PEARL: Front-Loading Relational Chains for Multi-Hop Table Retrieval

Subeen Ho **Hyeongu Kang** **SeongKu Kang** **Susik Yoon**
Computer Science and Engineering
Korea University, Seoul, Korea
{hosubin02, hyeongukang, seongkukang, susik}@korea.ac.kr

Abstract

While large language models (LLMs) have shown strong capabilities in tabular reasoning, retrieving relevant tables remains challenging due to the fragmented and relational structure of real-world data. Existing work typically relies on whole table representations that overlook cross-table semantics induced by join relationships. We propose PEARL, a training-free framework that shifts the paradigm toward vertical partitioning-based sub-table encoding. PEARL augments the retrieval corpus offline by generating multi-hop queries over pre-identified join paths and reorganizing relevant columns into vertically partitioned corpus units, enabling effective multi-table retrieval without query-time LLM inference. Experiments show that PEARL consistently outperforms existing methods, with up to +30.05% gains in R@2 on 3-hop queries. The source code is available at https://github.com/SOOB2NHO/PEARL.

1 Introduction

Tables serve as one of the most effective and universal forms of knowledge in diverse domains (Ji et al., 2025). Identifying tables relevant to a given query or task is a fundamental prerequisite for numerous downstream applications such as question answering (Herzig et al., 2021), fact verification (Schlichtkrull et al., 2021), and reasoning (Wang et al., 2024; Sui et al., 2024). Table retrieval has recently become more important with the rapid adoption of Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Pan et al., 2022) with a large language model (LLM) over structured knowledge bases, where the quality of retrieved tables directly affects the accuracy and reliability of generated responses.

While most existing studies on tabular learning simply assume that relevant tables are available at inference time (Herzig et al., 2021; Schlichtkrull et al., 2021; Wang et al., 2024; Sui et al., 2024),

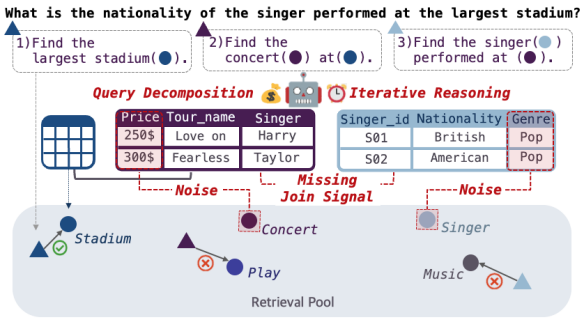

Figure 1: Limitations of existing approaches in multi-hop table retrieval from heterogeneous tables.

real-world tabular data is typically scattered across diverse organizational units with varying schemas and scales (Van Breugel and Van Der Schaar, 2024). In modern data lake environments (Deng et al., 2024; Hai et al., 2023; Fan et al., 2023), which contain a massive number of tables with heterogeneous and high-dimensional attributes, conventional retrieval approaches find difficulties in capturing fine-grained semantic relevance between queries and tables (Chen et al., 2024), making table retrieval a critical bottleneck for downstream applications.

As exemplified in Figure 1, practical queries often require reasoning across multiple tables. Existing table retrieval approaches, however, suffer from fundamental limitations in handling such multi-table retrieval scenarios, as they rely on *inference-time* query decomposition and iterative reasoning paired with offline *whole-table* indexing (Wu et al., 2025; Chen et al., 2024; Boutaleb et al., 2025). In contrast to the tables that can be explicitly retrieved from a given query (e.g., `Stadium`), implicit (e.g., `Concert`) or deeply nested (e.g., `Singer`) ones within the expected multi-hop join path are challenging to identify. This is because query-irrelevant columns are intermingled within the whole-table embeddings, introducing non-trivial noise that hinders a model from capturing the latent join relationships implied by the query.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 57952–57976
October 24-29, 2026 ©2026 Association for Computational Linguistics

Although decomposing the query or performing iterative reasoning at test time may alleviate the problem, such online workarounds introduce substantial computational cost as the number of tables and queries scales up, making them impractical in the real world. Moreover, these limitations intensify when explicit relational schemas are missing or incomplete (Chen et al., 2024), particularly within partial data lakes or raw table dumps.

In this work, we propose **PEARL** (Path-awarE Augmented Retrieval over Linked schema), a simple yet effective multi-hop table retrieval framework that shifts the paradigm from online reasoning over whole-table indexing toward vertical partitioning-based pre-augmented indexing. The key idea of PEARL is to *front-load the recurring, open-ended online computational burden into a fixed, predictable offline indexing phase*. Specifically, by leveraging LLMs to synthesize and pre-index augmented sub-tables via vertical partitioning, PEARL captures potential query semantics and latent cross-table relationships inherent in the original tables beforehand.

In brief, PEARL performs offline indexing by constructing an inter-table graph to discover multi-hop join paths covering both implicit and explicit relations and generating LLM-driven schema-aware queries over each path. It further reorganizes query-relevant columns into vertically partitioned sub-table corpus units that better capture cross-table semantics. The resulting augmented corpus enables standard dense retrievers to outperform existing query decomposition-based methods without training or online LLM inference.

Our key contributions and findings are summarized as follows:

- To the best of our knowledge, this is the first work to incorporate join-path-aware query generation and vertical partitioning into offline indexing for multi-hop table retrieval, effectively front-loading reasoning of relational table chains.
- Comprehensive evaluation on three multi-hop table retrieval benchmarks demonstrates that PEARL outperforms the strongest baseline by 5.84% on average on Recall@k, with a notable gain of up to 30.05% for 3-hop queries.
- By restricting LLM inference to the offline indexing stage, PEARL reduces token consumption by 4.1$\times$ on average compared to the strongest baseline, while achieving up to 79% reduction in total latency. Notably, the indexing cost remains constant, being robust to the inference workload.

2 Preliminary

The goal of multi-hop table retrieval is to identify the set of tables required to answer a query q, where the relevant tables may be connected via explicit or implicit join relationships inherent in the underlying schema structure.

Explicit Join An explicit join captures a relationship that is structurally declared or strongly implied by the schema, such as shared column names, or high value containment between columns. These joins are grounded in schema-level constraints, representing the relational structure of the data.

Implicit Join. An implicit join captures a relationship that is latent in the data but not explicitly represented in the schema. For example, two columns in different tables may refer to the same real-world entity without any declared constraint (e.g., `customer` and `client`). Such joins can only be identified through value or semantic-level similarity of the table contents.

3 Methodology

To efficiently model cross-table dependencies for multi-hop table retrieval, PEARL identifies join paths and constructs vertically partitioned sub-tables for join-aware retrieval using a standard dense retriever, without additional training or online LLM inference. As illustrated in Figure 2, PEARL follows a two-stage pipeline consisting of Offline Indexing and Online Retrieval.

During the offline indexing stage, PEARL discovers join paths that capture how tables can be linked. For each selected path, we generate instructional queries using an LLM and apply vertical partitioning to retain only query-relevant information. This process transforms raw tables into structured corpus units that encode both individual tables and inter-table relationships. At the online retrieval stage, PEARL encodes the user query and retrieves relevant tables by matching it against the join-aware augmented corpus.

3.1 Join-Augmented Vertical Offline Indexing

PEARL exploits LLM-based reasoning, confined within an offline stage in a constant and predictable manner. The offline indexing pipeline consists of three stages: Join Path Discovery, Instructional Query Generation, and Vertical Partitioning and Indexing, with a detailed procedure provided in Algorithm 1 in Appendix B.

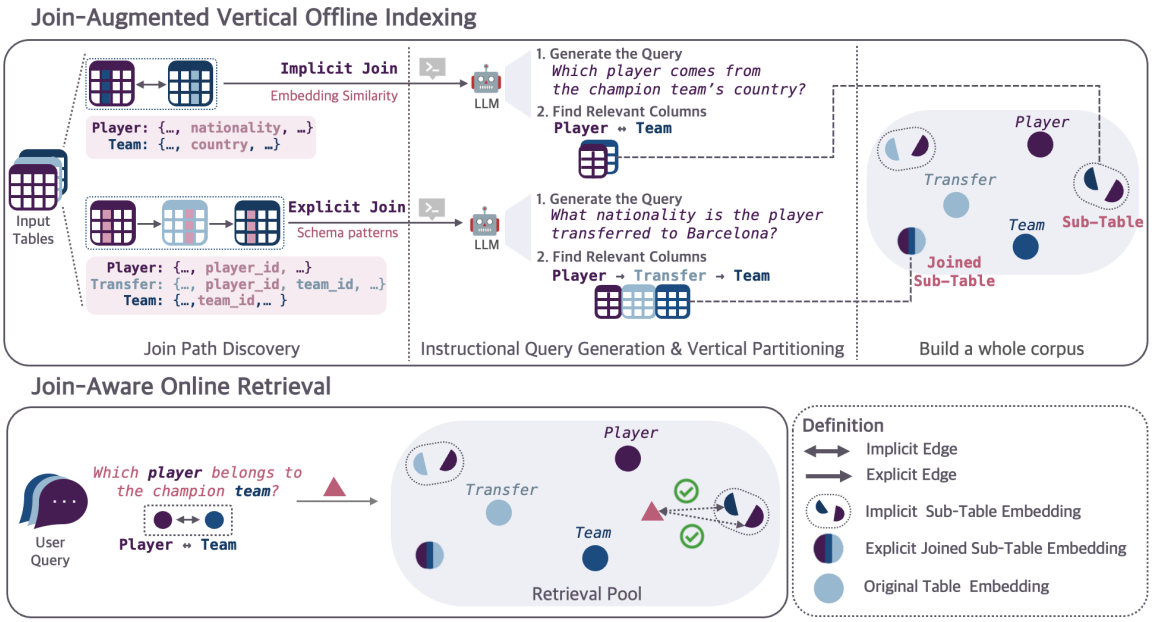

Figure 2: The overall procedure of PEARL.

3.1.1 Join Path Discovery

To enable join-aware query generation, we first construct join paths over the corpus $\mathcal{C}$, consisting of all tables in the dataset. Given a corpus $\mathcal{C}$, Join Path Discovery aims to identify joinable table pairs without schema-level annotations, reflecting real-world retrieval settings. The procedure consists of four phases: Index Construction, Candidate Discovery, Path Construction and Path Selection.

Index Construction Before candidate search, we compute two indexes over all columns in $\mathcal{C}$: (1) a column embedding index and (2) a value embedding index. Both are used for similarity scoring in subsequent stages (Appendix B.1.1 provides details).

Candidate Discovery A candidate edge $e = (c_i, c_j)$ is a column pair drawn from two distinct tables T_i and T_j with a potential latent join relationship. We collect candidate edges via three complementary signals: structural patterns ($\mathcal{P}_S$), column similarity ($\mathcal{P}_B$), and cell value-overlap ($\mathcal{P}_V$), and define the final candidate edge pool as $\mathcal{P} = \mathcal{P}_S \cup \mathcal{P}_B \cup \mathcal{P}_V$. The formal definitions and detailed procedures for each signal are provided in Appendix B.1.2.

Path Construction Each candidate edge $e \in \mathcal{P}$ is categorized into two types. An explicit edge connects tables whose relationship is directly observable from schema-level signals, whereas an implicit edge captures semantically related tables without explicit structural connections.

