Retrieve Only Relevant Tables Whether Few or Many: Adaptive Table Retrieval Method

Taehee Kim[1,2]*, **Seungbin Yang**[1]*, **Jihwan Kim**[1], **Jaegul Choo**[1]
[1]KAIST AI, [2]Letsur
{taeheekim, sby99, jihvvan.kim, jchoo}@kaist.ac.kr

Abstract

Retrieving relevant tables from extensive databases for a given natural language query is essential for accurately answering questions in tasks such as text-to-SQL. Existing table retrieval approaches select a pre-determined set of k tables with the highest similarity to the query. However, the number of required tables varies across queries and cannot be known in advance. Enforcing a fixed number of retrieved tables regardless of the query may either retrieve an undersized set, failing to obtain all necessary evidence, or retrieve an oversized pool, including irrelevant tables. To address this issue, we propose an adaptive table retrieval method that adjusts the number of tables retrieved according to the requirements of each query. Specifically, we utilize an adaptive thresholding mechanism to selectively retrieve tables and integrate a sliding-window reranking algorithm to efficiently process a large table corpus. Extensive experiments on Spider, BIRD, and Spider 2.0 demonstrate that our method effectively addresses the limitations of the existing retrieval strategy, improving performance in retrieval and downstream tasks. Our code and data are available at link.

1 Introduction

Advances in large language models (LLMs) have led to substantial gains in tasks demanding structured reasoning over tabular data (Gao et al., 2023; Xie et al., 2024; Yang et al., 2024). These improvements are critical to real-world applications such as text-to-SQL and open-domain question answering, where leveraging structured data is essential (Zhong et al., 2017; Yu et al., 2018; Herzig et al., 2021). Retrieval-augmented generation (RAG) approaches address this need, first retrieving tables relevant to a query and then generating an answer conditioned on the retrieved tables (Lewis et al., 2020; Pan et al., 2022; Kothyari et al., 2023; Kang et al., 2024; Kong et al., 2024).

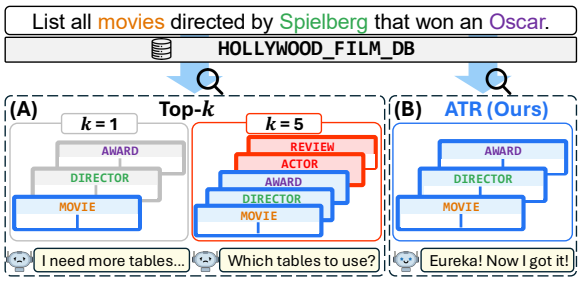

Figure 1: Rather than rely on a rigid fixed k retrieval strategy, ATR retrieves only relevant tables. Gray indicates tables required by the query but not retrieved, red denotes irrelevant tables, and blue highlights retrieved relevant tables.

Existing table retrieval methods compute query-table similarity and select a pre-determined k tables with the highest similarity (Chen et al., 2024b; Wu et al., 2025; Zhang et al., 2025). However, this top-k retrieval strategy overlooks the fact that the number of tables required by each query is unknown in advance and can vary significantly depending on its complexity. Even within the same database, the ground-truth tables can range from one to several hundred depending on the query. For example, the problem is evident in Spider 2.0 (Lei et al., 2025), a realistic enterprise-level text-to-SQL benchmark where the number of ground-truth tables for each query ranges from 1 to 366.

Because of the uncertainty, a fixed k retrieval method can miss necessary tables or retrieve too many tables to answer, depending on the value of k. As illustrated in Figure 1-(A), answering the query *"List all movies directed by Spielberg that won an Oscar."* requires three tables: `MOVIE`, `DIRECTOR`, and `AWARD`. With $k = 1$, the retriever misses required tables, whereas $k = 5$ inevitably retrieves two irrelevant ones. Thus, a small k sacrifices recall and a large k inflates latency and injects noise, de-

*Equal contribution

Findings of the Association for Computational Linguistics: ACL 2026, pages 13031–13054

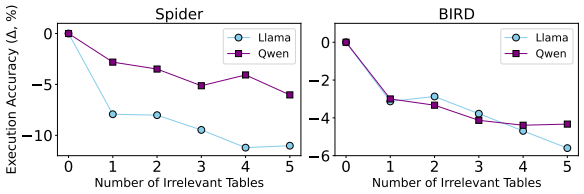

Figure 2: Retrieving irrelevant tables introduces noise, degrading performance on the text-to-SQL task. Execution accuracy consistently decreases as more irrelevant tables are added to the ground-truth tables.

grading downstream performance (Kothyari et al., 2023). Figure 2 quantifies how performance decreases as irrelevant tables are added.

To overcome this uncertainty in deciding how many tables a query needs, we introduce **A**daptive **T**able **R**etrieval (ATR). As depicted in Figure 1-(B), ATR identifies the exact set of tables, without relying on a fixed k. ATR uses an adaptive thresholding mechanism to selectively retrieve tables whose logits surpass the query-specific threshold. A sliding-window reranking algorithm is integrated into ATR to efficiently process large table corpora. Furthermore, ATR leverages a relevance calibration and semantic grouping loss to effectively model query-table and table-table relevance.

We evaluate ATR on Spider (Yu et al., 2018), BIRD (Li et al., 2023), and Spider 2.0 (Lei et al., 2025). ATR outperforms top-k baselines across all datasets, retrieving fewer irrelevant tables and more essential ones, and improving both retrieval metrics and downstream text-to-SQL execution accuracy. These improvements hold regardless of the number of tables required, demonstrating robustness.

Our contributions can be summarized as follows:

- We demonstrate that fixed k table retrieval overlooks the variation in required tables, resulting in the over-selection of irrelevant tables or the omission of essential ones.
- We propose ATR, an adaptive table retrieval method that leverages query-dependent thresholding with relevance calibration and semantic grouping losses to precisely retrieve necessary tables, combined with a sliding-window reranking to ensure scalable and efficient processing across extensive table corpora.
- Experiments on Spider, BIRD, and Spider 2.0 show that ATR consistently outperforms strong top-k baselines by precisely retrieving the necessary tables required to answer each query, thereby improving both efficiency and effectiveness in downstream text-to-SQL task.

2 Related Work

Table Retrieval Table retrieval is the task of selecting the subset of tables that provide evidence for a natural-language query from databases (Herzig et al., 2021; Wang et al., 2021; Dong et al., 2022; Balaka et al., 2025). Chen et al. (2024b) introduce methods capturing inter-table relationships and Li et al. (2025b) propose dynamic weighting of multiple fields based on query. Li et al. (2025a) propose a field-aware hybrid matching framework integrating sparse and dense representations. More recently, studies have explored LLM-based retrieval methods, leveraging LLMs to retrieve relevant tables (Kothyari et al., 2023; Zhang et al., 2025; Wu et al., 2025). Despite these advances, all of the above methods still rely on a fixed k retrieval strategy, failing to adapt to the varying table requirements across queries.

Adaptive Retrieval Strategy Recent RAG research has shown growing interest in adaptive retrieval methods, which adjust the size of text chunks based on query (Jiang et al., 2023; Asai et al., 2023; Jeong et al., 2024). These approaches assess query complexity and selectively increase the retrieval budget accordingly. However, such methods are designed exclusively for text and do not consider structured tabular data. Moreover, they involve iterative interactions with an LLM, increasing inference cost. To overcome these limitations, ATR adopts an adaptive thresholding to retrieve a query-dependent number of tables without iterative generator interactions and explicitly learns table representations optimized for tabular data.

