the Gemini model, outperforming the ARM baseline of 30.42%. On Llama, REAR maintains competitive performance at 22.1%, outperforming ARM (20.6%) while ReAct degrades severely to just 4.7%. On retrieval quality, our method achieves superior recall of 93.9% and full recall of 87.3%, compared to MURRE's 87.6% and 80.1%, respectively. Similarly, on the Spider dataset, our Gemini-based result of 74.3% is highly competitive with JAR's reported accuracies. The retrieval quality metrics further validate our approach: REAR achieves consistently high full recall@k of 87.3% (BIRD) and 97.7% (Spider), where missing key tables leads to cascading SQL generation failures. These outcomes are significant, proving that an efficient, non-LLM retrieval architecture can match or exceed the performance of methods that rely on costly iterative LLM reasoning for retrieval.

What computational efficiency advantages does the REAR pipeline offer over LLM-based retrieval approaches? Beyond raw performance, computational overhead is a critical vector for evaluating real-world viability. As quantified in Table 5, the efficiency gains of our LLM-free retrieval architecture are an order of magnitude. On the BIRD dataset, the REAR pipeline requires an average of only 1,533.89 tokens per query for the final SQL generation step. In stark contrast, the LLM-guided ARM retriever consumes 19,747.58 tokens. This represents a **92.24% reduction in token consumption**, underscoring the profound cost and latency advantages inherent to our design. This level of

efficiency makes REAR a far more scalable and economically viable solution for latency-sensitive, production-scale environments.

Table 5: Cost comparison with LLM-based methods based on number of tokens

Tokens	#Input	# Output	# Total
ARM	19307.34	440.24	19747.58
MURRE	16055	1600	17655
JAR	2140.04	81.74	2221.78
REAR	1492.47	41.41	**1533.89**

4 Ablation Studies

We conduct ablation studies to isolate the contribution of each component in REAR: base retrieval, expansion through joinability discovery, and refinement via attention-inspired pruning. Figure 2 presents the cumulative performance gains, averaged across three datasets with UAE retrieval at each stage.

Expansion. We isolate the effect of expansion by comparing base retrieval (BR) against retrieval with expansion only (REAR w/o Refinement). Here we naively prune the tables after expansion using query-table similarity computed with the cross encoder. Expansion yields **+4.76 points in PR** (77.72% → 82.48%) and **+3.21 points in R** (87.55% → 90.76%). This confirms joinability-based expansion recovers relevant tables missed by semantic retrieval alone.

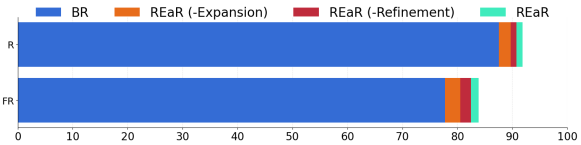

Figure 2: The average recall and Full Recall of base retrieval (UAE) and our method with modules removed: REAR(-Expansion), REAR(-Refinement), and REAR for the top-5 retrieved objects across Bird, MMQA, and Spider

Refinement. We isolate the effect of refinement by comparing base retrieval (BR) against retrieval with refinement only (REAR w/o Expansion). Refinement yields **+2.82 points in PR** (77.72% → 80.54%) and **+2.14 points in R** (87.55% → 89.69%). Without expansion, refinement still improves performance by applying cross-encoder scoring to better rank the initial candidate set, filtering out weakly related tables while retaining those most likely to contribute to correct SQL generation.

Combined Effect. The full pipeline (REAR) achieves **83.86% PR and 91.83% R**, representing total gains of **+6.14 points in PR and +4.28 points in R** over the baseline. Comparing REAR against each ablated variant reveals the synergistic effect: adding refinement to expansion contributes an additional **+1.38 points in PR** (82.48% → 83.86%) and **+1.07 points in R** (90.76% → 91.83%), while adding expansion to refinement contributes **+3.32 points in PR** (80.54% → 83.86%) and **+2.14 points in R** (89.69% → 91.83%).