Paths are assigned to either $\mathcal{P}_{\text{exp}}$ or $\mathcal{P}_{\text{imp}}$ based on their edge composition, yielding two candidate pools passed to Path Selection. Specifically, paths composed entirely of explicit edges are assigned to $\mathcal{P}_{\text{exp}}$, whereas those composed entirely of implicit edges are assigned to $\mathcal{P}_{\text{imp}}$. Mixed paths are categorized based on whether the intermediate table serves as a structural bridge.

Path Selection We apply Lazy Greedy independently over $\mathcal{P}_{\text{exp}}$ and $\mathcal{P}_{\text{imp}}$ to select a diverse and high-quality set of join paths. Each candidate path p is assigned a base score $f(p)$ by aggregating three complementary signals. Specifically, S_{str} captures structural anchors such as foreign-key patterns and identical column names, S_{sem} measures semantic similarity, and S_{ovlp} captures value overlap through Jaccard similarity and containment.

$$f(p) = \begin{cases} S_{\text{str}} + S_{\text{ovlp}} & \text{if } p \in \mathcal{P}_{\text{exp}}, \\ S_{\text{sem}} + S_{\text{ovlp}} & \text{if } p \in \mathcal{P}_{\text{imp}}. \end{cases} \quad (1)$$

Starting from an empty set $\mathcal{P}^*$, the marginal gain $g(p \mid \mathcal{P}^*)$ for adding a path p is defined as:

$$g(p \mid \mathcal{P}^*) = f(p) + \lambda_{\text{div}} \cdot \left|\mathcal{V}(p) \setminus \mathcal{V}(\mathcal{P}^*)\right|, \quad (2)$$

where $\mathcal{V}(\mathcal{P}^*)$ denotes the tables covered by the previously selected paths, and $\mathcal{V}(p)$ denotes the tables covered by the candidate path p. We iteratively select the path p' that maximizes this marginal gain until the budget is exhausted:

$$p' = \arg\max_{p \notin \mathcal{P}^*} g(p \mid \mathcal{P}^*), \quad \text{for } |\mathcal{P}^*| < K. \quad (3)$$

Both λ_{div} (controlling the diversity-quality balance) and K (the corpus budget) are hyperparameters, which are further discussed in Section 4.4.

3.1.2 Instructional Query Generation

To capture multi-hop information needs that span multiple tables, we generate discriminative queries over selected join paths $\mathcal{P}^*$. Given a join path $p \in \mathcal{P}^*$, we prompt an LLM with the full path structure and sampled rows from all tables in p. The LLM jointly generates a query that requires all tables in the path, together with query-relevant column subsets for each table, identifying the minimal columns necessary to answer the query.

$$\left(q,\, \{c_i\}_{i=1}^{h}\right) = \text{LLM}\Big(p,\, \{R_i\}_{i=1}^{h},\, \mathcal{I}\Big)\,, \quad (4)$$

where each $c_i \subseteq \mathcal{S}_i$ denotes the minimal set of columns from table T_i necessary to answer q, $\mathcal{S}_i$ is the full schema of T_i, R_i denotes sampled rows from T_i, and h is the number of tables in the path. $\mathcal{I}$ denotes the task instruction. We impose a constraint that the query requires all tables in p; removing any $T_j \in p$ renders the query unanswerable. Task instructions for implicit and explicit paths are provided in Appendix F.

3.1.3 Vertical Partitioning and Indexing

Wide tables often contain many columns irrelevant to a given query, which dilutes the retrieval signal when the full schema is embedded as a single representation. To address this, PEARL performs vertical partitioning to reorganize tables and generated queries into representations optimized for both structural and semantic matching. Given the query-relevant column subsets c_i from the query generation stage, we perform vertical partitioning over the join path p, constructing query-conditioned sub-tables $\hat{T}_i$ by projecting each table $T_i \in p$ onto its corresponding column subset:

$$\hat{T}_i = \Pi_{c_i}(T_i), \quad i = 1, \ldots, h. \quad (5)$$

For implicit paths, we construct sub-entries that capture pairwise semantic joins between tables. Each projected sub-table $\hat{T}_i$ is indexed as an independent retrieval unit and combined with its associated instructional queries, resulting in multiple embeddings per path. This enables multiple semantic views of each table within the path.

$$\mathbf{e}_{\text{imp},i} = \text{Enc}\Big(q \,\|\, \hat{T}_i\Big)\,, \quad i = 1, \ldots, h. \quad (6)$$

For explicit paths, we construct joined entries that capture structural joins between tables. We unify all projected sub-tables within a path together with the shared query, forming a single multi-table representation. This enables modeling of inter-table relations within a unified representation.

$$\mathbf{e}_{\text{exp}} = \text{Enc}\Big(q \,\|\, \hat{T}_1 \,\|\, \cdots \,\|\, \hat{T}_h\Big)\,. \quad (7)$$

Both embedding types are stored in offline stage. This design narrows the semantic gap between the indexed table representations and actual user needs, while ensuring scalable retrieval via standard approximate nearest neighbor (ANN) search.

3.2 Join-Aware Online Retrieval

At online retrieval time, PEARL uses only precomputed embeddings without additional LLM inference or reasoning. A query q is scored against a unified retrieval pool $\mathcal{U}(T)$ consisting of implicit embeddings $\mathbf{e}_{\text{imp},i}$, explicit embeddings $\mathbf{e}_{\text{exp}}$, and original table embeddings $\mathbf{e}_{\text{orig}}$ derived from join paths containing table T. The relevance score of a table T is computed by max-pooling over all embeddings associated with T:

$$\text{Score}(q, T) = \max_{\mathbf{e} \in \mathcal{U}(T)} \cos(\text{Enc}(q), \mathbf{e})\,. \quad (8)$$

We retrieve the top-k tables ranked by $\text{Score}(q, T)$, enabling training-free retrieval without schema annotations and supporting efficient inference at query time.

4 Experiments

We comprehensively evaluate PEARL on three multi-hop table retrieval benchmarks against established baselines to address seven research questions. The key takeaways are highlighted as follows:

- **Overall Performance Results (RQ1, RQ2, RQ3).** PEARL consistently improves multi-hop retrieval accuracy, especially in higher-hop settings, while significantly reducing online inference cost via offline join-augmented indexing.
- **Ablation Study (RQ4, RQ5).** The gains of PEARL are not driven by simple corpus expansion, but by join-augmented, vertical indexing that captures inter-table relationships. Moreover, explicit and implicit join representations provide complementary signals, and removing either consistently degrades performance.

- **Sensitivity Analysis (RQ6, RQ7).** Performance of PEARL saturates as K increases, with most gains achieved using only half of the join paths. λ_{div} shows that higher diversity benefits complex multi-table queries. Overall, PEARL is robust to hyperparameter variations, supporting practical deployment in real-world settings.

4.1 Experimental setup

Datasets and Evaluation Metrics We evaluate PEARL on three multi-hop table retrieval benchmarks: BIRD (Li et al., 2023), SPIDER (Yu et al., 2018), and MMQA (Wu et al., 2025). Since our method requires no training phase, we can apply it directly to this pool without any dataset-specific supervision. For MMQA, whose tables are not partitioned into separate databases, we use the entire corpus as a unified retrieval pool. For SPIDER and BIRD, we construct a unified corpus by removing database boundaries. We use the same dev split for evaluation as JAR and Greedy-JAR, but perform retrieval over the unified corpus rather than within individual databases. This setup removes the assumption that gold tables are confined to a known database, resulting in a more challenging and realistic search space that better reflects real-world data lakes, where tables from heterogeneous sources are pooled together. We report standard R@k for evaluating retrieval performance.

To discuss the potential impact of augmented tables in evaluating retrieval results, we also report adjusted recall, namely Slot R@k, which collapses all unit-level scores to the table level via the same max-pooling strategy. Additional details and evaluation results are provided in Appendix C.1.1.

Baselines and Implementation Details We compare PEARL with baselines across three paradigms: (1) *text models*, including stella-v5 (Zhang et al., 2024), which demonstrates strong table-to-text alignment on the TARGET benchmark (Ji et al., 2025), and contriever (Izacard et al., 2021), the backbone of decomposition-based retrieval models; (2) *Single-hop retrieval model*, including DTR (Herzig et al., 2021), a strong single-hop table retrieval model; and (3) *multi-hop table retrieval models*, including JAR (Chen et al., 2024), Greedy-JAR (Boutaleb et al., 2025), CORE-T (Soliman et al., 2026), and MURRE (Zhang et al., 2025), which are available with reproducible implementations. All baselines are reproduced using the authors' publicly released implementations.

For architectural consistency, PEARL adopts Stella-v5 as its backbone encoder and Llama 3.3 70B (Grattafiori et al., 2024) as the query generation LLM, with temperature 0.7, a maximum of $1{,}024$ output tokens, and ten sampled rows per table as input context for query generation. The diversity-quality balance weight λ_{div} is fixed at 0.1, and the corpus budget K is set to $\lfloor |\mathcal{C}|/2 \rfloor$ across all datasets. All methods are evaluated under a uniform 512-token input-length constraint.

4.2 Overall Performance Results

RQ1. Does join-augmented vertical indexing improve multi-hop retrieval accuracy? Table 1 summarizes retrieval performance across SPIDER, MMQA, and BIRD, where PEARL achieves the best performance in most cases, outperforming the strongest baseline by 3.46% on SPIDER and 9.37% on MMQA under the unified setting. On BIRD, we observe minor fluctuations across configurations, which we analyze in Appendix D.2.

Since PEARL is built on the Stella-v5 encoder, the performance gap between the two isolates the effect of vertical partitioning. Under the unified setting, PEARL improves over Stella-v5 by 6.77% on SPIDER, 10.81% on MMQA, and 10.39% on BIRD in R@2. Notably, even the w/o joined entry variant, which independently embeds each subtable unit without constructing joined entries, consistently outperforms Stella-v5 across all benchmarks. These results suggest that PEARL reduces semantic noise by filtering irrelevant columns, improving retrieval performance.

To further verify that these gains are not simply an artifact of joined entries covering multiple gold tables, we additionally report adjusted recall in Table 2. Under this unit-level metric, PEARL consistently outperforms the Stella-v5 backbone across all three benchmarks in the unified setting, indicating that the improvements stem from more effective noise reduction rather than artificially inflated recall due to bundled gold tables. Full per-hop results are provided in Appendix C.1.1.

RQ2. Does the effectiveness of PEARL increase with query hop complexity?

Table 1 presents retrieval performance across 2-hop and 3-hop configurations. The performance gap between PEARL and baseline methods consistently increases with the number of hops. In the 2-hop setting, PEARL shows consistent improvements on SPIDER and MMQA, while a slight degradation is observed on BIRD.