Text-to-SQL Text-to-SQL is the task of generating SQL queries from natural language questions, enabling effective access to structured databases (Zhong et al., 2017; Yu et al., 2018; Hong et al., 2024). Recent studies in text-to-SQL adopt retrieval-augmented approaches that integrate table retrieval with text-to-SQL generation to handle large-scale database scenarios (Kothyari et al., 2023; Kong et al., 2024; Chen et al., 2024b). Additionally, large-scale datasets recently introduced by Chen et al. (2024a) and Lei et al. (2025) focus on enterprise-level text-to-SQL tasks. In line with these developments, our work contributes an adaptive retrieval framework that efficiently scales to

large table corpora, significantly enhancing text-to-SQL performance and efficiency in resource-intensive contexts.

3 Problem Definition

We formally define the table retrieval task as follows. Given a query q and a table corpus $\mathcal{C} = \{t_i\}_{i=1}^{N}$, where each table t_i contains structured information, the objective of the table retrieval task is to find a subset of tables $\mathcal{C}_q \subseteq \mathcal{C}$ that cover the information required to accurately answer the query.

A retrieval function f ranks tables in corpus $\mathcal{C}$ by descending order of the relevance scores $s(q, t_i)$, computed based on the query-table relevance. The top-k tables $\hat{\mathcal{C}}_q$ are selected according to $s(q, t_i)$.

$$\hat{\mathcal{C}}_q = \{ t_q^{(i)} \}_{i=1}^{k} = f(q, \mathcal{C}),$$
$$\text{where} \quad s\big(q, t_q^{(n)}\big) > s\big(q, t_q^{(m)}\big) \quad \forall\, n < m$$

4 Adaptive Table Retrieval

While standard top-k retrieval always returns a fixed k of tables, ATR adaptively selects a query-specific number of tables k_q required for each query:

$$\hat{\mathcal{C}}_q = \{ t_q^{(i)} \}_{i=1}^{k_q} = \mathrm{ATR}(q, \mathcal{C})$$

ATR is a transformer encoder that takes as input a query along with tables. It encodes tables into hidden representations conditioned on query relevance. The final retrieved set $\hat{\mathcal{C}}_q$ and the number of retrieved tables k_q are adaptively determined by comparing the logit of each table's hidden states with a query-dependent adaptive threshold. To effectively capture both query-table and inter-table relevance, we use two complementary objectives: the *relevance calibration* loss that sharpens query-table alignment, and the *semantic grouping* loss that pulls the embeddings of joinable tables closer. For efficient inference over large table corpora, we propose a sliding-window reranking method that refines the ranking without exhaustively scoring every table at once. Figure 3 illustrates the operational mechanism of ATR, detailing the adaptive retrieval process and its training objectives.

Adaptive Thresholding ATR is trained to separate relevant tables from irrelevant tables for each query through an adaptive thresholding mechanism inspired by Zhou et al. (2021). We prepend a *threshold token* T_{th} to the input sequence to represent the adaptive boundary between relevant and irrelevant tables. For each table t_i consisting of the database name, table name, and column names, we prepend a *table token* T_i to serve as its representative embedding. The resulting input sequence is structured as: $[T_{th}; q; T_1; t_1; \ldots; T_n; t_n]$.

ATR computes logits $logit_{T_{th}}$ and $logit_{T_i}$ from the hidden states of special tokens T_{th} and T_i, respectively. While training, we enforce that $logit_{T_i}$ is bigger than $logit_{T_{th}}$ when the table is relevant, and is smaller than $logit_{T_{th}}$ otherwise. The loss for adaptive thresholding is as follows:

$$L_1 = -\sum_{r \in \mathcal{T}^+} \log \frac{\exp(\mathrm{logit}_r)}{\sum_{r' \in \mathcal{T}^+ \cup \{T_{th}\}} \exp(\mathrm{logit}_{r'})}$$

$$L_2 = -\log \frac{\exp(\mathrm{logit}_{T_{th}})}{\sum_{r' \in \mathcal{T}^- \cup \{T_{th}\}} \exp(\mathrm{logit}_{r'})}$$

$$L_{AT} = \alpha L_1 + \beta L_2$$

where $\mathcal{T}^+$ denotes the set of relevant table tokens, and $\mathcal{T}^-$ denotes the set of irrelevant table tokens.

L_1 raises logits of query-relevant tables above the threshold logit $logit_{T_{th}}$, creating a clear margin from irrelevant tables. Since a query can have multiple relevant tables, we compute a binary cross-entropy loss for each relevant table and sum the results. In contrast, L_2 suppresses the logits of irrelevant tables below $logit_{T_{th}}$ by treating the threshold token as the correct class. The threshold logit thus becomes a query-dependent decision boundary that distinguishes between relevant and irrelevant tables. The hyper-parameters α and β weight the relative contributions of the two loss terms.

Relevance Calibration Adaptive thresholding assigns a query-specific cut-off by computing a logit for every table and comparing it with the logit of the threshold token. To sharpen the distinction between relevant tables $\mathcal{T}^+$ and irrelevant tables $\mathcal{T}^-$, we maximize the logit gap between $\mathcal{T}^+$ and $\mathcal{T}^-$ using a binary cross-entropy (BCE) loss:

$$L_{RC} = -\frac{1}{|\mathcal{T}^+ \cup \mathcal{T}^-|} \left(\sum_{r \in \mathcal{T}^+} \log\left(\sigma(\mathrm{logit}_r)\right) + \sum_{r \in \mathcal{T}^-} \log\left(1 - \sigma(\mathrm{logit}_r)\right) \right)$$

where σ denotes the sigmoid function.

This relevance calibration loss aligns each query with its relevant tables, giving ATR a signal to dis-

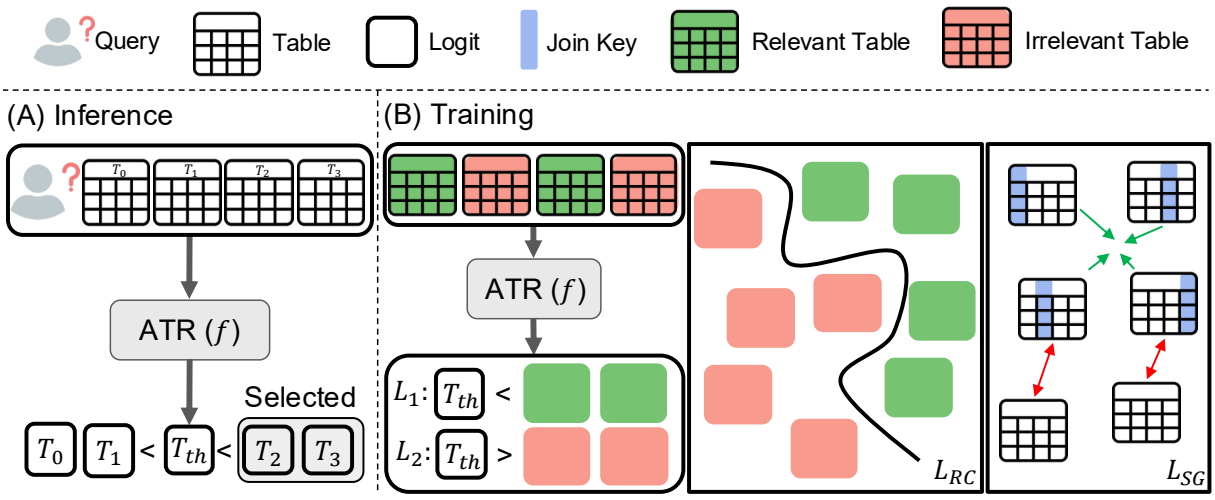

Figure 3: Overview of the ATR framework. (A) Inference: ATR takes a query and candidate tables as input to compute relevance logits. It dynamically selects the subset of tables (e.g., T_2, T_3) whose logits exceed that of the adaptive threshold token (T_{th}), thereby automatically determining the specific number of tables needed for the query. (B) Training: ATR is optimized using three complementary objectives: 1) Adaptive Thresholding (L_{AT}) trains the threshold token to serve as a decision boundary, enforcing relevant tables to score higher (L_1) and irrelevant ones lower (L_2) than T_{th}; 2) Relevance Calibration (L_{RC}) maximizes the logit margin between relevant and irrelevant tables; and 3) Semantic Grouping (L_{SG}) adopts contrastive learning to pull embeddings of joinable tables that are sharing join keys closer together while pushing non-joinable ones apart.

tinguish them from irrelevant ones and improving its discriminative capability.