This dual benefit validates our design: expansion maximizes coverage by discovering joinable tables, while refinement maintains precision by pruning noise through multi-faceted relevance scoring. This structured approach is particularly valuable for smaller language models, which lack the implicit filtering capabilities of larger models and thus benefit more from explicit noise reduction.

5 Conclusion

We introduce REAR—a three-stage (Retrieve, Expand, Refine) framework that jointly optimizes query–table relevance and table–table joinability. REAR retrieves semantically relevant tables, expands with structurally joinable candidates via precomputed column embeddings, and refines with precision-focused pruning, enabling efficient, high-fidelity multi-table retrieval without online LLM calls. Across BIRD, MMQA, and Spider, REAR consistently improves retrieval and the resulting SQL execution accuracy. Despite being LLM-free, it performs on par with state-of-the-art LLM-based systems (ARM, JAR) while using fewer tokens, making it practical at scale. These results highlights that explicitly modeling table–table compatibility is crucial for effective multi-table retrieval.

We will extend REAR in four directions: (1) richer join patterns, supporting n-ary and conditional joins with join-path discovery under structural constraints and cost-based optimization; (2) learned joinability signals that blend schema metadata, FK structure, and column-embedding similarity to predict valid joins while suppressing spurious ones; (3) adaptive expansion that balances recall and latency by adjusting depth to query complexity via query-aware heuristics or reinforcement learning; and (4) execution-in-the-loop refinement, using lightweight SQL feedback and counterfactual probes to iteratively prune candidates and improve end-to-end SQL accuracy.

Limitation

Our work focuses on static relational databases with explicit schema structures, limiting exploration of other data paradigms such as NoSQL databases, knowledge graphs, and dynamically evolving schemas that require real-time index updates. While REAR demonstrates effectiveness on merged database corpora, true cross-database retrieval scenarios involving distributed systems with heterogeneous schema conventions remain unexplored. Additionally, our evaluation is confined to single-turn text-to-SQL queries; multi-turn conversational scenarios where context accumulates across interactions and previous retrieval decisions influence subsequent queries present unique challenges our current framework does not address. Extension to multilingual scenarios, databases with implicit or complex join patterns (multi-hop joins, self-joins, non-equi joins), and semi-structured or hierarchical data representations requires further investigation. These limitations suggest important directions for future work in building more robust and generalizable multi-table retrieval systems.

Ethics Statement

All datasets used: BIRD, MMQA, and Spider, are publicly available and released under open research licenses. No private, personal, or demographic information is used or inferred in any form. Our pipeline operates solely over structured, non-sensitive tabular data, ensuring privacy and security compliance.

We acknowledge that retrieval-based methods can, in theory, be misused to aggregate or generate misleading information. However, our approach is designed exclusively for academic research in question answering and is not intended for deployment in decision-making or surveillance systems.

To minimize environmental impact, we reuse pretrained retrievers and perform experiments on limited-scale infrastructure. While these datasets are domain-neutral, inherent topical biases may exist, which we mitigate through standardized evaluation protocols and transparent reporting of all results.

This research primarily benefits the NLP community by improving efficiency and interpretability in multi-table reasoning, with minimal risk to end-users. Any future public release of models or code will follow responsible open-source practices, including detailed documentation of intended use to reduce misuse potential. We used AI assistance to help in the writing process.

References

Hasan Alp Caferoğlu and Özgür Ulusoy. 2024. E-sql: Direct schema linking via question enrichment in text-to-sql. *arXiv preprint arXiv:2409.16751*.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. Rsl-sql: Robust schema linking in text-to-sql generation. *CoRR*.

Peter Baile Chen, Tomer Wolfson, Mike Cafarella, and Dan Roth. 2025a. Enrichindex: Using LLMs to enrich retrieval indices offline. In *Second Conference on Language Modeling*.

Peter Baile Chen, Yi Zhang, Mike Cafarella, and Dan Roth. 2025b. 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.

Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2023. Deepjoin: Joinable table discovery with pre-trained language models. *Proceedings of the VLDB Endowment*, 16:2458–2470.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281*.

Carlos Lassance and Stéphane Clinchant. 2022. An efficiency study for splade models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2220–2226, New York, NY, USA. Association for Computing Machinery.

Jinhyuk Lee, Feiyang Chen, Sahil Dua, Daniel Cer, Madhuri Shanbhogue, Iftekhar Naim, Gustavo Hernández Ábrego, Zhe Li, Kaifeng Chen, Henrique Schechter Vera, and 1 others. 2025. Gemini embedding: Generalizable embeddings from gemini. *arXiv preprint arXiv:2503.07891*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023a. Can llm already serve as a database interface? a big bench for large-scale

database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.

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.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023b. Towards general text embeddings with multi-stage contrastive learning.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*.

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*, page 1.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

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.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. 2025a. MURRE: Multi-hop table retrieval with removal for open-domain text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5789–5806, Abu Dhabi, UAE. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025b. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

A Appendix

A.1 Query-table relevance

We employ multiple retrieval strategies to obtain the base tables in order to capture the different aspects of table-query relevance:

Dense Retrieval. Dense retrievers encode both the query and table representations into a shared semantic embedding space. Given a query q and a table t_i, we compute:

$$\text{score}_{\text{dense}}(q, t_i) = \text{sim}(\mathbf{e}_q, \mathbf{e}_{t_i})$$

where $\mathbf{e}_q$ and $\mathbf{e}_{t_i}$ are the embeddings of the query and table respectively, and $\text{sim}(\cdot, \cdot)$ is typically cosine similarity. Dense retrievers excel at capturing semantic similarity and can match queries to tables even when there is no lexical overlap. They are particularly effective at understanding paraphrases, synonyms, and conceptual relationships between the query and table content.

Sparse Retrieval. Sparse retrieval methods score tables based on term frequency and inverse document frequency, capturing exact keyword matches. For a query q and table t_i, the sparse retrieval score is computed based on lexical matching:

$$\text{score}_{\text{sparse}}(q, t_i) = \sum_{w \in q} \text{IDF}(w) \cdot \frac{f(w, t_i) \cdot (k_1 + 1)}{f(w, t_i) + k_1 \cdot (1 - b + b \cdot \frac{|t_i|}{\text{avgdl}})}$$

where $f(w, t_i)$ is the frequency of term w in table t_i, $|t_i|$ is the table length, avgdl is the average table length, and k_1 and b are tuning parameters. Sparse methods provide strong baselines and excel when queries contain specific technical terms or entity names that appear verbatim in table schemas or data.

Hybrid Retrieval. We also explore hybrid approaches that combine dense and sparse signals to leverage both semantic understanding and exact matching capabilities. The hybrid score is computed as a weighted combination:

$$\text{score}_{\text{hybrid}}(q, t_i) = \alpha \cdot \text{score}_{\text{dense}}(q, t_i) + (1 - \alpha) \cdot \text{score}_{\text{sparse}}(q, t_i)$$

where α is a hyperparameter balancing the two retrieval paradigms. Hybrid methods aim to capture the complementary strengths of both approaches: the semantic understanding of dense retrievers and the precision of sparse keyword matching.

Standard retrievers compare the query q against individual tables in isolation, selecting the top-k based solely on query-table semantic similarity. It does not consider the implicit relationships between the tables.

A.2 Prompts and Experimental Setup

A.2.1 Experimental Setup

We run all our experiments on Nvidia A6000 GPU with 48GB of VRAM, and 120 GB RAM.