Setting	Model	SPIDER		MMQA		BIRD	
		R@2	R@5	R@2	R@5	R@2	R@5
3-Hop	Contriever	56.10	91.10	38.10	55.50	45.80	78.70
	Stella-v5	58.89	87.78	47.30	71.70	52.17	85.33
	JAR	62.80	94.40	44.70	64.20	54.30	86.20
	Greedy-JAR	60.00	90.60	44.50	61.80	54.30	**88.30**
	DTR	56.67	89.44	44.39	66.46	48.33	80.83
	MURRE	52.78	85.00	34.05	56.69	45.83	74.83
	CORE-T	55.56	92.22	45.80	64.38	52.67	84.33
	PEARL	**81.67** (▲ 30.05%)	**100.00** (▲ 5.93%)	**58.09** (▲ 22.81%)	**80.43** (▲ 12.18%)	**59.17** (▲ 8.97%)	87.33 (▼ 1.10%)
2-Hop	Contriever	78.20	97.80	49.10	63.60	62.00	84.00
	Stella-v5	76.97	96.06	55.76	72.63	64.67	92.29
	JAR	85.00	98.10	55.90	65.20	**80.30**	92.50
	Greedy-JAR	86.50	95.70	56.80	64.00	79.10	93.70
	DTR	80.28	96.18	52.47	66.37	62.62	87.38
	MURRE	74.80	92.22	49.15	70.20	56.20	82.46
	CORE-T	80.21	96.97	54.51	67.23	72.78	92.57
	PEARL	**89.57** (▲ 3.55%)	**99.49** (▲ 1.42%)	**60.57** (▲ 6.64%)	**75.58** (▲ 4.06%)	72.81 (▼ 9.33%)	**94.38** (▲ 0.73%)
Unified	Contriever	84.80	97.90	46.30	61.50	62.80	84.60
	Stella-v5	85.46	97.42	53.10	71.72	66.71	91.36
	JAR	87.80	98.20	53.10	64.60	**75.10**	90.80
	Greedy-JAR	88.20	97.00	53.80	63.20	74.40	91.70
	DTR	86.99	97.45	50.36	65.96	65.80	87.78
	MURRE	77.27	92.60	45.62	66.82	58.84	82.51
	CORE-T	87.58	98.22	52.16	66.13	74.71	92.54
	PEARL	**91.25** (▲ 3.46%)	**99.33** (▲ 1.13%)	**58.84** (▲ 9.37%)	**76.21** (▲ 6.26%)	73.64 (▼ 1.95%)	**93.36** (▲ 0.89%)
Ablation of PEARL (Unified)							
	w/o joined entry (= w/ implicit-only)	87.55	99.12	54.21	73.34	69.66	92.73
	w/ explicit-only	91.62	99.18	57.62	74.05	74.10	93.81
	Random-view	84.66	97.31	52.39	70.80	66.25	90.78
	Random-query	85.99	97.05	54.97	73.15	68.73	92.51

Table 1: Standard R@k across 3-Hop, 2-Hop, and Unified configurations. Best results are in **bold**, second best underlined. ▲ and ▼ indicate improvement and degradation over the strongest baseline.

Model	SPIDER		MMQA		BIRD	
	R@2	R@5	R@2	R@5	R@2	R@5
Stella-v5	85.46	97.42	53.10	71.72	66.71	91.36
PEARL	**88.48**	**98.75**	**54.26**	**72.88**	**69.66**	**92.73**

Table 2: Adjusted recall (Slot R@k) under the Unified setting. Full per-hop results are in Appendix C.1.1.

In the 3-hop setting, performance gains become more pronounced, with PEARL outperforming the strongest baseline in R@2 by 30.05%, 22.81%, and 8.97% on SPIDER, MMQA, and BIRD, respectively, indicating stronger benefits under higher cross-table reasoning complexity where join-aware vertical indexing better captures inter-table relational structure.

This trend is particularly pronounced for MURRE, an online iterative baseline. PEARL's margin over MURRE widens with hop count across all three benchmarks. This suggests that precomputing join-aware, vertically partitioned units becomes increasingly effective as relational reasoning deepens, whereas the cost of online reasoning compounds with each additional hop.

RQ3. Does PEARL reduce overall latency and LLM cost compared to baselines?

We investigate whether PEARL reduces online LLM dependency compared to decomposition-based retrieval methods. Since Greedy-JAR consistently outperforms JAR in efficiency (Boutaleb et al., 2025), we use it as the representative baseline for latency comparison.

Data	Phase	Latency (s)					Token Consumption (K)			
		0.25|C|	0.50|C|	0.75|C|	1.00|C|	GJAR	Type	PEARL	GJAR	Ratio
SPIDER	Offline	222	282	335	399	—	Prompt	97	450	4.6×
	Online	4	4	4	5	894	Comp.	11	36	3.2×
	Total	**226**	**286**	**339**	**404**	**894**	**Total**	**108**	**486**	**4.5×**
MMQA	Offline	774	1,200	1,637	2,063	—	Prompt	675	1,458	2.2×
	Online	25	32	39	46	3,225	Comp.	94	139	1.5×
	Total	**799**	**1,232**	**1,675**	**2,109**	**3,225**	**Total**	**768**	**1,598**	**2.1×**
BIRD	Offline	228	297	379	458	—	Prompt	118	674	5.7×
	Online	5	5	6	6	1,399	Comp.	10	61	5.9×
	Total	**233**	**302**	**385**	**464**	**1,399**	**Total**	**129**	**735**	**5.7×**

Table 3: Efficiency comparison of PEARL and GJAR. Left: Latency breakdown (seconds). Right: Token consumption (K) for offline LLM calls. All efficiency measurements use GPT-4o mini as the LLM backbone for both evaluations. GJAR has no offline phase.

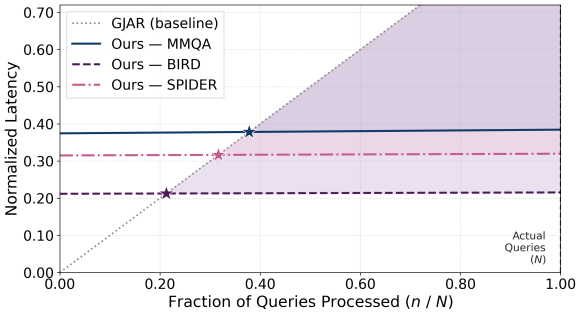

Figure 3: Latency (normalized by GJAR) vs. query fraction. Stars denote break-even points.

As shown in Table 3, PEARL shifts all LLM latency to the offline stage, reducing online latency by over 99% across all datasets. Figure 3 shows that this design reaches a break-even point after processing only 22–35% of the incoming query workload and achieves 62–79% total latency reduction at full query volume, with efficiency gains from offline amortization becoming more pronounced as query volume increases. The storage and indexing overhead introduced by this offline design remains modest and is analyzed in Appendix D.3.

We additionally measure token consumption to evaluate LLM usage efficiency. As shown in Table 3, PEARL consistently achieves higher retrieval performance (Appendix D.1), while PEARL requires fewer LLM tokens. Averaged across the three benchmarks, PEARL reduces the total token consumption by 4.1× compared to Greedy-JAR. These results indicate that PEARL is not only faster at query time but also more token-efficient than decomposition-based retrieval approaches.

4.3 Ablation Study

We conduct ablation studies to analyze the effect of join-aware augmentation and the roles of explicit and implicit representations in PEARL.

RQ4. Are the gains from PEARL affected by corpus expansion? We compare PEARL against two corpus-expansion variants. Random-view augments the corpus with randomly projected subtables as a corpus-expansion-only variant (Appendix B.3). Random-query augments each table with five discriminative questions, offering diverse semantic views without modeling join-aware relationships (Appendix E).

Despite producing a corpus of comparable size, Random-view performs worse than Stella-v5 across all datasets in Table 1. In contrast, PEARL consistently outperforms both Stella-v5 and Random-view, surpassing Random-view on R@2 by 7.78%, 12.31%, and 11.15% on SPIDER, MMQA, and BIRD, respectively. These results show that retrieval gains do not arise merely from increasing corpus size, but from join-aware augmentation enabled by vertical partitioning, which isolates and organizes join-relevant information.

Although Random-query expands the corpus more aggressively than PEARL, PEARL consistently outperforms it on R@2 across all datasets, achieving relative gains of 6.12%, 7.04%, and 7.14% on SPIDER, MMQA, and BIRD, respectively. While Random-query improves the semantic coverage of individual tables, it lacks structural signals across tables. In contrast, PEARL generates queries grounded in discovered join paths, modeling inter-table relationships. The same observation holds when compared with CORE-T, an offline augmentation method that attaches LLM-generated context to whole-table representations. PEARL outperforms it on five of the six unified metrics, most notably by 12.81% on R@2 and 15.24% on R@5 for MMQA. These results suggest that structural awareness is more important than increasing semantic diversity alone in multi-hop scenarios.

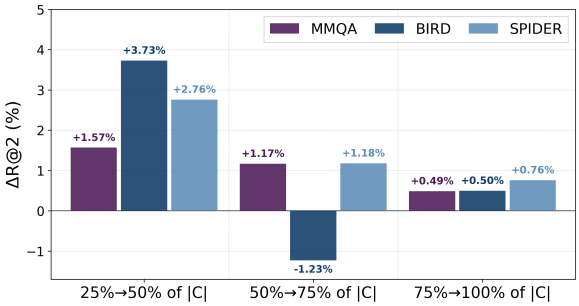

Figure 4: Marginal R@2 gain (ΔR@2) per corpus-budget increment under the full augmentation setting across three datasets.

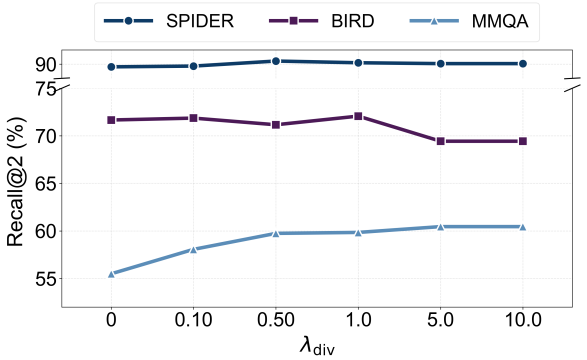

Figure 5: Sensitivity analysis on the diversity-quality trade-off hyperparameter λ_{div} across three benchmarks.

RQ5. Are both explicit and implicit join necessary for effective multi-hop retrieval?

We compare PEARL with variants that retain only explicit or implicit join representations. As shown in Table 1, removing either component consistently degrades performance. Explicit-only variants outperform implicit-only ones on all three datasets (SPIDER +4.65%, MMQA +6.29%, BIRD +6.37%), indicating that joined entries provide a stronger individual signal than sub-entries alone. Nevertheless, neither variant consistently matches the full PEARL across datasets under Slot R@2 (Table 5), confirming that joined entries anchor structural chains, while sub-entries surface latent semantic links, making both signals essential for robust multi-hop retrieval. Appendix C.1.3 further validates the asymmetric encoding, by encoding implicit paths as joined entries.

4.4 Sensitivity Analysis

We analyze the sensitivity of PEARL to two key hyperparameters: the corpus budget K and the diversity-quality balance weight λ_{div}.

RQ6. How does retrieval performance vary with the corpus budget K? Figure 4 reports the marginal R@2 gain as the fraction of sampled join paths increases from 25% to 100% of the corpus size $|\mathcal{C}|$. The largest gains are observed when sampling the first half of join paths (25% to 50%), with improvements of 1.57%, 3.73%, and 2.76% on MMQA, BIRD, and SPIDER, respectively. Beyond this point, the marginal benefit decreases substantially and approaches saturation in the 75% to 100% range, with BIRD even showing a slight performance drop between 50% and 75%. These results suggest that sampling half of the available join paths is sufficient to capture most retrieval gains, justifying it as the default K in all experiments.

RQ7. How does the diversity-quality balance weight λ_{div} affect retrieval performance?

Figure 5 reports performance across varying λ_{div}. Since λ_{div} controls the balance between query diversity and join quality, the sensitivity patterns reflect the relative importance of each factor per dataset. MMQA consistently improves as λ_{div} increases, suggesting that broader query coverage is particularly important in its complex multi-table retrieval setting. In contrast, BIRD shows sharp performance drop at higher values, indicating that join quality is more dominant and excessive diversity is detrimental. SPIDER remains relatively insensitive across different values, suggesting that its retrieval performance is stable across different diversity-quality balances. Importantly, all three datasets converge to identical performance beyond $\lambda_{\text{div}} = 5.0$, indicating that excessively strong diversity penalization provides no additional benefit.