Semantic Grouping Relationships between tables, especially whether tables can be joined, are critical for accurate multi-table retrieval (Chen et al., 2024b; Wu et al., 2025). To consider these dependencies between tables, we add a contrastive learning objective (Hadsell et al., 2006; Chen et al., 2020) that pulls embeddings of *joinable* tables closer together while separating embeddings of *non-joinable* tables by a fixed margin. Let e_i be the embedding of table t_i and let g_i denote its label, where tables sharing the same label are joinable; the semantic grouping loss L_{SG} is defined as:

$$L_{SG} = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \Big[\mathbb{I}(g_i = g_j) \, \|e_i - e_j\|_2^2 + \mathbb{I}(g_i \neq g_j) \, \max\big(0,\, m - \|e_i - e_j\|_2\big)^2 \Big]$$

where $\mathcal{P}$ represents all unique pairs of $\mathcal{C}$, and m denotes the margin hyper-parameter. L_{SG} encourages the embeddings to reflect relational connectivity and, in turn, promotes the retrieval of semantically coherent table sets.

Finally, the ATR objective function is defined as follows:

$$L_{ATR} = L_{AT} + \lambda L_{RC} + \gamma L_{SG}$$

where λ and γ are hyper-parameters that adjust the magnitude of the losses. These losses enable ATR to adaptively retrieve tables, considering the relevance between the query and the tables, as well as the relevance between tables.

Sliding Window Reranking Since the encoder used in ATR has a strict length constraint and a quadratic complexity with respect to input length, directly processing large numbers of tables at once is computationally impractical. To mitigate this inefficiency, ATR uses a *sliding window reranking* strategy. Given a window size W and a retention size R with $R < W$, ATR processes the tables $\mathcal{C}$ from lowest to highest in their initial ranking. First, in W lowest-ranked tables, ATR computes logits for every token T_i and for the threshold token T_{th}, and keeps the top R tables by logit value. Then the retained set is merged with the next $W - R$ tables in the original order. If the threshold logit $logit_{T_{th}}$ ranks lower than R within W, its rank is finalized. This iterative process continues until all candidate tables have been processed. Eventually, all the tables that outrank the threshold are included in the final table list. Since ATR reranks subsets of tables within overlapping windows, the method avoids the cost of reranking the full list at once. Pseudo-code for this sliding window reranking is presented in Algorithm 1 and an illustrative example is provided

		Spider				BIRD				Spider 2.0			
	k	P	R	CR	F1	P	R	CR	F1	P	R	CR	F1
Encoder-based													
Contriever	3	46.5	94.0	89.2	60.1	45.6	72.9	55.1	54.6	24.7	37.6	24.8	27.4
	5	29.3	97.4	95.5	43.7	31.2	82.1	68.6	44.2	19.1	45.1	32.6	24.4
	10	15.0	99.2	98.7	25.6	18.5	96.1	92.6	30.5	13.3	56.5	44.8	19.5
UAE	3	46.1	93.0	88.0	59.5	48.2	79.0	63.5	58.3	27.1	40.6	27.6	29.7
	5	29.5	97.8	96.4	44.1	32.7	87.2	77.3	46.5	20.7	48.5	34.9	26.3
	10	15.0	99.4	99.0	25.6	18.7	97.6	95.1	30.9	14.2	60.7	49.0	20.7
LLM-based													
RankZephyr	3	47.7	95.9	92.8	61.5	47.9	76.6	60.4	57.3	26.0	39.3	26.4	28.7
	5	29.5	98.2	96.9	44.1	32.1	84.4	72.4	45.5	19.7	46.7	33.8	25.2
	10	15.1	99.5	99.2	25.7	18.5	96.5	93.1	30.6	13.6	57.3	45.5	19.8
Murre	3	40.5	82.9	73.8	52.5	44.6	72.5	54.6	53.7	28.1	42.3	29.4	31.0
	5	26.9	90.3	85.5	40.3	30.1	80.5	66.4	42.8	21.8	51.1	36.8	28.0
	10	14.6	96.8	95.1	24.8	16.7	88.2	78.3	27.6	14.8	61.9	48.5	21.5
Reranker-based													
JAR (w. Contriever)	3	48.1	96.5	93.6	62.0	**54.4**	87.4	76.3	65.0	**29.4**	42.3	26.9	**31.6**
	5	29.7	98.5	97.2	44.3	35.1	92.5	85.9	49.7	21.9	48.6	34.0	27.0
	10	15.1	99.5	99.2	25.7	18.8	97.6	96.0	31.0	14.0	55.1	41.4	19.5
JAR (w. UAE)	3	48.4	96.5	94.1	62.3	53.6	86.3	74.8	64.4	29.1	41.6	27.1	30.8
	5	29.9	99.1	98.5	44.7	34.4	91.1	82.9	48.9	21.8	47.9	33.8	26.5
	10	15.1	99.5	99.2	25.7	18.7	97.2	95.2	30.9	14.3	56.4	43.9	19.7
Ours													
ATR (w. Contriever)		**69.6**	99.5	99.2	**78.3**	54.0	98.2	96.0	**65.8**	21.9	72.4	64.4	27.8
ATR (w. UAE)		69.3	**99.6**	**99.4**	78.1	52.8	**98.6**	**97.1**	65.1	19.9	**75.4**	**68.7**	26.7

Table 1: Retrieval performance comparison with baseline methods, evaluated using Precision (P), Recall (R), Complete Recall (CR), and F1 scores (F1). We use SGPT-5.8B as the backbone model for Murre. ATR consistently outperforms all baselines across datasets. The best and second-best scores for each metric are highlighted in **bold** and underlined, respectively.

in Figure 7.

5 Experiments

5.1 Setup

Dataset We use three datasets: Spider (Yu et al., 2018), BIRD (Li et al., 2023), and Spider 2.0 (Lei et al., 2025). Spider and BIRD are widely used benchmarks for text-to-SQL. We adopt the "union" setting, merging all databases into a single corpus (Kothyari et al., 2023; Zhang et al., 2025). Spider 2.0 is a benchmark composed of enterprise text-to-SQL workflows derived from a large-scale database. We use Spider 2.0-Lite[1], a subset featuring multiple SQL dialects, and apply the union setting grouped by dialect. We note that ATR training utilizes only the training sets of Spider and BIRD. Spider and BIRD contain queries requiring between 1 and 5 tables, while Spider 2.0 queries range from 1 to 366 tables. Detailed data preprocessing methods are described in Appendix B, and dataset statistics are presented in Table 6.

Task and Metrics We evaluate ATR on two tasks: table retrieval and text-to-SQL generation. In table retrieval, given a natural language query q, a model retrieves a set of tables $\hat{\mathcal{C}}_q \subset \mathcal{C}$ from the table corpus $\mathcal{C}$. We report *precision*, *recall*, and *F1*, as well as *complete recall* by comparing $\hat{\mathcal{C}}_q$ with the ground-truth set $\mathcal{C}_q$, following Zhang et al. (2025). In the text-to-SQL task, the query q and its retrieved tables $\hat{\mathcal{C}}_q$ are fed to a generator that produces a SQL statement. We measure *execution accuracy*: the proportion of generated SQL queries whose execution results match those of the reference SQL (Yu et al., 2018). To evaluate downstream performance, we ensure that the only difference between retrieval methods is the input tables, allowing a precise assessment of how retrieval performance influences the downstream results.