Table Description Generation

We use the following prompt for generating table descriptions before running our pipeline:

```
prompt = f"""You are a database
    documentation expert. Analyze this
    database table variant and create a
    comprehensive description for
    retrieval and search purposes.

Table Name: {table_sample['table_name']}
Variant ID: {table_sample['variant_id']}
     (Variant #{table_sample['
    variant_index'] + 1})
Columns: {', '.join(table_sample['
    columns'])}
Total Rows in this variant: {
    table_sample['total_rows_in_variant
    ']}

Sample Data (first 10 rows):
{json.dumps(table_sample['sample_data'],
     indent=2)}

Create a detailed table description that
     includes:
    1. What this table represents and
        its main purpose
    2. Columns and their roles


The description should be 2-4 sentences
    long, informative, and optimized for
     database retrieval systems. Focus
    on being descriptive yet concise,
    and highlight what makes this
    variant unique.

Table Variant Description:"""
```

We use GEMINI-1.5-FLASH for generating prompts with temperature=0.3 and max tokens=800.

SQL Generation

We use the following prompt for SQL generation:

```
prompt = f"""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.

AVAILABLE TABLES:
{tables_info}

QUESTION: {question}

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
3. Generate a syntactically correct SQL
    query that answers the question
4. Use appropriate JOINs if multiple
    tables are needed
5. Apply proper filtering, grouping, and
     ordering as required
6. Use the exact column names as shown
    in the schema
7. Be careful with column names that
    contain spaces - use backticks or
    quotes as needed
8. Return ONLY the SQL query without any
     explanation or markdown formatting

SQL QUERY:"""
```

For all LLMs, we use temperature 0.2, top-k sampling with $k = 1$, top-p sampling with $p = 1.0$, maximum token length of 500, and random seed 42. We access GPT-4o-mini via OpenAI's API, Gemini-2.0-Flash via Google's API (Team et al., 2023), and other models via DeepInfra's API.

A.3 Joinability analysis

We adapt the column-level joinability approach from DeepJoin (Dong et al., 2023) to discover additional relevant tables. While DeepJoin fine-tunes a language model on the table corpus, we use a general-purpose embedding model to avoid the computational overhead of fine-tuning while achieving comparable performance. We adopt DeepJoin's column description format:

Name	Pattern
title-colname-stat-col	$table_title$.$colname-stat-col$

We construct a column-level vector index by embedding all columns in the database and indexing them using FAISS for efficient approximate nearest neighbor search. Given the initially retrieved tables, we query this index to find columns with high similarity to those in the retrieved set, explicitly excluding tables already present to avoid redundancy.

This yields a ranked list of candidate tables based on column-level joinability.

To select tables for expansion, we rerank these candidates using Jina-Reranker-v2 based on query-table relevance and add the top-3 tables to the retrieved corpus. We choose to evaluate on 3 of the top models in the MTEB leaderboard: Qwen3-embedding-0.6B (Zhang et al., 2025b), gemini-embedding-001 (Lee et al., 2025), and bge-large-en. Table 6 compares embedding models for this task on MMQA with e5-mistral retrieval. We select bge-large-en for this component due to its strong balance of performance and efficiency. While gemini-embedding-001 achieves marginally higher recall (+0.46pp) and full recall (+1.1pp), bge-large-en offers comparable performance with significantly lower computational cost and greater accessibility for reproduction.

Table 6: Comparison of various expansion models evaluated on MMQA using e5-mistral.

Model	Precision	Recall	Full Recall
bge-large-en	24.31	87.39	74.73
gemini-embedding-001	24.47	87.85	75.83
Qwen3-embedding-0.6B	23.91	85.78	72.56

A.4 Pruning Techniques

We evaluated multiple pruning strategies to optimize table selection, ultimately adopting the Refinement approach described in appendix A.4.1. Table 7 presents results for various pruning methods on MMQA using the UAE retriever. We report on MMQA rather than BIRD or Spider because the performance differences between methods are more pronounced on this dataset, allowing clearer comparison of pruning strategies. The experimental variants are described below:

A.4.1 Pruning With Alpha-Beta

In this method we compute the query-table and table-table scores using a cross-encoder model (Jina reranker). Based on the intuition that columns across two tables are joinable will have high similarity, we compute column-column scores pairwise between all the tables across the retrieved table corpus and take the maximum score across the columns of a pair of tables as the final score between those two tables. We also compute the table-query score using the same cross-encoder and finally get a final score as follows:

$$S(T_i) = \alpha.C(Q, T_i) + \frac{1}{n}\beta.\sum_{j \in N} C(T_j, T_i).C(T_j, Q)$$

where $S(T_i$ is the score of the ith table in the retrieved corpus, α, β are the weighing values, $C(.,.)$ is the cross encoder score, N is the neighborhood of table i and n are the total tables in the neighborhood. We stuck with the values $\alpha = 0.6$ and $\beta = 0.4$ after some hyperparameter tuning.

Table 7: Comparison of various pruning methods evaluated on MMQA using UAE.

Method	Precision	Recall	Full Recall
Alpha-Beta	36.48	82.85	66.07
Adaptive	**44.48**	80.48	62.11
Max Pruning[1]	36.15	82.89	66.21
Max Pruning[2]	36.22	81.95	64.59
Refinement	36.86	**83.19**	**66.61**

A.4.2 Adaptive Pruning

We employ the same scoring mechanism described in appendix A.4.1, but with an adaptive thresholding strategy. Rather than selecting a fixed top-k tables, we compute the mean score across all candidates and retain only those tables scoring above this threshold. This dynamic selection adjusts the retrieved set size based on score distribution: high-quality candidates yield smaller, focused sets, while ambiguous queries produce larger sets.

This approach trades recall for precision. By filtering out below-average candidates, we reduce noise in the final corpus at the potential cost of excluding marginally relevant tables. In practice, this adaptive strategy is particularly effective when precision is critical for downstream SQL generation.

A.4.3 Max Pruning

We compute table-table and query-table scores as described in appendix A.4.1, but aggregate neighborhood scores using maximum instead of mean. The intuition is that a table's relevance should be determined by its strongest connection: if it is highly similar to even one neighbor that is also query-relevant, it likely contains useful information for SQL generation.

We evaluate two variants of this max-pooling strategy, $S(T_i) =$

$$\alpha \cdot C(Q, T_i) + \beta \cdot \max_{T_j \in \mathcal{N}(T_i)} \max(C(T_j, T_i), C(T_j, Q))$$

$$\max\left(C(Q, T_i), \max_{T_j \in \mathcal{N}(T_i)} \max(C(T_j, T_i), C(T_j, Q))\right)$$

Table 8: Retrieve, Expand, and Refine comparison grouped by number of tables.