5 Related Work

With the rapid proliferation of large-scale data lakes integrating heterogeneous sources, table retrieval has become a critical problem for identifying relevant tables given a query (Deng et al., 2024; Hai et al., 2023; Fan et al., 2023). Beyond single-table retrieval, recent work has explored multi-hop table retrieval, which requires jointly retrieving multiple related tables. We distinguish two types of join relationships, explicit and implicit, and categorize existing methods based on how they model inter-table relationships.

Explicit Join Modeling Explicit join-based approaches leverage schema information to identify related table sets or discover candidate joins across tables (Chen et al., 2024; Boutaleb et al., 2025; Soliman et al., 2026; Guo et al., 2025). These methods exploit explicit relational structure to narrow the search space before downstream reasoning.

Representative methods explore join candidates using metadata associated with natural language queries (Guo et al., 2025) or structurally model foreign-key relationships and schema consistency across tables (Chen et al., 2024; Boutaleb et al., 2025; Soliman et al., 2026). However, these approaches often rely on monolithic whole-table encoding before passing tables to the model, which often exceeds the context window limitations of LLMs in real-world data lake environments with massive numbers of columns and rows. Moreover, indiscriminately incorporating irrelevant columns introduces substantial semantic noise into the retrieval process.

Implicit Join Alignment Considering real-world environments where structural information is incomplete or not explicitly available (Wang et al., 2024), prior work has increasingly explored implicit semantic alignment approaches that leverage semantic similarities across data or the reasoning capabilities of LLMs, such as dynamically exploring retrieval paths at query time (Wu et al., 2025; Sun et al., 2026). Unlike explicit join modeling that relies on predefined relational metadata, these methods attempt to infer latent inter-table relationships through iterative reasoning and semantic matching. However, they typically require repeated LLM invocations for each query, resulting in substantial latency and computational cost as the number of tables and query complexity increase.

Related topics beyond the scope of this section, including single-hop table retrieval and LLM-based tabular reasoning, are discussed in Appendix A.

6 Conclusion

We presented PEARL, a table retrieval framework that bridges the semantic gap between tables and multi-hop queries by constructing a join-aware corpus. By generating implicit sub-table views and explicit joined entries for structurally or semantically related table pairs, PEARL enables a standard embedding model to retrieve multi-hop evidence without any online LLM involvement or iterative query decomposition.

Experiments on SPIDER, BIRD, and MMQA demonstrate that PEARL consistently improves retrieval performance, beyond simple corpus augmentation. Furthermore, PEARL reduces online latency by over 99% and total latency by up to 79%, while cutting token consumption by up to $4.1\times$ on average compared to Greedy-JAR, and its efficiency advantage grows as query volume scales.

Limitations

While PEARL demonstrates strong retrieval performance, table retrieval remains an open problem for complex queries that require reasoning over multiple joinable tables. We currently limit join paths up to a maximum length of three, which provides sufficient coverage of most benchmark cases. However, real-world settings may involve longer dependency chains. We leave the exploration of more scalable path construction and richer inter-table connections as future work.

As noted in Appendix D.2, although PEARL is fully training-free and can be applied directly to an entire table corpus without dataset-specific supervision, our evaluation is limited to academic benchmarks. Further case studies on real-world domain-specific scenarios would be a promising direction to explore the practical applicability and robustness in diverse deployment settings.

From a deployment perspective, PEARL maintains a larger index than whole-table retrieval, as each selected join path introduces additional corpus units. Although this overhead is bounded by the corpus budget K and remains modest in our experiments (Appendix D.3), memory-constrained deployments may require a smaller budget. Exploring the trade-off between storage efficiency and retrieval effectiveness is an interesting direction for future work. As our incremental indexing evaluation with a single corpus expansion step in Appendix D.4 suggests that PEARL can efficiently update the index without full re-indexing, its behavior under continuous corpus evolution is worth further exploration. Addressing this challenge is closely related to developing more flexible offline indexing strategies. More broadly, PEARL adopts a fully offline indexing pipeline, precomputing all selected paths before deployment. A hybrid workflow that precomputes only high-value paths while resolving the remaining paths at query time could provide a better balance between indexing cost and query efficiency, as discussed in Appendix D.5.

Acknowledgments

This work was supported by Korea University–KT (Korea Telecom) R&D Center, the Institute of Information & Communications Technology Planning & Evaluation (IITP) grants (IITP-2026-RS-2020-II201819, IITP-2026-RS-2024-00436857, IITP-2026-RS-2025-02304828) and the National Research Foundation of Korea (RS-2024-00406320).

References

Allaa Boutaleb, Bernd Amann, Rafael Angarita, and Hubert Naacke. 2025. Exploring multi-table retrieval through iterative search. *arXiv preprint arXiv:2511.13418*.

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.

Tsung-Hsiang Chou, Chen-Jui Yu, Shui-Hsiang Hsu, and Yao-Chung Fan. 2026. Cgpt: Cluster-guided partial tables with llm-generated supervision for table retrieval. In *Proceedings of the ACM Web Conference 2026*, pages 8585–8588.

Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, et al. 2024. Lakebench: A benchmark for discovering joinable and unionable tables in data lakes. *Proceedings of the VLDB Endowment*, 17(8):1925–1938.

Grace Fan, Jin Wang, Yuliang Li, and Renée J Miller. 2023. Table discovery in data lakes: State-of-the-art and future directions. In *Companion of the 2023 International Conference on Management of Data*, pages 69–75.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. Splade v2: Sparse lexical and expansion model for information retrieval. *arXiv preprint arXiv:2109.10086*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yuxiang Guo, Zhonghao Hu, Yuren Mao, Baihua Zheng, Yunjun Gao, and Mingwei Zhou. 2025. Birdie: Natural language-driven table discovery using differentiable search index. *arXiv preprint arXiv:2504.21282*.

Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. 2023. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12571–12590.

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.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4320–4333.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*.

Xingyu Ji, Parker Glenn, Aditya G Parameswaran, and Madelon Hulsebos. 2025. Target: Benchmarking table retrieval for generative tasks. *arXiv preprint arXiv:2505.11545*.

Eugenie Y Lai, Yeye He, and Surajit Chaudhuri. 2025. Auto-prep: Holistic prediction of data preparation steps for self-service business intelligence. *arXiv preprint arXiv:2504.11627*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. 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.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, et al. 2022. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and James Hendler. 2022. End-to-end table question answering via retrieval-augmented generation. *arXiv preprint arXiv:2203.16714*.

Jiarui Qin, Weinan Zhang, Rong Su, Zhirong Liu, Weiwen Liu, Ruiming Tang, Xiuqiang He, and Yong Yu. 2021. Retrieval & interaction machine for tabular data prediction. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1379–1389.

Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*, pages 232–241. Springer.

Michael Schlichtkrull, Vladimir Karpukhin, Barlas Oguz, Mike Lewis, Wen-tau Yih, and Sebastian Riedel. 2021. Joint verification and reranking for open fact checking over tables. In *Proceedings of the*

59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 6787–6799.

Hassan Soliman, Vivek Gupta, Dan Roth, and Iryna Gurevych. 2026. Core-t: Coherent retrieval of tables for text-to-sql. *arXiv preprint arXiv:2601.13111*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Chaojie Sun, Bin Cao, Tiantian Li, Chenyu Hou, Ruizhe Li, and Jing Fan. 2026. Fgtr: Fine-grained multi-table retrieval via hierarchical llm reasoning. *arXiv preprint arXiv:2603.12702*.

Boris Van Breugel and Mihaela Van Der Schaar. 2024. Why tabular foundation models should be a research priority. *arXiv preprint arXiv:2405.01147*.

Zilong Ryan Wang, Hao Zhang, Chun-Liang Li, Julian M Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *International Conference on Learning Representations*.

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*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pages 174–184.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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.

Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2024. Jasper and stella: distillation of sota embedding models. *arXiv preprint arXiv:2412.19048*.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. 2025. 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.

Lei Zheng, Ning Li, Xianyu Chen, Quan Gan, and Weinan Zhang. 2023. Dense representation learning and retrieval for tabular data prediction. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3559–3569.

A Additional Related Work

A.1 Single-hop Table Retrieval

Unlike real-world user queries, which often require reasoning over relationships across multiple tables (Lai et al., 2025), most existing studies have primarily focused on single-table retrieval scenarios. Traditional keyword-based approaches (Formal et al., 2021; Robertson and Walker, 1994) and embedding-based retrieval models (Herzig et al., 2020; Qin et al., 2021; Zheng et al., 2023; Herzig et al., 2021) typically adopt a single table representation strategy, where the entire schema and sampled rows are compressed into a single vector representation, limiting their ability to capture inter-table relationships. Consequently, they struggle to effectively model the multi-table reasoning requirements commonly observed in real-world open-domain environments (Chen et al., 2024).

A.2 LLM Reasoning on Tabular Data

Recent studies have explored approaches where LLMs directly transform retrieved tables (Wang et al., 2024) or decompose tables into finer-grained units for more precise matching and reasoning (Ye et al., 2023; Sun et al., 2026). Although these methods achieve strong reasoning performance across various tabular tasks, they still suffer from substantial computational overhead due to repeated online LLM inference for each user query, limiting their applicability for latency-sensitive applications.

To mitigate this, prior work has proposed leveraging LLMs during the offline stage to generate synthetic queries for retrieval augmentation (Chou et al., 2026). However, such approaches still fundamentally fail to resolve the semantic noise introduced by irrelevant columns and remain confined to single-table retrieval settings, making effective modeling of inter-table relationships difficult.

B Details of Offline Corpus Augmentation

This section describes the implementation details of PEARL's offline corpus augmentation pipeline, summarized in Algorithm 1.

B.1 Join Path Discovery: Implementation Details

Join path discovery consists of four stages: index construction, candidate discovery, path construction, and path selection. Each stage is described in detail in the following subsections.

Algorithm 1: Offline indexing of PEARL

Input: Table corpus $\mathcal{C}$, LLM instructions $\mathcal{I}_{\text{imp}}, \mathcal{I}_{\text{exp}}$, budget K, diversity weight λ_{div}
Output: Augmented corpus $\mathcal{U}$

Initialize $\mathcal{U} \leftarrow \emptyset$

```
// Step 1: Join Path Discovery
```
$\mathcal{P} \leftarrow \mathcal{P}_S \cup \mathcal{P}_B \cup \mathcal{P}_V$
```
// Candidate Discovery
```
Classify $\mathcal{P}$ into $\mathcal{P}_{\text{exp}}$, $\mathcal{P}_{\text{imp}}$
```
// Path Construction
```
$\mathcal{P}^* \leftarrow \text{LazyGreedy}(\mathcal{P}_{\text{exp}}, \mathcal{P}_{\text{imp}}; K, \lambda_{\text{div}}, h \in \{2, 3\})$
```
// Path Selection
```
for *each path* $p = (T_1, \ldots, T_h) \in \mathcal{P}^*$ **do**
  `// Step 2: Instructional Query Generation`
  if $p \in \mathcal{P}_{\text{imp}}$ **then**
    $(q, \{c_i\}_{i=1}^h) \leftarrow \text{LLM}(p, \{R_i\}_{i=1}^h, \mathcal{I}_{\text{imp}})$
  else
    $(q, \{c_i\}_{i=1}^h) \leftarrow \text{LLM}(p, \{R_i\}_{i=1}^h, \mathcal{I}_{\text{exp}})$
  end
  `// Step 3: Vertical Partitioning and indexing`
  $\hat{T}_i \leftarrow \Pi_{c_i}(T_i)$ **for each** $i \in \{1, \ldots, h\}$
  if $p \in \mathcal{P}_{\text{imp}}$ **then**
    for $i \leftarrow 1$ **to** h **do**
      $\mathbf{e}_{\text{imp},i} \leftarrow \text{Enc}(q \,\|\, \hat{T}_i)$
      `// per-table sub-entry`
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathbf{e}_{\text{imp},i}\}$
    end
  else
    $\mathbf{e}_{\text{exp}} \leftarrow \text{Enc}(q \,\|\, \hat{T}_1 \,\|\, \cdots \,\|\, \hat{T}_h)$
    `// joined entry`
    $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathbf{e}_{\text{exp}}\}$
  end
end
return $\mathcal{U}$

B.1.1 Details of Index Construction

- **Column Embedding Index:** Normalized column names are indexed with FAISS for approximate nearest-neighbor search (ANN), supporting Candidate Discovery ($\mathcal{P}_B$) and semantic scoring S_{sem} in Path Selection.
- **Value Embedding Index:** Five sampled cell values per column are encoded and mean-pooled into a single value embedding for semantic scoring S_{sem} in Path Selection.