[1]For simplicity, we refer to Spider 2.0-Lite as Spider 2.0.

	k	Spider			BIRD			Spider 2.0		
		Llama (8B/70B)	Qwen (7B/32B)	Gemma (4B/27B)	Llama (8B/70B)	Qwen (7B/32B)	Gemma (4B/27B)	Llama (8B/70B)	Qwen (7B/32B)	Gemma (4B/27B)
Encoder-based										
Contriever	3	54.6 / 64.8	65.4 / 66.3	58.0 / 67.4	21.7 / 40.4	35.4 / 46.0	21.2 / 38.9	1.1 / 3.4	0.5 / 3.9	0.2 / 2.5
	5	53.3 / 65.0	65.7 / 68.6	58.3 / 67.7	22.4 / 44.5	37.0 / 49.8	24.1 / 39.7	1.1 / 3.2	1.4 / 3.7	0.7 / 3.2
	10	54.6 / 67.1	66.0 / 70.1	60.3 / 67.6	24.3 / 47.4	39.1 / 53.0	25.6 / 43.2	1.1 / 3.5	0.9 / 4.1	0.2 / 3.4
UAE	3	55.7 / 65.0	63.6 / 68.0	52.5 / 68.1	22.6 / 40.7	35.1 / 45.4	20.1 / 38.8	0.7 / 3.5	1.1 / 3.4	0.5 / 3.4
	5	54.9 / 66.1	66.0 / 69.6	52.8 / 70.2	23.9 / 45.0	37.8 / 49.5	22.0 / 43.5	0.9 / 3.9	1.1 / 3.4	0.9 / 4.1
	10	54.7 / 67.3	66.1 / 70.6	54.1 / 69.2	26.0 / 47.9	40.4 / 52.2	22.2 / 46.8	1.4 / 4.1	1.4 / 3.4	0.5 / 3.9
LLM-based										
RankZephyr	3	53.3 / 64.8	63.1 / 66.6	54.2 / 69.4	22.4 / 40.7	34.5 / 45.8	21.7 / 43.2	1.1 / 3.2	1.4 / 4.1	0.2 / 2.5
	5	54.8 / 65.9	65.4 / 67.2	58.2 / 70.2	23.0 / 45.0	38.1 / 46.2	23.2 / 45.5	0.9 / 3.0	1.1 / 4.1	0.7 / 3.0
	10	57.5 / 67.0	66.4 / 67.3	59.0 / 70.7	25.4 / 47.5	39.3 / 49.5	24.0 / 46.2	1.1 / 3.4	1.4 / 3.4	0.5 / 4.4
Murre	3	51.2 / 65.3	63.4 / 64.6	53.6 / 69.3	21.8 / 44.5	34.2 / 44.2	20.9 / 42.8	0.9 / 4.1	1.4 / 3.2	0.9 / 4.1
	5	53.7 / 65.6	64.1 / 65.8	54.1 / 70.5	22.6 / 44.8	37.5 / 45.4	20.7 / 44.5	0.9 / 3.0	1.4 / 3.7	0.9 / 3.0
	10	54.5 / 65.9	65.3 / 68.0	55.0 / 70.2	25.3 / 46.4	39.4 / 48.4	21.2 / 47.0	0.7 / 4.1	1.4 / 4.4	0.7 / 4.6
Reranker-based										
JAR (w. Contriever)	3	53.9 / 65.9	66.6 / 68.8	54.8 / 69.1	25.4 / 45.3	40.1 / 49.7	23.9 / 44.4	0.7 / 3.7	0.9 / 4.1	0.5 / 3.0
	5	54.8 / 66.6	67.5 / 69.7	59.4 / 67.8	26.8 / 46.4	43.4 / 52.0	27.4 / 43.2	1.1 / 3.9	0.7 / 3.7	0.9 / 3.2
	10	56.0 / 66.8	66.6 / 69.2	59.5 / 67.7	26.7 / 47.5	43.0 / 53.0	24.8 / 44.9	0.9 / 3.0	1.4 / 4.4	0.5 / 4.1
JAR (w. UAE)	3	54.6 / 67.2	66.7 / 66.8	55.1 / 69.2	24.8 / 44.7	38.7 / 44.9	22.6 / 44.3	0.9 / 3.9	1.6 / 4.4	0.5 / 2.8
	5	56.0 / 67.4	67.0 / 69.7	55.3 / 70.2	25.3 / 45.4	39.2 / 48.4	23.1 / 44.7	0.9 / 4.1	1.4 / 4.1	0.9 / 4.4
	10	55.6 / 66.9	64.6 / 67.7	56.6 / 70.8	26.6 / 46.9	41.3 / 52.2	24.2 / 46.6	0.9 / 3.0	1.6 / 3.7	0.9 / 3.2
Ours										
ATR (w. Contriever)		58.7 / 67.8	69.7 / **71.5**	**60.7** / 67.8	28.6 / **49.9**	**45.0** / 53.3	**28.8** / 45.1	1.1 / 4.4	1.4 / **5.7**	**1.1** / **4.8**
ATR (w. UAE)		**59.8** / **68.2**	**69.8** / 71.4	59.0 / **71.1**	**29.0** / 48.7	43.8 / **53.9**	25.6 / **49.7**	**1.6** / **4.5**	**2.1** / 4.9	**1.1** / 4.6
Oracle		66.6 / 70.8	75.6 / 75.2	66.5 / 71.7	31.8 / 53.5	50.6 / 58.0	32.7 / 50.9	4.4 / 7.2	3.5 / 7.4	1.4 / 7.6

Table 2: Text-to-SQL execution accuracy comparison across different table retrieval methods. ATR consistently outperforms baseline retrievers on the Spider, BIRD, and Spider 2.0 datasets.

Models We use ModernBERT-large (Warner et al., 2024), a bidirectional encoder-only transformer, as the backbone model for ATR. For table embedding models, we use Contriever (Izacard et al., 2021) and UAE-Large-V1 (Li and Li, 2024). For LLM-based retrieval, we leverage SGPT-5.8B (Muennighoff, 2022) and RankZephyr-7B-V1 (Pradeep et al., 2023). For SQL generation, we utilize Llama-3.1-8B/70B-Instruct (Grattafiori et al., 2024), Qwen2.5-Coder-7B/32B-Instruct (Hui et al., 2024), and Gemma-3-4B/27B-IT (Team et al., 2025) as generators. Results for additional models, including proprietary models, are demonstrated in Appendix C and E.

Baseline We compare ATR with two bi-encoder baselines, Contriever and UAE-Large-V1, and with the reranking method JAR (Chen et al., 2024b), which encodes table joinability. Contriever and UAE-Large-V1 embed the query and each table independently by flattening tables into text, and rank tables based on cosine similarity between their embedding vectors. Both ATR and JAR utilize a two-stage retrieval pipeline, where a bi-encoder first retrieves the top 50 candidate tables, and a reranker subsequently refines their order, following standard practice (Glass et al., 2022; Sun et al., 2023; Qin et al., 2024; Wasserman et al., 2025). Additionally, we include LLM-based retrieval methods: RankZephyr (Pradeep et al., 2023), an open-source LLM specialized in listwise zero-shot reranking, and Murre (Zhang et al., 2025), a multi-hop retrieval method based on LLM that iteratively retrieves tables.