#Table	#Q	Approach	Dense				SPARSE				Hybrid			
			R	P	FR	SQL	R	P	FR	SQL	R	P	FR	SQL
								BIRD						
		Retrieve	96.7	20.8	96.7	46.7	97.8	20.5	97.8	49.6	98.9	20.8	98.9	50.3
1	**361**	**Expand**	97.9	13.0	97.9	48.1	99.5	12.8	99.5	49.8	99.5	12.8	99.5	49.9
		Refine	97.8	19.6	97.8	50.0	99.2	19.8	99.2	49.8	99.2	19.8	99.2	54.3
		Retrieve	91.8	38.7	85.5	37.8	91.6	37.8	84.9	35.7	92.8	38.3	86.2	37.9
2	**924**	**Expand**	97.6	26.6	95.6	40.8	98.4	25.0	97.0	39.5	97.9	24.9	96.2	39.6
		Refine	95.1	38.0	90.7	39.1	94.9	38.0	90.0	40.5	94.6	37.8	89.5	39.6
		Retrieve	77.7	50.1	52.6	25.1	78.9	50.0	47.0	23.7	82.0	52.3	54.4	26.2
3+	**249**	**Expand**	92.1	37.1	78.2	39.1	92.5	36.5	78.7	34.9	91.6	36.2	76.7	31.7
		Refine	84.3	52.9	59.4	32.2	84.0	52.6	59.0	27.7	83.6	52.4	57.8	25.7
								MMQA						
		Retrieve	76.2	30.5	58.3	34.65	77.7	31.1	59.4	35.39	80.9	32.4	64.4	35.9
2	**2591**	**Expand**	89.1	22.3	79.5	43.4	89.9	22.5	80.7	42.18	90.4	22.6	81.6	39.9
		Refine	85.1	34.0	72.1	42.2	85.6	34.2	72.6	41.8	84.0	33.6	69.9	41.3
		Retrieve	65.7	40.3	30.5	25.17	69.5	42.6	34.1	23.71	69.4	42.6	31.1	24.1
3+	**721**	**Expand**	83.7	32.2	61.9	31.8	85.8	33.0	66.9	32.78	84.7	32.7	63.7	33.5
		Refine	76.4	47.0	47.0	31.1	77.6	47.7	50.5	32.2	72.7	44.8	38.7	31.8
								SPIDER						
		Retrieve	96.8	19.8	96.8	75.5	81.2	16.2	81.2	66.9	87.7	17.5	87.7	52
1	**585**	**Expand**	99.2	12.6	99.2	76.0	99.2	12.4	99.2	78.0	96.1	12.0	96.1	78.6
		Refine	98.6	20.2	98.6	76.0	96.4	19.3	96.4	77.4	95.9	19.2	95.9	77.7
		Retrieve	95.6	38.5	91.1	66.3	82.1	32.9	64.2	48.3	80.3	32.1	61.4	39.9
2	**383**	**Expand**	99.5	25.0	99.0	65.3	99.1	24.8	98.2	66.1	97.7	24.4	95.3	65.3
		Refine	97.6	39.3	95.3	65.3	98.4	39.4	96.9	66.8	97.5	39.0	95.0	65.8
		Retrieve	93.9	58.6	83.3	34.9	75.0	46.4	31.8	30.3	80.1	49.4	42.4	33.3
3+	**66**	**Expand**	100.0	38.9	100.0	36.4	100.0	38.7	100.0	36.4	97.5	37.8	92.4	39.4
		Refine	95.5	59.6	86.4	36.5	97.5	60.3	92.4	37.9	95.5	59.1	86.4	40.9

where $S(T_i)$ is the score of table T_i, $C(\cdot, \cdot)$ denotes cross-encoder similarity, $\mathcal{N}(T_i)$ is the neighborhood of T_i, and α, β are weighting hyperparameters. The first variant combines query relevance and neighborhood strength linearly, while the second uses pure max-pooling across all relevance signals.

A.5 Table Wise Multitable Retrieval

Table 8 presents a granular analysis of REAR's performance stratified by query complexity, measured by the number of gold tables required for correct SQL generation.

The results reveal a clear correlation between query complexity and the magnitude of improvement delivered by our pipeline. For single-table queries, where retrieval is relatively trivial, the baseline **"Retrieve"** approach already achieves near-perfect recall (96.7-98.9% across datasets), with expansion and refinement providing marginal gains primarily in SQL execution accuracy. The benefits of REAR become more pronounced for two-table queries, where the expansion stage consistently boosts full recall by 8-13 percentage points across all datasets (e.g., BIRD Dense for 2 tables: 85.5% $\rightarrow$ 95.6%), while refinement successfully recovers precision losses without sacrificing these recall gains.

Most notably, for the most challenging 3+ table queries, expansion delivers substantial improvements. For example, on BIRD, full recall increases from 52.6% to 78.2% for dense retrievers, demonstrating that join-aware expansion is critical for complex multi-table reasoning. The refinement stage demonstrates its effectiveness in restoring precision while preserving the recall gains achieved through expansion.

Across all query complexities, refinement consistently improves precision by 10-15 percentage points, on BIRD with 2-table queries, precision increases from 26.6% to 38.0% for dense retrievers, while full recall drops only marginally from 95.6% to 90.7%. This precision-recall trade-off is particularly valuable for 3+ table queries, where refinement boosts precision from 37.1% to 52.9% on BIRD, indicating successful removal of weakly related tables introduced during expansion.