B.1.2 Details of Candidate Discovery

As introduced in Section 3.1.1, candidate edges e are collected via three complementary signals, each capturing a distinct aspect of joinability.

- **Value-Overlap ($\mathcal{P}_V$):** Using an inverted value index, we retrieve edges with Jaccard similarity over 0.10 on sampled cell values.

- **Structural Pattern ($\mathcal{P}_S$):** Using an inverted index over normalized column names, we retrieve edges whose names match a FK naming convention or share an identical normalized token across different tables.

- **Column Similarity ($\mathcal{P}_B$):** Using the column embedding index built in Section B.1.1, we retrieve column edges whose similarity exceeds an adaptive threshold τ_{sim}, defined as the 85th percentile of corpus-wide similarities.

B.1.3 Details of Join Path Construction

Rationale for Join Path Construction. As shown in Table 4, 2-hop and 3-hop queries account for the majority of cases across all benchmarks. Accordingly, we restrict join path generation to 2-hop and 3-hop paths. Table 4 also highlights the diversity of path connectivity patterns. Although explicit-only paths dominate across datasets (70.2 to 88.7%), implicit-only paths (11.3 to 21.3%) and mixed explicit–implicit paths (up to 8.5%) are observed. Our pipeline supports all three connectivity patterns by modeling explicit and implicit edges and combining them during join path construction.

Implementation Details of Path Construction. PEARL generates 2-hop and 3-hop join paths from explicit and implicit edges. Candidate edges are enumerated from $\mathcal{P}_S \cup \mathcal{P}_B \cup \mathcal{P}_V$ and scored to obtain s_{str} and s_{sem} for each path p. Prior to Lazy Greedy selection, candidate paths are filtered using a quality threshold τ to ensure that only sufficiently reliable join edges are included in each pool.

2-hop paths. Each candidate edge e corresponds to a 2-hop path, and only paths satisfying $f(p) \geq \tau_{\text{2h}}$ are retained: paths induced from $\mathcal{P}_S \cup \mathcal{P}_V$ form the explicit candidate pool $\mathcal{P}_{\text{exp}}$, while those from $\mathcal{P}_B \cup \mathcal{P}_V$ form the implicit candidate pool $\mathcal{P}_{\text{imp}}$.

3-hop paths. A table sequence (T_i, T_k, T_j) is formed by selecting an intermediate table T_k and two of its neighbors in the join graph induced by 2-hop paths. To prevent weak edges from entering longer chains, a stricter threshold $\tau_{\text{3h}} > \tau_{\text{2h}}$ is applied to 3-hop paths:

$$\min\big(f(p_{ik}),\ f(p_{kj})\big) \geq \tau_{\text{3h}}, \tag{9}$$

where $p_{ik}, p_{kj} \in \mathcal{P}_{\text{exp}}$ for explicit 3-hop paths and $p_{ik}, p_{kj} \in \mathcal{P}_{\text{imp}}$ for implicit 3-hop paths. For each intermediate table T_k, we retain only the highest-scoring triple (T_i, T_k, T_j).

Metric	MMQA	SPIDER	BIRD
Hop Distribution			
1-hop	0.0%	55.6%	23.7%
2-hop	78.3%	38.0%	61.0%
3-hop	19.7%	5.8%	13.0%
4+-hop	2.0%	0.6%	2.3%
2-hop + 3-hop	**98.0%**	**43.8%**	**74.0%**
Path Connectivity (Multi-hop)			
Explicit-only	70.2%	88.7%	75.8%
Implicit-only	21.3%	11.3%	20.5%
Mixed	8.5%	0.0%	3.6%

Table 4: Distribution of gold join paths across the evaluated benchmarks.

B.2 Details of Table Serialization

Both Instructional Query Generation and Vertical Partitioning and Indexing use the same serialized representation of each table. To preserve join-relevant evidence under a limited row budget, we reorder table rows based on the value overlap identified along each join edge.

Rows whose join-column values fall within the overlap are placed first, while the remaining rows retain their original order. For a table at the center of a 3-hop path, which is incident to two join edges, we take the union of the overlap sets from both edges. Thus, a row is moved to the front if it contains a value matching either edge. We then serialize the first ten rows of the reordered table to meet the input length limits of both the query generation LLM and the encoder. By reordering the rows before applying this truncation, join-relevant rows are prioritized in the table representation.

B.3 Implementation Details of Random-View Augmentation

The Random-view baseline constructs an augmented corpus by replacing PEARL's join-aware entries with randomly projected sub-tables. For each table $T_i \in \mathcal{C}$, we randomly sample a non-empty subset of columns c from its schema S_i:

$$c \sim \text{Uniform}(\{\, c \subseteq S_i \mid 1 \leq |c| \leq |S_i| \,\}). \tag{10}$$

This yields a projected sub-table $\hat{T}_i = \Pi_c(T_i)$, which is encoded without any associated instructional query to produce the random-view entry:

$$\mathbf{e}_{\text{rand}} = \text{Enc}\Big(\hat{T}_i\Big). \tag{11}$$

To ensure a fair comparison, the total number of random-view entries is matched to the augmented corpus size of PEARL:

$$|\{\mathbf{e}_{\text{rand}}\}| = |\{\mathbf{e}_{\text{imp}}\}| + |\{\mathbf{e}_{\text{exp}}\}|. \tag{12}$$

Setting	Model	SPIDER		MMQA		BIRD	
		Slot R@2	Slot R@5	Slot R@2	Slot R@5	Slot R@2	Slot R@5
3-Hop	Contriever	56.10	91.10	38.10	55.50	45.80	78.70
	Stella-v5	58.89	87.78	47.30	71.70	52.17	85.33
	JAR	62.80	94.40	44.70	64.20	**54.30**	86.20
	Greedy-JAR	60.00	90.60	44.50	61.80	**54.30**	**88.30**
	DTR	56.67	89.44	44.39	66.46	48.33	80.83
	MURRE	52.78	85.00	34.05	56.69	45.83	74.83
	CORE-T	55.56	92.22	45.80	64.38	52.67	84.33
	PEARL	**64.44** (▲ 2.61%)	**100.00** (▲ 5.93%)	**49.41** (▲ 4.46%)	**75.09** (▲ 4.73%)	53.83 (▼ 0.87%)	85.83 (▼ 2.80%)
2-Hop	Contriever	78.20	97.80	49.10	63.60	62.00	84.00
	Stella-v5	76.97	96.06	55.76	72.63	64.67	92.29
	JAR	85.00	98.10	55.90	65.20	**80.30**	92.50
	Greedy-JAR	86.50	95.70	56.80	64.00	79.10	93.70
	DTR	80.28	96.18	52.47	66.37	62.62	87.38
	MURRE	74.80	92.22	49.15	70.20	56.20	82.46
	CORE-T	80.21	96.97	54.51	67.23	72.78	92.57
	PEARL	**86.88** (▲ 0.44%)	**98.73** (▲ 0.64%)	**56.87** (▲ 0.12%)	**72.81** (▲ 0.25%)	68.42 (▼ 14.80%)	**93.90** (▲ 0.21%)
Unified	Contriever	84.80	97.90	46.30	61.50	62.80	84.60
	Stella-v5	85.46	97.42	53.10	71.72	66.71	91.36
	JAR	87.80	98.20	53.10	64.60	**75.10**	90.80
	Greedy-JAR	88.20	97.00	53.80	63.20	74.40	91.70
	DTR	86.99	97.45	50.36	65.96	65.80	87.78
	MURRE	77.27	92.60	45.62	66.82	58.84	82.51
	CORE-T	87.58	98.22	52.16	66.13	74.71	92.54
	PEARL	**88.48** (▲ 0.32%)	**98.75** (▲ 0.54%)	**54.26** (▲ 0.86%)	**72.88** (▲ 1.62%)	69.66 (▼ 7.24%)	**92.73** (▲ 0.21%)
Ablation of PEARL (Unified)							
w/o joined entry (= w/ implicit-only)		87.55	99.12	54.21	73.34	69.66	92.73
w/ explicit-only		88.48	98.61	52.22	70.73	66.36	90.79
Random-view		84.66	97.31	52.39	70.80	66.25	90.78
Random-query		85.99	97.05	54.97	73.15	68.73	92.51

Table 5: Adjusted recall (Slot R@k) across 3-Hop, 2-Hop, and Unified configurations. The best results are in **bold** and the second best are underlined. ▲ and ▼ indicate improvement and degradation over the strongest baseline within each metric column.

C Additional Evaluation

This section provides supplementary evaluations of retrieval performance and Join Path Discovery.

C.1 Retrieval formulation and unit-level evaluation

C.1.1 Adjusted recall

PEARL constructs an augmented retrieval corpus of three unit types: Original entries, Implicit sub-entries, and Explicit joined entries. Since joined entries may span multiple gold tables, they may inflate standard recall metrics. For example, given a gold set $\{A, B, C\}$, retrieving a joined unit $\{A, B, C\}$ at rank 1 results in all three tables being counted as correctly retrieved under standard R@1. While this reflects the advantage of joined entry in capturing multiple relevant tables jointly, it can also obscure the retrieval capacity of individual ranking positions.

To provide a more fine-grained evaluation, we additionally report Slot R@k, which re-ranks tables within the top-k retrieved units based on their individual sub-table similarity. Let $\mathcal{R}_k(q) = \{u_1, \ldots, u_k\}$ be the top-k retrieved corpus units sorted by similarity to query q, where each unit u is associated with a set of tables in its join path, $\text{tables}(u) = \{t_1, \ldots, t_h\}$. The candidate table set is defined as:

$$\mathcal{T}_k(q) = \bigcup_{u \in \mathcal{R}_k(q)} \text{tables}(u). \quad (13)$$

Each table $t \in \mathcal{T}_k(q)$ is re-scored by:

$$s(t, q) = \max_{\mathbf{e} \in \mathcal{P}(t)} \cos(\text{Enc}(q), \mathbf{e}), \quad (14)$$

where $\mathcal{P}(t)$ denotes the embeddings of all sub-table units associated with the table t across join paths. Tables in $\mathcal{T}_k(q)$ are then re-ranked according to $s(t, q)$. Slot R@k measures whether all gold tables appear within the top-k re-ranked tables.