5.2 Table Retrieval Performance

Across all tested values of k, ATR outperforms most of the top-k baselines on the in-domain datasets (Spider and BIRD) and out-of-domain dataset (Spider 2.0), as shown in Table 1. On Spider, ATR achieves substantial gains over all the comparison targets, improving precision by more than 20.0% and F1 by over 15.0%. Compared to other baselines, it performs almost the best on precision and recall, and achieves the highest F1 score on BIRD. Although no query in Spider or BIRD requires more than five tables and the baselines already use conservative k values, our method still surpasses them. The performance improvement of recall becomes more evident on Spider 2.0, where the number of ground-truth tables varies from one to hundreds. ATR shows a characteristic that the recall and complete recall scores are at least 10.0% higher than other models. These findings confirm that the top-k retrieval strategy cannot be general-

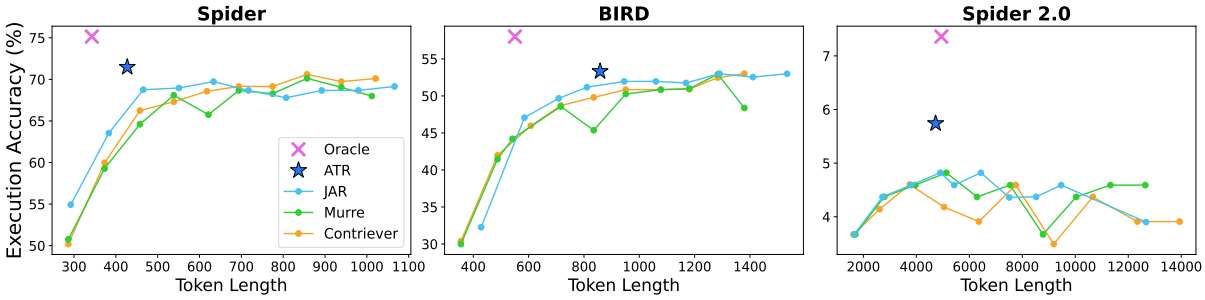

Figure 4: Comparison of execution accuracy and average token length for the text-to-SQL task across different retrieval methods. ATR achieves higher accuracy with fewer tokens compared to the best-performing top-k approach.

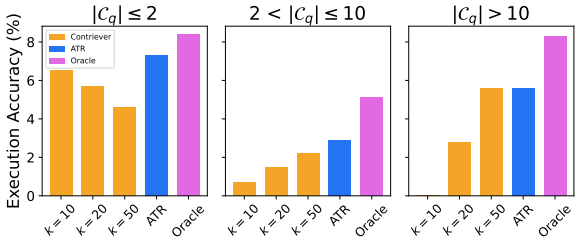

Figure 5: Execution accuracy on the Spider 2.0 dataset across varying numbers of required tables. $|\mathcal{C}_q|$ denotes the number of tables required per query.

ized across queries requiring varying numbers of tables, whereas ATR demonstrates robustness to such variations.

5.3 End-to-end Performance

We evaluate end-to-end effectiveness through the text-to-SQL task. To set an upper bound for retrieval-based approaches, we report an *Oracle* setting where the generator receives only the ground-truth tables. As described in Table 2, ATR shows the best execution accuracy across all three datasets and evaluated generators. With Qwen2.5-Coder-32B and $k = 10$, ATR improves on JAR by 3.7% on Spider and 1.7% on BIRD when using UAE-Large-V1 as the bi-encoder; the performance gains increase to 5.2% and 2.5% when the 7B model is used, respectively. Compared to Oracle, ATR has a difference only 2.6% and 3.6% on Spider and BIRD with Llama-3.1-70B, respectively. On Spider 2.0, there is a difference of only 1.4% with Qwen2.5-Coder-7B. These results confirm earlier findings that stronger retrieval performance translates into higher downstream accuracy (Kothyari et al., 2023; Chen et al., 2024b).

6 Analysis

6.1 Efficiency and Accuracy Improvement

Retrieving too few tables omits evidence and degrades performance, whereas retrieving too many injects noise and inflates inference cost. We analyze how many tables ATR and top-k methods retrieve compared to the Oracle, and how this affects downstream performance. For the top-k methods, we vary k from 1 to 10 for Spider and BIRD, and from 5 to 50 with intervals of 5 for Spider 2.0.

Figure 4 shows execution accuracy against the average number of input tokens used to generate SQL queries. ATR achieves higher execution accuracy with fewer tokens than top-k baselines on all three datasets. Specifically, ATR uses 430 fewer tokens than the best-performing top-k ($k = 8$) on Spider, and 522 fewer tokens than the best top-k ($k = 10$) on BIRD. When compared to top-k baselines with similar token budget, ATR demonstrates higher execution accuracy—improving by 5.2% on Spider ($k = 3$) and 3.5% on BIRD ($k = 5$). The gap widens on the out-of-domain Spider 2.0 dataset, where ATR narrows the gap toward the Oracle, which is the upper bound. These results demonstrate that ATR is a robust method that retrieves ground-truth tables accurately while minimizing the retrieval of irrelevant tables.

6.2 Performance by Number of Required Tables

The fixed k retrieval strategy inherently suffers from trade-offs; either failing to retrieve necessary tables or retrieving irrelevant ones. To illustrate this trade-off and demonstrate the robustness of our method, we analyze the downstream task performance by categorizing the Spider 2.0 queries into three groups based on the number of ground-truth tables: two or fewer, between three and ten, and more than ten.

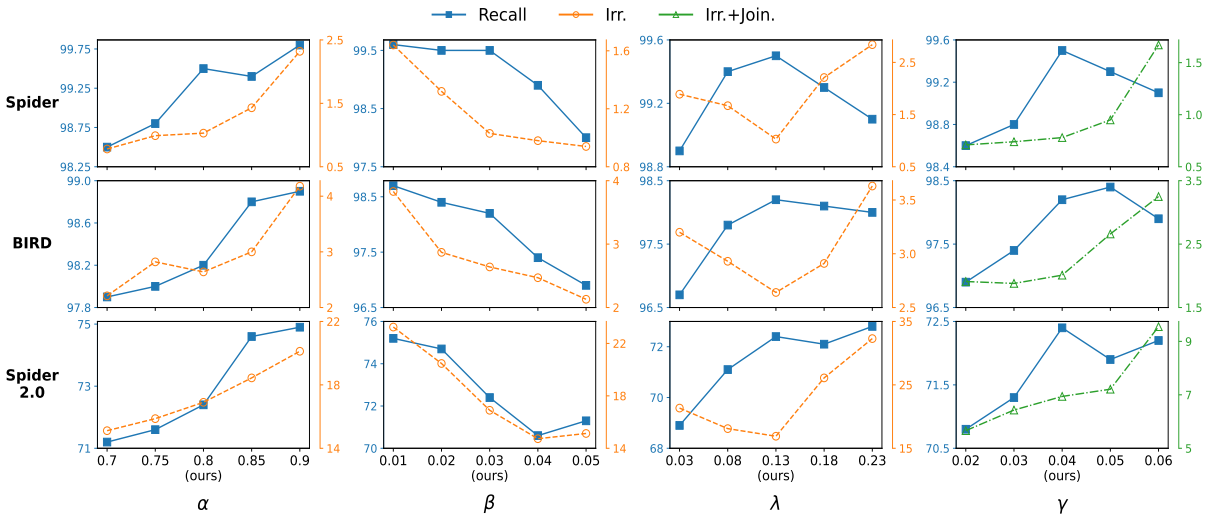

Figure 6: Analysis of training loss hyper-parameters. Irr. indicates the number of retrieved tables irrelevant to the query, and Irr.+Join. represents the retrieved tables that are irrelevant but can be joined with the ground-truth tables.

	Spider			BIRD			Spider 2.0		
W*, *R	50, N/A	20, 15	10, 6	50, N/A	20, 15	10, 6	50, N/A	10, 5	6, 4
Time	0.12	0.14	0.17	0.18	0.21	0.26	0.60	0.41	0.45
Length	1164.4	471.7	237.5	1592.7	617.9	324.0	3982.8	846.2	532.8
# Inference	1	7	11	1	7	11	1	9	23

Table 3: Analysis of sliding window hyper-parameters (W, R), where W denotes the window size and R is the retention size. The metrics include average inference time in seconds (Time), average input token length (Length), and number of inferences (# Inference).

Figure 5 illustrates the limitations of a fixed k approach. Top-k retrieval performs best on queries that require two or fewer tables when using a smaller retrieval count ($k = 10$), but its performance collapses for queries that need more than ten tables. Conversely, using a larger retrieval count ($k = 50$) enhances performance for queries that require more than ten tables, but it falls for queries that require two or fewer tables because of noise from irrelevant tables. ATR addresses this trade-off, consistently outperforming baselines across queries with various table requirements.

6.3 Comparison with Adaptive Document Retrieval

Recent adaptive document retrieval methods adjust the retrieval strategy based on query complexity. FLARE (Jiang et al., 2023) triggers additional retrieval whenever the generator outputs low-confidence tokens, and Adaptive-RAG (Jeong et al., 2024) trains a classifier to determine the number of retrieval iterations based on query complexity. Iter-RetGen (Shao et al., 2023) serves as an iterative baseline, retrieving documents repeatedly for a fixed number of iterations. To compare ATR with these adaptive document retrieval methods, we conduct experiments on the Spider and BIRD datasets with Qwen2.5-Coder-32B. We evaluate recall, text-to-SQL execution accuracy, and end-to-end latency in seconds. To ensure fairness, we train the Adaptive-RAG classifier on the Spider and BIRD training splits.

	Spider			BIRD		
	R	**Acc.**	**Time(↓)**	**R**	**Acc.**	**Time(↓)**
Iter-RetGen	<u>98.8</u>	**71.7**	8.9	<u>96.5</u>	<u>52.7</u>	14.4
FLARE	89.0	63.2	<u>4.0</u>	78.3	43.1	<u>5.7</u>
Adaptive-RAG	86.0	62.8	6.3	89.8	50.7	13.2
ATR (Ours)	**99.5**	<u>71.5</u>	**2.2**	**98.2**	**53.3**	**3.8**

Table 4: ATR outperforms existing adaptive document retrieval strategies in both efficiency and accuracy. Acc. and Time denote execution accuracy and the average end-to-end inference time in seconds, respectively.

Table 4 shows that ATR consistently outperforms adaptive document retrieval baselines in

both retrieval and end-to-end tasks. Compared to Adaptive-RAG, ATR improves execution accuracy by 8.7% on Spider and 2.6% on BIRD, simultaneously reducing processing times by 4.1 and 9.4 seconds, respectively. Against Iter-RetGen, ATR reduces the inference time by 6.7 seconds on Spider and 10.6 seconds on BIRD, while achieving comparable execution accuracy. These results confirm the effectiveness of ATR through improved table representation learning and the efficiency of operating without iterative LLM retrieval interactions.

6.4 Hyper-parameter Analysis

We analyze how retrieval behavior changes with respect to hyper-parameters controlling each loss component. α and β control the margin between relevant and irrelevant tables based on the threshold token. As shown in Figure 6, increasing α and decreasing β tend to increase the number of tables included in retrieval results, consequently improving recall but also increasing the number of irrelevant tables. Similarly, λ and γ significantly impact both recall and the number of retrieved tables. Notably, as γ increases, retrieval results include more tables that are irrelevant but joinable with the ground-truth tables. These observations align closely with our design of the loss functions. Furthermore, an ANOVA test confirms that the threshold token forms a statistically significant decision boundary ($p < 0.05$) separating relevant and irrelevant tables, as detailed in Appendix F.

ATR performs re-ranking on the top 50 tables initially retrieved by bi-encoder, varying the window and retention sizes. Depending on the window size and retention size, inference time can vary due to changes in the average input token length and the number of required inferences. We present in Table 3 how the input length, number of inferences, and inference time vary according to the window and retention sizes. For Spider and BIRD datasets, tables generally have shorter schemas, resulting in shorter token lengths; thus, inference time is primarily influenced by the number of inferences. Conversely, for Spider 2.0, which contains larger individual tables with longer schemas, inference time is predominantly determined by the window size, directly affecting input token length per inference.

6.5 Ablation Study

ATR is trained with two auxiliary objectives: a BCE loss for relevance calibration and a contrastive loss for semantic grouping. To evaluate each loss component, we train separate models by removing each objective individually.

	Spider		BIRD		Spider 2.0	
	R	**CR**	**R**	**CR**	**R**	**CR**
ATR	**99.5**	**99.2**	**98.2**	**96.0**	**72.4**	**64.4**
− (1) BCE	99.0	98.7	97.5	95.8	68.2	60.8
− (2) Contrastive	99.0	98.4	97.4	95.2	69.1	60.1
− (1) & (2)	96.4	94.4	91.8	85.7	67.7	58.2

Table 5: Ablation study on table representation losses.

As shown in Table 5, removing the BCE loss results in both lower recall and complete recall, confirming that explicit query-table alignment is crucial for distinguishing relevant tables from irrelevant ones. Similarly, removing contrastive loss reduces retrieval performance, indicating that inter-table joinability enhances the discriminative quality of table embeddings, thereby improving retrieval accuracy. These findings indicate that both losses are essential for learning robust and discriminative table representations, which in turn enhance the retrieval performance of ATR over existing table retrieval methods.

7 Conclusion

In this work, we address the inherent limitations of conventional table retrieval methods that rely on a fixed k retrieval strategy. The rigidity in the fixed k often degrades downstream task performance and efficiency by retrieving unnecessary tables or failing to retrieve tables required for accurate reasoning. To mitigate these problems, we propose ATR, an adaptive table retrieval method that dynamically adjusts the number of retrieved tables based on the query. ATR leverages adaptive thresholding to determine the optimal number of tables required for each query. Furthermore, ATR adopts relevance calibration and semantic grouping loss to effectively learn table representations by capturing query-to-table and inter-table relationships. Extensive experiments demonstrate that ATR consistently outperforms top-k retrieval methods, demonstrating superior retrieval performance, downstream accuracy, and inference efficiency. These results confirm that ATR can be a practical solution suitable for large-scale database retrieval applications.

Limitations

Although ATR demonstrates substantial improvements in both retrieval and downstream execution accuracy, there is room for further improvement. ATR currently targets structured tabular data exclusively, and extending its adaptive retrieval strategy to handle other data modalities or mixed data types remains an open research challenge. Nevertheless, ATR provides a robust and efficient adaptive retrieval framework without direct interaction with generative LLMs. By explicitly learning effective table representations and efficiently managing retrieval through adaptive thresholding, ATR establishes a strong foundation for future studies aiming to generalize retrieval-augmented generation across various data types and broader application contexts.

Acknowledgements

This work was supported by the Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST)). It was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2025-00555621), and by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-02304967, AI Star Fellowship (KAIST)).

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. *Proceedings of the ACM on Management of Data*, 3(3):1–28.

Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2024a. Beaver: an enterprise benchmark for text-to-sql. *arXiv preprint arXiv:2409.02038*.

Peter Baile Chen, Yi Zhang, and Dan Roth. 2024b. 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.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PmLR.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2022. Deepjoin: Joinable table discovery with pre-trained language models. *arXiv preprint arXiv:2212.07588*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Naik, Pengshan Cai, and Alfio Gliozzo. 2022. Re2G: Retrieve, rerank, generate. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2701–2715, Seattle, United States. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

R. Hadsell, S. Chopra, and Y. LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742.

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.

Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. 2024. Knowledge-to-SQL: Enhancing SQL generation with data expert LLM. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10997–11008, Bangkok, Thailand. Association for Computational Linguistics.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

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*.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7029–7043.

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore. Association for Computational Linguistics.

Deokhyung Kang, Baikjin Jung, Yunsu Kim, and Gary Geunbae Lee. 2024. Denoising table-text retrieval for open-domain question answering. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 4634–4640, Torino, Italia. ELRA and ICCL.