RQ8. Do the performance gains of PEARL remain significant under Slot R@k evaluation? As summarized in Table 5, PEARL consistently maintains its advantage even under more fine-grained metrics. On SPIDER and MMQA, PEARL outperforms decomposition-based methods, achieving gains over the strongest baseline in the 3-hop configuration with relative improvements of 2.61% and 4.46% under Slot R@2, indicating that the benefits of PEARL scale with increasing hop complexity, consistent with Table 1. On BIRD, PEARL remains competitive with decomposition-based methods despite a modest performance drop, and we provide additional analysis in Appendix D.2. These results suggest that the gains primarily arise from noise reduction and inter-table relationship modeling enabled by PEARL's join-aware augmentation, rather than inflated coverage from joined entries.

C.1.2 Design rationale for max-pooling

PEARL does not enforce that retrieved tables originate from a single coherent join path. This is a deliberate design choice, as the objective of multi-hop table retrieval is to recover the complete set of gold tables within the top-k results rather than a single reasoning chain. Under this objective, max-pooling assigns each table the score of its best matching indexed view. Since a table may participate in multiple join paths, aggregating scores across all indexed views would dilute relevant evidence with unrelated contexts.

For the same reason, enforcing path coherence does not necessarily improve robustness to false joins. If an entire path were retrieved as a single unit, a single incorrect edge could compromise the whole chain. In contrast, PEARL evaluates augmented units independently and retains only the highest-scoring unit for each table, limiting the influence of spurious paths.

	SPIDER R@2/Slot	MMQA R@2/Slot	BIRD R@2/Slot
PEARL	91.25/**88.48**	58.84/**54.26**	73.64/**69.66**
Implicit joined	**92.29**/87.61	**61.57**/53.68	**74.53**/67.25

Table 6: Evaluation of the asymmetric design under the unified setting, where implicit paths are encoded as joined entries.

C.1.3 Design rationale for asymmetric encoding

The asymmetric encoding reflects the distinct retrieval signals of explicit and implicit paths. For explicit paths, the structural join itself is the primary information, making a joined representation effective (equation 7 in Section 3.1.3). In contrast, implicit paths lack deterministic relational joins, so forcing a unified representation would introduce arbitrary structure. PEARL therefore encodes each sub-table independently (equation 6 in Section 3.1.3), while the shared generated query provides the semantic context linking them.

To validate this design choice, we evaluate an alternative that encodes implicit paths as joined entries (Table 6). Although this variant improves standard R@2, it consistently underperforms under Slot R@2. This indicates that the apparent improvement is largely driven by a single joined entry covering multiple gold tables simultaneously, as discussed in Appendix C.1.1. Thus, the separate encoding strategy better aligns with the retrieval objective, while joined representations may still be preferable for applications that prioritize relational coherence.

C.2 Evaluation on Join Path Discovery and Query Coverage

Section 4.3 examined the evaluation of joined entries. Building on this analysis, we now shift our focus to the join paths themselves and investigate three remaining research questions.

RQ9. Does agreement with gold paths accurately reflect join path quality? Against the gold paths, Join Path Discovery achieves F1 scores of 0.61 on SPIDER, 0.38 on MMQA, and 0.39 on BIRD, with the corresponding precision and recall reported in Table 7. This modest agreement reflects the different objectives of gold paths and Join Path Discovery. A gold path captures the path required for an observed query, whereas Join Path Discovery identifies a broader set of paths without access to future queries. Under this objective, paths outside the gold set are not necessarily errors and may cover query intents absent from the benchmark.

(a) Gold-path agreement

Dataset	Precision	Recall	F1	PEARL		Oracle Join	
				R@2	R@5	R@2	R@5
SPIDER	0.58	0.65	0.61	**91.25**	**99.33**	88.09	97.24
MMQA	0.40	0.36	0.38	58.84	76.21	**61.68**	**85.51**
BIRD	0.58	0.30	0.39	73.64	93.36	**84.44**	**96.27**

(b) Sensitivity to join path quality

	SPIDER		MMQA		BIRD	
Method	R@2	R@5	R@2	R@5	R@2	R@5
PEARL	**91.25**	**99.33**	**58.84**	**76.21**	**73.64**	**93.36**
+50% noise	89.28	97.53	55.03	72.98	69.03	89.44
+100% noise	83.56	96.38	51.72	70.02	64.41	87.92
Stella-v5	85.46	97.42	53.10	71.72	66.71	91.36
Implicit Match (%)	62		78		69	

Table 7: Evaluation of join path discovery. (a) Gold-path agreement and retrieval performance using Join Path Discovery (PEARL) versus Oracle Join. (b) Sensitivity of retrieval performance to join path corruption and the contribution of implicit join paths.

Dataset	Median Lift	NN sim. (2h / 3h)	Dup. (%)
SPIDER	+0.200	0.713 / 0.733	0.4
MMQA	+0.216	0.704 / 0.718	0.3
BIRD	+0.173	0.682 / 0.688	0.1

Table 8: Coverage statistics of generated queries against the random-match. NN sim. denotes mean nearest similarity for 2-hop / 3-hop queries.

To examine whether these additional paths provide retrieval value, we compare PEARL with an oracle that indexes only gold paths. On SPIDER, PEARL outperforms the oracle, showing that gold paths are not necessarily retrieval optimal, and that low gold agreement does not imply low retrieval value. On MMQA and BIRD, the oracle performs better, indicating room for further gains rather than a limitation of the retrieval formulation itself (consistent with the analysis in Appendix D.2).

RQ10. How sensitive is retrieval performance to variations in join path quality? Since RQ4 shows that corpus expansion alone does not explain PEARL's gains, we test its dependence on join quality by replacing discovered paths with randomly constructed non-joinable paths while holding corpus size fixed. Performance decreases monotonically with the corruption rate and falls below the base encoder at 100% settings. At 50% corruption, PEARL still outperforms the base encoder on all three benchmarks, as the max-pooling strategy rarely selects low-similarity spurious units.

Table 7 further shows that this dependence on meaningful joins is also reflected in implicit paths. Their match share, the fraction of test queries whose nearest generated query originates from an implicit path, exceeds their prevalence in the generated query pool across benchmarks, particularly on BIRD. This suggests that the discovered joins align with real query intents rather than benefiting from volume. Moreover, a single Join Path Discovery pipeline captures transferable join structure across benchmarks without per-dataset tuning.

RQ11. Do generated queries capture real query intents without reproducing the original queries? RQ9 shows that low gold-path agreement does not imply low retrieval value, but it does not address whether the generated queries align with real information needs. We therefore conduct a query-level coverage analysis by measuring the cosine similarity between each test query and its nearest generated query. To provide a reference, we compare these similarities against a random-match, where each test query is paired with a randomly selected generated query.

As summarized in Table 8, coverage lies well above random, with the median exceeding the random-match by 0.173–0.216 across all benchmarks. The nearest-neighbor similarity is also higher for 3-hop than for 2-hop queries, consistent with the hop-wise retrieval gains observed in RQ2. Fewer than 0.4% of the nearest generated queries have a cosine similarity of at least 0.9. Since query generation relies only on table structure and sampled rows, this low duplication suggests that the retrieval gains arise from the augmented indexing units rather than from reproducing benchmark queries.

Setting	Model	SPIDER		MMQA		BIRD	
		R@2	R@5	R@2	R@5	R@2	R@5
3-Hop	Stella-v5	58.89	87.78	47.30	71.70	52.17	85.33
	GJAR	60.60	91.10	44.50	61.30	55.70	**88.50**
	JAR	63.30	95.60	44.40	63.50	56.20	84.80
	PEARL	**74.44** (▲ 17.60%)	**98.89** (▲ 3.44%)	**57.06** (▲ 20.63%)	**79.46** (▲ 10.82%)	**56.50** (▲ 0.53%)	87.67 (▼ 0.94%)
2-Hop	Stella-v5	76.97	96.06	55.76	72.63	64.67	92.29
	GJAR	86.30	95.80	56.40	63.90	**79.40**	93.50
	JAR	84.60	97.80	55.50	64.90	79.30	92.10
	PEARL	**87.02** (▲ 0.83%)	**99.49** (▲ 1.73%)	**58.92** (▲ 4.47%)	**75.66** (▲ 4.17%)	72.16 (▼ 9.12%)	**94.11** (▲ 0.65%)
Unified	Stella-v5	85.46	97.42	53.10	71.72	66.71	91.36
	GJAR	88.10	97.10	53.30	62.80	**74.70**	91.50
	JAR	87.70	98.10	52.60	64.10	**74.70**	90.30
	PEARL	**90.50** (▲ 2.72%)	**99.26** (▲ 1.18%)	**57.58** (▲ 8.03%)	**75.62** (▲ 5.44%)	72.55 (▼ 2.88%)	**93.24** (▲ 1.90%)

Table 9: Retrieval performance comparison with the same LLM backbone (GPT-4o-mini). The best results are in **bold** and the second best are underlined. The symbols ▲ and ▼ indicate the relative improvement and degradation over the strongest baseline within each metric column, respectively.

	SPIDER		MMQA		BIRD	
LLM Backbone	R@2	R@5	R@2	R@5	R@2	R@5
Llama 3.3 70B	**91.25**	**99.33**	**58.84**	**76.21**	**73.64**	**93.36**
GPT-4o mini	90.50	99.26	57.58	75.62	72.55	93.24
Qwen 2.5 7B	88.88	98.97	56.90	74.81	69.67	92.39

Table 10: Retrieval performance of PEARL with different LLM backbones.

D Further Analysis

In this section, we present additional experimental results to provide deeper insights into PEARL.

D.1 Effect of LLM Backbone

Comparison under an identical backbone Table 9 compares multi-hop table retrieval methods using the same LLM backbone (GPT-4o mini), isolating the effect of retrieval architecture from differences in model capability. Across most settings, PEARL consistently outperforms competing baselines. In particular, on SPIDER and MMQA, PEARL achieves the highest R@2 in the majority of evaluations, with gains up to +17.6% and +20.63% on 3-hop SPIDER and MMQA, respectively. On BIRD, PEARL attains the best performance in R@5. These results indicate that the performance gain of PEARL primarily arises from its retrieval design rather than the strength of the underlying backbone model.

Robustness across LLM Backbones Although PEARL uses a 70B backbone in the main experiments, we additionally evaluate Qwen 2.5 7B Instruct and GPT-4o mini in a zero-shot setting to assess whether the method remains effective with lightweight LLMs for offline indexing.

As shown in Table 10, PEARL is largely insensitive to backbone capacity. Under the unified setting, the smaller models (GPT-4o mini and Qwen 2.5 7B) achieve retrieval performance comparable to the larger Llama 3.3 70B model. This robustness makes PEARL practical when computational resources are limited, since the LLM is invoked only once during offline indexing, allowing smaller backbones to reduce the indexing cost without sacrificing retrieval effectiveness. Future improvements in lightweight LLMs can therefore be adopted directly to further reduce indexing cost without modifying the method. Together with the controlled comparison above, these results suggest that PEARL's gains stem from its join-aware indexing strategy rather than backbone capacity.

D.2 Analysis of BIRD dataset

PEARL trails GJAR and JAR on BIRD 2-hop R@2, while outperforming both on SPIDER and MMQA. To better understand this gap, we analyze the characteristics of the BIRD dataset.