Kezhi Kong, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Chuan Lei, Christos Faloutsos, Huzefa Rangwala, and George Karypis. 2024. Opentab: Advancing large language models as open-domain table reasoners. *arXiv preprint arXiv:2402.14361*.

Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. 2023. CRUSH4SQL: Collective retrieval using schema hallucination for Text2SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14054–14066.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Da Li, Keping Bi, Jiafeng Guo, and Xueqi Cheng. 2025a. Tailoring table retrieval from a field-aware hybrid matching perspective. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 27681–27692, Suzhou, China. Association for Computational Linguistics.

Jinyang Li, Binyuan Hui, GE QU, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Millicent Li, Tongfei Chen, Benjamin Van Durme, and Patrick Xia. 2025b. Multi-field adaptive retrieval. In *The Thirteenth International Conference on Learning Representations*.

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

Niklas Muennighoff. 2022. Sgpt: Gpt sentence embeddings for semantic search. *arXiv preprint arXiv:2202.08904*.

OpenAI. 2025. GPT-5.1: A smarter, more conversational ChatGPT. `https://openai.com/index/gpt-5-1/`. Accessed: 2025-12-14.

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*.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. Rankzephyr: Effective and robust zero-shot listwise reranking is a breeze! *ArXiv*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2024. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1504–1518.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937. Association for Computational Linguistics.

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

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro Szekely. 2021. Retrieving complex tables with multi-granular graph representation learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1472–1482.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. 2024. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *Preprint*, arXiv:2412.13663.

Navve Wasserman, Oliver Heinimann, Yuval Golbari, Tal Zimbalist, Eli Schwartz, and Michal Irani. 2025. DocReRank: Single-page hard negative query generation for training multi-modal RAG rerankers. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8651–8669, Suzhou, China. Association for Computational Linguistics.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. 2025. MMQA: Evaluating LLMs with multi-table multi-hop complex questions. In *The Thirteenth International Conference on Learning Representations*.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, Matrixmxlin Matrixmxlin, Liang Chen, Chenyun Yu, Cheng Lei, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 10796–10816.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-sql data from weak and strong llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Wenxuan Zhou, Kevin Huang, Tengyu Ma, and Jing Huang. 2021. Document-level relation extraction with adaptive thresholding and localized context pooling. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 14612–14620.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, and 1 others. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

A Implementation Details

We train ATR using two NVIDIA RTX A6000 GPUs, each equipped with 48GB of memory. Training is conducted for three epochs with a batch size of 64 and a learning rate of $3e^{-5}$. The maximum token length of ATR is set to 8,192, matching the input length constraints of ModernBERT-large. For the adaptive thresholding loss, the hyperparameters are set as $\alpha = 0.8$ and $\beta = 0.03$. For relevance calibration and semantic grouping loss, we select $\lambda = 0.13$ and $\gamma = 0.04$, respectively. The window size is set to 20 for the Spider and BIRD datasets, and 10 for Spider 2.0-Lite. The retention size is set to 15 for Spider and BIRD, and 5 for Spider 2.0-Lite.

For our experiments, we utilize the following models: We employ ModernBERT-large (Warner et al., 2024), specifically the `answerdotai/ModernBERT-large`, as the backbone of ATR. For table embedding, we employ Contriever (Izacard et al., 2021), using the `facebook/contriever-msmarco`, and UAE-Large-V1 (Li and Li, 2024) with the `WhereIsAI/UAE-Large-V1`. For LLM-based retrieval, we leverage RankZephyr (Pradeep et al., 2023), specifically the `castorini/rank_zephyr_7b_v1_full`, and SGPT (Muennighoff, 2022) from the `Muennighoff/SGPT-5.8B-weightedmean-msmarco-specb-bitfit`.

B Dataset Pre-processing

ATR is trained on the union of the Spider and BIRD training splits. For each query, we first use Contriever to retrieve the top 100 tables and then split this list in half: the higher-ranked segments that rank from 1 to 50 and the lower-ranked segments that rank from 51 to 100. We pair each segment with the query to create two training examples: one that is likely to contain relevant tables and one that is likely not. This contrastive environment makes ATR learn how to operate when the candidate set contains no relevant tables, reducing false positives at inference time. We split this dataset into training and validation sets at a ratio of 85% and 15% and the best checkpoint is selected by validation performance.

ATR adopts a semantic grouping loss to effectively learn table representations by capturing table-to-table relationships. To achieve this, we leverage joinability information between tables. Specifically, we identify joinable table groups by performing syntactic analysis on the publicly available database schemas from the Spider and BIRD training datasets. We exclude training samples for which joinability cannot be determined from the given database corpus. Additionally, we remove tables that exceed the maximum input token length of 512 tokens, consistent with the constraints of the bi-encoders used in our experiments, along with queries requiring these tables as ground truths. Furthermore, we exclude cases from Spider 2.0-Lite where tables labeled as ground truths are not present in the corresponding databases. Dataset statistics are summarized in Table 6.

Dataset	#Q	#DB	#T	Min/Max
Spider				
Train	6,989	140	737	1 / 5
Eval	1,034	20	81	1 / 4
BIRD				
Train	9,198	69	522	1 / 4
Eval	1,534	11	75	1 / 4
Spider 2.0				
Eval	435	155	6,321	1 / 366

Table 6: Data statistics. Number of queries (#Q), databases (#DB), tables (#T), and required tables (Min/-Max). Evaluations on Spider and BIRD use development sets.

		Spider		BIRD		Spider 2.0	
	k	R	CR	R	CR	R	CR
OpenAI	3	96.8	93.5	85.8	72.1	40.7	28.2
	5	99.7	99.4	92.8	85.5	50.0	36.7
	10	**100**	**100**	98.4	96.7	62.8	49.8
ATR (Ours)		99.6	99.6	**99.5**	**99.2**	**79.5**	**70.2**

Table 7: Evaluation of retrieval performance comparing ATR and a proprietary embedding model. OpenAI indicates text-embedding-3-large.

C Effectiveness of ATR Leveraging an Advanced Embedding Model

We evaluate the effectiveness of ATR when leveraging an advanced embedding model to potentially enhance retrieval performance. Our experiments use text-embedding-3-large, a state-of-the-art proprietary embedding model, and results are shown in Table 7. ATR achieves high complete recall, obtaining 99.2% on the BIRD and 70.2% on Spider 2.0. These results highlight ATR's robustness and effectiveness, demonstrating that it can achieve

		Spider				BIRD				Spider 2.0			
	k	P	R	CR	F1	P	R	CR	F1	P	R	CR	F1
RankGPT-4o-mini	3	48.2	96.8	93.5	62.1	*54.5*	88.0	76.2	65.6	33.2	48.5	33.8	35.2
RankGPT-4o-mini	5	29.9	98.9	97.9	44.5	35.0	92.7	84.6	49.7	24.6	56.1	42.1	30.5
RankGPT-4o-mini	10	15.1	99.3	99.0	25.5	18.7	97.5	94.6	30.9	15.3	64.0	52.2	21.9
RankGPT-5-mini	3	48.9	98.2	95.5	63.1	**57.3**	91.6	81.6	**66.3**	**36.6**	53.6	37.5	**39.4**
RankGPT-5-mini	5	30.0	99.5	98.9	44.8	36.4	95.7	90.0	51.5	25.8	59.3	44.1	32.4
RankGPT-5-mini	10	15.1	**100.0**	**100.0**	25.8	18.9	98.0	96.0	31.4	15.8	66.1	54.2	22.7
RankQwen3-32B	3	48.4	97.4	94.2	62.5	53.1	88.7	76.4	65.5	*33.3*	48.4	32.6	*35.4*
RankQwen3-32B	5	30.1	99.5	98.9	44.8	35.8	94.6	87.6	50.8	24.3	55.5	40.0	30.2
RankQwen3-32B	10	15.1	99.4	99.0	25.6	18.9	98.0	*96.4*	31.2	15.3	64.0	52.0	22.0
ATR (w. Contriever)		**69.6**	99.5	99.2	**78.3**	54.0	*98.2*	96.0	*65.8*	21.9	*72.4*	*64.4*	27.8
ATR (w. UAE)		*69.3*	*99.6*	*99.4*	*78.1*	52.8	**98.6**	**97.1**	65.1	19.9	**75.4**	**68.7**	26.7

Table 8: Performance comparison with LLM-based retrievers and rerankers. **Bold** indicates the highest score, and *italic* indicates the second-highest score.

strong retrieval performance when combined with advanced embedding models.