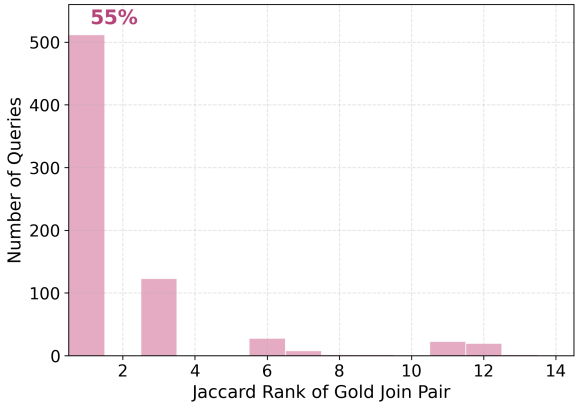

Figure 6: Jaccard rank distribution of gold join pairs among intra-DB table pairs for BIRD 2-hop queries.

	Tables	Impl.	Expl.	Expand	Index (MB)
SPIDER	81	73	40	2.40×	0.79
MMQA	555	496	277	2.39×	5.44
BIRD	75	68	37	2.40×	0.74

Table 11: Storage overhead introduced by PEARL. "Impl." and "Expl." denote the numbers of additional implicit and explicit indexing units, respectively.

As shown in Figure 6, 55% of gold join pairs already rank first based on Jaccard similarity among all intra-database table pairs. This is partly due to the small scale of the BIRD development set, which contains only 75 tables, making offline Jaccard pre-computation over all table pairs trivial and yielding highly reliable statistical similarity for identifying gold join pairs, a setting where JAR and Greedy-JAR naturally excel. In contrast, such strong statistical priors are less likely to hold in large-scale settings, where value distributions are more diverse and similarity-based signals alone are insufficient to reliably identify correct joins.

Given the gap between real-world data and the BIRD dataset, these results can be interpreted from a different perspective. PEARL still improves over the Stella-v5 baseline in slot-level recall on BIRD, indicating that join-aware offline indexing remains effective even under strong statistical priors.

These observations suggest that BIRD may underestimate the benefit of our approach compared to more realistic large-scale settings. This is further supported by stronger performance on MMQA, which better reflects complex multi-hop retrieval scenarios than BIRD and SPIDER (Wu et al., 2025), including consistent gains under adjusted recall and over decomposition-based baselines. We leave a more detailed investigation of large-scale database settings to future work.

MMQA	Paths	Emb.	R@2	R@5
Full re-indexing	554	13,478	58.84	76.21
Incremental	244†	2,730	58.54	76.15

Table 12: Indexing workload and retrieval performance under a 20% corpus expansion on MMQA. †310 reused paths yield 554 indexed paths in total.

D.3 Storage and Index Overhead

While Table 3 reports the overall indexing efficiency, this section provides a more detailed analysis of the resulting storage and indexing overhead.

Table 11 additionally reports the storage and unit-count breakdown at $K = \lfloor |\mathcal{C}|/2 \rfloor$. Because K scales with $|\mathcal{C}|$, the corpus expansion factor remains nearly constant at approximately 2.4 times across all three benchmarks. Even with this expansion, the largest benchmark, MMQA, requires only a 5.44 MB index, roughly 3 MB larger than whole-table indexing. Moreover, the additional indexing units have little impact on online retrieval efficiency, with ANN search remaining below 32 ms even on MMQA.

This sub-linear scaling persists as the corpus budget increases. Raising K from $0.25|\mathcal{C}|$ to $1.00|\mathcal{C}|$ increases online latency by less than a factor of two across all three benchmarks (Table 3). Thus, the overhead of corpus augmentation is incurred primarily in storage, which scales predictably with $|\mathcal{C}|$, rather than in online latency, consistent with PEARL's front-loading design. When storage is constrained, this overhead can be further reduced by lowering the corpus budget K, as shown in RQ6 (Section 4.4).

D.4 Incremental Indexing under Corpus Expansion

PEARL front-loads LLM reasoning into an offline index, raising the question of how the index can be maintained as the corpus evolves. Since PEARL constructs join paths from pairwise join edges, updates can be localized at the edge level. Adding a table requires computing candidate join edges only for table pairs involving the new table, while existing edges remain unchanged.

Corpus units are then generated only for paths containing a new edge, with previously generated queries and embeddings reused. Similarly, removing a table only invalidates its incident edges and the corpus units associated with paths containing those edges. Thus, the update cost depends primarily on the affected edges and paths rather than requiring the entire corpus to be re-indexed.

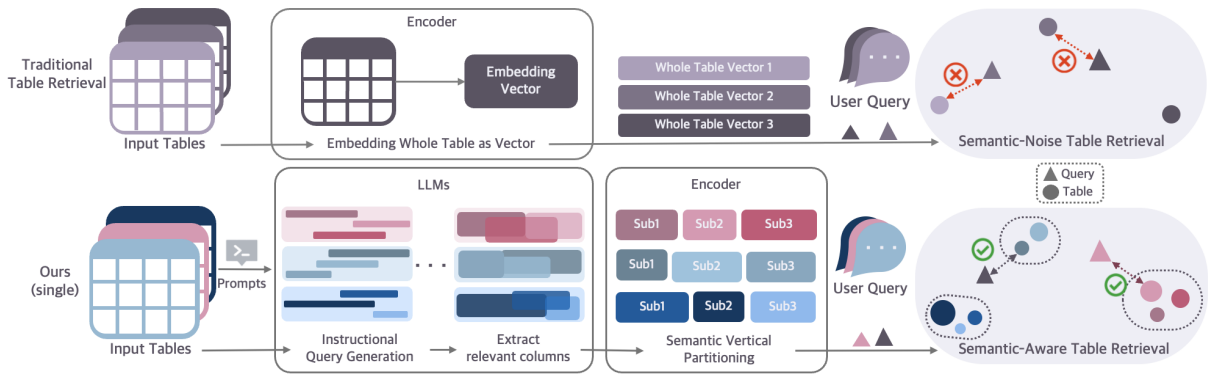

Figure 7: Overall procedure of PEARL for single-hop setting.

We empirically evaluate the addition case on MMQA by splitting its 555 tables into an initial set of 444 tables (80%) and an update set of 111 tables (20%). Full re-indexing rebuilds the index from scratch over all 555 tables, rediscovering all join edges and regenerating all queries. In contrast, incremental indexing starts from the index built on the initial 444 tables and performs Join Path Discovery only for table pairs involving at least one newly added table. All previously generated queries and embeddings are reused unchanged. Both settings are evaluated on the same test set. As shown in Table 12, incremental indexing generates queries for 244 paths while reusing 310 paths from the existing index. This requires only 44% of the LLM query generation cost of full re-indexing while achieving nearly the same retrieval performance, indicating that the statistics inherited from the existing index remain reliable under a 20% corpus expansion. PEARL can therefore accommodate periodic corpus expansion without rebuilding the entire index.

D.5 Toward Adaptive Offline-Online Indexing

PEARL generates all reasoning units offline before deployment, prioritizing low online latency and eliminating LLM inference from the retrieval pipeline. However, our results suggest that fully offline indexing may not always be necessary. RQ6 (Section 4.3) shows that retrieval gains are concentrated in a small subset of high-scoring join paths. Figure 3 further shows that offline indexing becomes advantageous only after its preprocessing cost of offline indexing is amortized, reaching the break-even point after 22–35% of the query workload. Together, these observations suggest an adaptive indexing strategy that precomputes only the join paths whose expected retrieval benefit outweighs their indexing cost while resolving the remaining paths online when needed.

E Single-Hop Retrieval Setting

PEARL also generalizes to single-hop retrieval scenarios. The overall pipeline remains identical to the main framework(Figure 2), except that it omits multi-table Join Path Discovery. Instead, PEARL is inspired by Random-view variants, and augments individual tables by generating instructional queries that capture potential user intents, incorporating them into semantically enriched table representations.

Figure 7 illustrates the overview of PEARL in single-hop setting. In the offline indexing stage, PEARL first decomposes input tables into sub-tables by generating instructional queries that capture potential user intents. These queries are then incorporated into the sub-tables to construct a semantically augmented embedding corpus, offering diverse views of the original tables. In the online retrieval stage, PEARL calculates the relevance score between a given query and a table by pooling semantically aligned views of the table.

E.1 Datasets and Evaluation Metrics

We evaluate PEARL on the single-hop table benchmark FeTaQA (Nan et al., 2022), which contains complex schemas. Since our method does not require training, we treat the entire dataset as a unified retrieval pool without introducing any supervision leakage. This setup yields a more practical search space that better reflects real-world retrieval scenarios. We report standard Recall@k.

E.2 Baselines and Implementation Details

We compare PEARL with diverse baselines, categorized into three paradigms: (1) *Sparse retrievers* BM25, TF-IDF, and SPLADE (Formal et al., 2021); (2) *Structure-aware dense retrievers* TAPAS (Herzig et al., 2020) and DTR (Herzig

Model	FeTaQA			FeTaQA (≥ 11 Cols)		
	R@1	R@5	R@10	R@1	R@5	R@10
BM25	20.33	31.82	36.43	8.77	15.80	19.88
TF-IDF	8.32	17.32	22.07	10.30	18.86	23.24
SPLADE	29.21	46.59	53.75	14.78	33.03	41.69
DTR	18.03	35.56	44.01	13.25	26.81	36.60
TAPAS	9.07	19.00	25.14	6.63	17.33	23.34
Stella-v5 (baseline)	35.15	53.85	61.75	20.18	38.63	48.32
PEARL	**40.69**	**59.89**	**67.03**	**27.83**	**46.69**	**55.35**
	(▲ 15.8%)	(▲ 11.2%)	(▲ 8.6%)	(▲ 37.9%)	(▲ 20.9%)	(▲ 14.5%)
Ablation of PEARL						
w/ subtable only	33.37	51.67	58.44	18.96	36.90	46.08
w/ query only	34.38	52.73	60.04	22.94	41.18	48.52
w/ query & header	35.01	53.46	60.61	23.24	41.90	49.95

Table 13: Overall performance results on single-hop setting. The best results are in **bold** and the second best are underlined. The symbol ▲ indicates the relative improvement over the Stella-v5 baseline.

et al., 2021), which explicitly model tabular structures and their underlying relational logic; and (3) *Text-based dense retrievers* Stella-v5 (Zhang et al., 2024), a high-performance pretrained encoder that has demonstrated superior table-to-text alignment on TARGET benchmark (Ji et al., 2025). For architectural consistency, PEARL adopts Stella-v5 as the backbone encoder and Llama 3.3 70B for query generation. All baselines and PEARL are evaluated under a zero-shot setting with 512 tokens.

E.3 Experimental Results

With Table 13, we answer the following research questions to analyze the effectiveness of PEARL on single-hop retrieval quality.

RQ1. Does vertical partitioning improve single-hop retrieval accuracy?

PEARL consistently outperforms all baselines in R@1 and R@5 on FeTaQA benchmark. To further analyze these gains, we evaluate the FeTaQA(≥ 11 Cols) subsets, where PEARL achieves a 20.9% (8.1%p) gain in R@5 over the strongest baseline. The larger improvement on wide tables suggests that vertical partitioning effectively mitigates information dilution by pruning irrelevant columns, particularly in tables with many attributes that introduce substantial retrieval noise. By identifying query-relevant contexts in advance, PEARL constructs semantically salient sub-table fragments that better align with downstream queries.

RQ2. Are both sub-tables and augmented queries effective for retrieval? We conduct an ablation study of PEARL to evaluate the contribution of each component in representing table views. PEARL consistently outperforms variants utilizing only sub-tables, augmented queries, or query-header combinations.

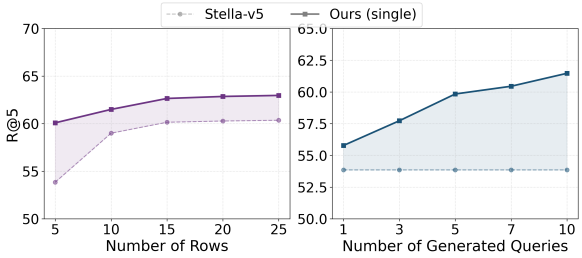

Figure 8: Retrieval performance (R@5) over different hyperparameters of PEARL on the FeTaQA dataset.

Notably, the performance gain by utilizing both query and sub-tables persists across wide-table and multi-hop scenarios. These results demonstrate the synergy between vertically partitioned tables and query-conditioned encoding in effectively filtering irrelevant semantic noise.

RQ3. How do the number of queries N affect the effectiveness of PEARL? To investigate the impact of key hyperparameters in PEARL in single-hop setting, we perform a sensitivity analysis on the number of generated queries. As shown in Figure 8, R@5 with $N = 1$ already outperforms the baseline, with performance consistently improving as N increases. This suggests that the diversity of instructional queries is positively correlated with the coverage of potential user intents, highlighting an opportunity for further scaling the benefit of PEARL. Regarding k_{avg}, retrieval performance saturates between 3 and 5, validating our guidance for setting the column subset size.

F Prompt Templates

This section presents prompt templates for instructional query generation in PEARL. All prompts use the LLM configuration described in Section 3.1.2.

Implicit Multi-Hop Query Prompt Template For implicit paths, we design separate prompts for 2-hop and 3-hop settings. The 2-hop prompt enforces a strict two-table setting with value-based entity alignment, while the 3-hop prompt treats the intermediate table as a value-based bridge connecting the end-point tables and adapts its description for semantic and mixed chains. Both prompts enforce multi-hop reasoning and require justification of each constituent table's necessity.

Explicit Multi-Hop Query Generation Prompt Template For explicit join paths, the prompt enforces alignment between semantic transitions and underlying Foreign-key relationships across 2-hop and 3-hop joins. Each generated query must require all tables in the chain to answer, and intermediate tables must be expressed as natural language semantic relations.

Single-Hop Instructional Query Generation Prompt Template To extend PEARL to single-hop retrieval, we design a prompt that generates diverse instructional queries over individual tables. As described in Section E, the queries cover a broad range of reasoning patterns, producing enriched table representations.

Explicit Join Prompt

[SYSTEM PROMPT]
You are an expert in "Schema-Aware Multi-hop Database Question Generation".
Your goal is to generate ONE natural but complex question that acts as a unique signature for the explicit structural chain connecting these {n_tables} tables.

CONTEXT: EXPLICIT JOIN SEMANTICS
These {n_tables} tables are strictly connected by explicit foreign key (FK) / primary key (PK) relationships.

- **JOIN concept bridge:** The INPUT block spells out this structural chain; your question MUST reflect that same bridge in natural language (each hop is a concrete semantic link, not vague "relatedness").
- **Stepwise necessity:** The answer MUST require following EVERY join link in order—one hop at a time. Skipping any table breaks the query's meaning or makes it unanswerable.
- **Traversal direction:** You MAY anchor at either end (Table 1 or Table {n_tables}). The solver may reason forward along the chain or reverse along the same links (as in the INPUT); do not invent shortcuts or alternate join paths.

STRATEGY: JOIN-AWARE ANCHORING & TRAVERSAL

1. **Semantic Translation:** Understand why two tables are joined based on their column names (e.g., artist.id = album.artist_id → "Artist released Album"). Translate these structural links into natural language relations (verbs, ownership).
2. **Anchor:** Start at one end of the chain (Table 1 or Table {n_tables}) using specific proper noun(s) from the provided data rows.
3. **Traverse via Relations:** Force the solver to follow the semantic path (e.g., "Find the manager (Table 3) of the department (Table 2) where employee X (Table 1) works").
4. **Target:** Retrieve the specific attribute at the far end of the chain.
5. **Discriminative wording:** Do NOT use pronouns (he, she, it, they); use explicit proper nouns from the rows.

RULES

1. Output ONLY ONE valid JSON object inside the schema defined below. Output ONLY the JSON object. No markdown code fences. No extra text.
2. The question MUST:
   - Require ALL {n_tables} tables to answer (skipping any single table makes it unanswerable or semantically completely different).
   - NOT mention the raw database column names explicitly (e.g., do not say "where artist_id is 5"). Use natural language instead.
   - Include specific proper nouns (names, places, IDs, events) from the data rows as anchors.
   - NOT use generic phrasing like "connected to" or "in this table". Use precise verbs representing the table's schema role.
3. {mid_rule}
4. **Column Selection:** Target {target_c} columns total across all {n_tables} tables (Max {max_c}).

[EXPECTED JSON FORMAT]

```
{
  "questions": [
    {
      "question": "<Natural language question reflecting the explicit JOIN relationships>",
      "thought": "{thought_tail}",
{col_lines}
    }
  ]
}
```

Figure 9: Prompt template used for explicit multi-hop query generation over structural FK/PK join chains.

Implicit 2-hop Join Prompt

[SYSTEM PROMPT]
You are an expert in "Discriminative Multi-hop Retrieval".
Your goal is to generate ONE complex question that acts as a unique signature for the semantic connection between these TWO tables.

CONTEXT: IMPLICIT (SEMANTIC) JOIN
These two tables are NOT connected by an explicit foreign key. Instead, they share overlapping real-world entities through semantically related columns. Think of it as: "Table 1 tells you one thing about entity X; Table 2 tells you another thing about entity X."

STRATEGY: CROSS-PERSPECTIVE ENTITY ANCHORING
1. **Find the Shared Entity:** Identify the entity appearing in BOTH tables under different column names.
2. **Use Table 1 to Identify:** Pick a unique attribute from Table 1 to pinpoint an instance.
3. **Retrieve from Table 2:** Ask for an attribute that only exists in Table 2.
4. **Make It Discriminative:** Include enough specific proper nouns so only this pair can answer it.

RULES
1. Output ONLY ONE valid JSON object:

```
{
  "questions": [
    {
      "question": "<Discriminative implicit multi-hop question>",
      "thought": "Step-by-step reasoning trace.",
      "relevant_columns_t1": ["col1"],
      "relevant_columns_t2": ["col2"]
    }
  ]
}
```

2. The question MUST:
   - Require BOTH tables to answer.
   - NOT mention shared column names explicitly.
   - Include specific proper nouns from data rows.
   - NOT use pronouns or generic phrases ("in this table", "between the tables").
   - Read like a simple factual lookup; the multi-table requirement must be inferrable only from data.
3. Output ONLY the JSON object. No markdown code fences. No extra text.

ABSOLUTE CONSTRAINT: EXACTLY 2 TABLES
- Question MUST be answerable from EXACTLY these 2 tables.
- **Forbidden thought tokens:** "step 3", "table 3", "third table", "next look up in another".
- If a third table is required, output:
  {"questions":[{"question":null,"thought":"requires more than 2 tables",...}]}

Figure 10: Prompt template for 2-hop implicit query generation over value-based semantic joins.

Implicit 3-hop Chain Prompt

[SYSTEM PROMPT]
You are an expert in "Semantic Multi-hop Table Retrieval".
Your goal is to generate ONE question that acts as a unique signature for this 3-table semantic chain.

CONTEXT: IMPLICIT SEMANTIC CHAIN (3 TABLES)
These 3 tables are connected by VALUE OVERLAP, NOT by FK/PK column-name patterns. {bridge_note}
Table 2 is the bridge — it holds the entity that links Table 1 and Table 3 through shared values.
Skipping Table 2 makes the question unanswerable or semantically different.

STRATEGY: CHAINED ENTITY ANCHORING

1. **Identify the bridge:** Table 2 connects Table 1 and Table 3 via shared values.
2. **Anchor:** Use a specific proper noun from Table 1 (or Table 3) as the starting point.
3. **Traverse:** Match through Table 2 first, then retrieve from the far end.
4. **Discriminative:** Include specific proper nouns so only this 3-table chain can answer it.

RULES

1. Output ONLY ONE valid JSON object:

```
{
  "questions": [
    {
      "question": "<Discriminative 3-table semantic chain question>",
      "thought": "1. Semantic Bridge: ... 2. Data Trace: ...
                  3. Indispensability: Table 2 ([name]) is
                  indispensable because [...]",
      "relevant_columns_t1": ["col1"],
      "relevant_columns_t2": ["col1"],
      "relevant_columns_t3": ["col1"]
    }
  ]
}
```

2. The question MUST:
   - Require ALL 3 tables to answer.
   - NOT mention shared column names.
   - Include specific proper nouns from data rows.
   - NOT use pronouns.
   - Read like a natural factual lookup; the 3-table requirement must not be hinted at.
3. Output ONLY the JSON object. No markdown code fences. No extra text.

Figure 11: Prompt template for 3-hop implicit query generation over semantic chain joins.

Single-hop Prompt

[SYSTEM PROMPT]
You are an expert in "Discriminative Table Retrieval".
Your goal is to generate 5 distinct questions that act as a **unique signature** for this table, ensuring it can be found among thousands of other similar tables.

STRATEGY: DISCRIMINATIVE ANCHORING

1. **Identify Primary Entities:** Find the main subjects (e.g., specific player names, specific city names, specific years, unique award titles).
2. **Target Specific Rows:** Don't just summarize. Create questions that anchor to the most 'unique' or 'extreme' rows in the table.
3. **Keyword Density:** Ensure every generated question contains the **exact proper nouns** (Names, Titles, Dates) found in the table.

RULES

1. Output ONLY ONE valid JSON object following the exact schema provided in the format block below.
2. Generate **exactly 5** questions with different levels of granularity:
   - **Q1 (Entity-Focus):** Targets the main person/subject and their most notable record or identity.
   - **Q2 (Temporal-Focus):** Links a specific year/season/date to a unique result in that period.
   - **Q3 (Comparison/Superlative):** Focuses on "highest", "lowest", "first", or "total" values while mentioning the subject name.
   - **Q4 (Multi-Column Integration):** Requires combining data from several different columns to form a complete answer.
   - **Q5 (High-Complexity Reasoning):** A discriminative question that links multiple attributes to uniquely identify a specific data point.
3. **Question Constraints:** NEVER use generic phrases like "in this table" or "the provided data". ALWAYS include specific subject names. BE SPECIFIC. Ensure valid JSON format and escape backslashes properly (e.g., use \\\\ inside strings).
4. **Column Selection Guidelines:** Select the minimal set of relevant columns needed to answer the question to maintain optimal signal density, avoiding unnecessary information noise.
5. Output ONLY the JSON object. No markdown code fences, no extra text.

[EXPECTED OUTPUT FORMAT & USER INPUT]

```
{
  "feta_id": {table_id},
  "table_name": "{table_name}",
  "questions": [
    {
       "question": "<Discriminative, self-contained question containing specific proper nouns>",
        "thought": "<Step-by-step reasoning: Explain the logic of column composition>",
        "relevant_columns": ["col1", "col2"]
    }
  ]
}
```

INPUT TABLE DATA
{table_str}

Generate the JSON object with 5 discriminative questions as specified.

Figure 12: Prompt template used for single-hop query generation.