D Comparison with Combined LLM-based Pipelines

To provide a comprehensive evaluation against state-of-the-art LLMs, we expand our baselines to include combined LLM-based retriever-and-reranker pipelines. Specifically, we evaluate a pipeline that utilizes powerful LLMs, including GPT-4o-mini, GPT-5-mini, and Qwen3-32B, to rerank the initially retrieved tables at various fixed top-k cutoffs.

As shown in Table 8, ATR achieves comparable or superior performance across all datasets. While LLM-based rerankers inherently suffer from a precision-recall trade-off dictated by a fixed top-k value, ATR balances this without requiring a predefined cutoff. Crucially, LLMs impose substantial computational overhead and API costs. ATR, on the other hand, uses a lightweight bi-encoder to deliver highly efficient and practical retrieval suitable for real-world enterprise workflows.

E Additional End-to-end Performance Results

To further verify the robustness of the end-to-end performance of ATR, we conduct additional experiments on SQL generation utilizing Qwen3-Coder-30B-A3B-Instruct (Team, 2025), DeepSeek-Coder-V2-Lite-Instruct (Zhu et al., 2024), GPT-5-mini (OpenAI, 2025), and Gemini-2.5-flash (Comanici et al., 2025). As shown in Table 9, ATR consistently outperforms all baselines regardless of whether the generator is an open-source or proprietary model. Notably, using Gemini-2.5-flash, ATR narrows the performance gap with the Oracle setting to just 0.3% on Spider and 3.3% on BIRD. Furthermore, with Qwen3-Coder-30B-A3B-Instruct and DeepSeek-Coder-V2-Lite-Instruct, ATR maintains gaps of 4.2% and 2.1% in the Oracle setting on Spider 2.0, while substantially outperforming all baselines. These results demonstrate that ATR effectively retrieves the optimal set of tables, enabling advanced LLMs to fully leverage their reasoning capabilities.

F Statistical Analysis of Table Representations

ATR assigns logits on tokens T_{th} for the threshold and T_i for the table representation. We use analysis of variance (ANOVA) to investigate differences between group means within relevant tables, irrelevant tables, and a threshold. Most of the variance is explained by the difference of group means on Spider, revealing large effects ($\eta^2 \approx 0.95$) with significant p-values ($p < 0.05$). On the BIRD and Spider 2.0 datasets, ANOVA reveals significant effects ($\eta^2 \approx 0.86, 0.15$) with significant p-values ($p < 0.05$). A pairwise Tukey post-hoc test reveals a significant difference ($p < 0.05$ for all the pairs) between relevant tables, irrelevant tables, and the threshold for the three datasets. These results confirm that ATR robustly differentiates between relevant and irrelevant tables, with the adaptive threshold serving as a clear decision boundary that guides accurate table selection for each query.

		Spider				BIRD				Spider 2.0			
	k	Qwen3	DeepSeek-V2	GPT-5	Gemini-2.5	Qwen3	DeepSeek-V2	GPT-5	Gemini-2.5	Qwen3	DeepSeek-V2	GPT-5	Gemini-2.5
Encoder-based													
Contriever	3	71.7	61.7	60.4	73.7	45.2	29.9	35.1	48.8	3.2	1.8	3.2	2.5
	5	74.2	61.9	63.4	75.0	50.3	32.3	37.5	54.0	3.2	1.6	5.1	3.9
	10	75.9	62.3	64.8	76.5	53.7	36.8	40.4	58.1	3.7	1.6	5.8	4.4
UAE	3	71.9	61.5	60.6	73.3	46.1	31.8	34.2	50.5	3.7	1.6	4.6	2.5
	5	75.9	62.4	64.0	76.3	49.7	33.6	37.6	54.8	3.9	2.8	5.3	4.6
	10	76.2	63.8	65.1	76.9	54.0	38.3	41.3	59.3	4.4	1.8	7.6	4.6
LLM-based													
RankZephyr	3	71.3	62.4	60.1	73.2	45.1	29.7	36.0	48.6	3.2	2.1	5.1	4.1
	5	74.3	61.3	64.9	75.4	51.2	33.1	37.9	54.1	3.5	0.9	4.6	3.5
	10	76.2	63.2	65.0	76.9	54.6	37.0	40.0	58.7	3.7	1.8	7.4	5.1
Murre	3	71.3	62.4	60.7	73.9	49.3	30.1	36.1	50.3	2.5	2.3	4.1	3.5
	5	74.1	63.6	62.8	74.6	50.5	32.5	37.0	52.5	3.9	2.3	5.8	5.3
	10	75.9	64.4	64.3	76.2	53.6	37.3	41.3	58.7	4.1	2.1	7.4	5.5
Reranker-based													
JAR	3	74.5	63.8	63.2	74.8	49.3	36.3	37.1	55.3	2.3	2.1	4.8	3.2
(w. Contriever)	5	74.6	61.9	64.5	76.4	52.4	37.0	37.7	56.5	3.0	1.6	5.3	4.6
	10	75.6	63.1	66.2	76.5	54.0	38.2	39.4	58.1	3.9	2.3	5.5	4.4
JAR	3	74.6	62.9	65.4	75.8	49.6	36.3	37.2	52.9	3.0	2.5	4.4	4.1
(w. UAE)	5	76.2	63.1	67.1	77.4	50.6	36.8	38.0	54.3	3.2	2.3	5.8	4.6
	10	75.6	64.7	67.3	78.1	53.6	38.0	41.1	57.7	3.0	2.5	6.9	5.5
Ours													
ATR (w. Contriever)		76.3	**66.4**	65.5	77.9	**55.0**	38.5	41.6	59.4	4.6	**3.0**	**9.0**	5.8
ATR (w. UAE)		**76.7**	65.8	**67.8**	**79.4**	54.3	**39.6**	**41.9**	**59.5**	**4.8**	2.5	8.5	**6.7**
Oracle		79.0	71.4	68.3	79.7	59.5	44.9	43.1	62.8	9.0	5.1	11.5	8.9

Table 9: Text-to-SQL execution accuracy comparison for the various models, including open source and proprietary LLMs, across different table retrieval methods.

Dataset	Metric	Sliding Window (Ours)	No Sliding Window	Reduction
Spider	Avg Peak	30.62 MB	63.98 MB	52.1%
	Max Peak	44.01 MB	80.67 MB	45.4%
BIRD	Avg Peak	51.76 MB	93.32 MB	44.5%
	Max Peak	94.31 MB	145.13 MB	35.0%
Spider 2.0	Avg Peak	66.52 MB	340.57 MB	80.5%
	Max Peak	284.44 MB	1027.88 MB	72.3%

Table 10: GPU memory consumption profiling on the test sets. Avg Peak refers to the average maximum memory allocated per sample, and Max Peak denotes the single highest memory spike observed during inference.

G GPU Memory Profiling and Efficiency Analysis

Processing massive relational databases presents significant memory challenges because the aggregated input sequence length can easily exceed the strict context limits of standard encoders. Direct processing of all tables incurs quadratic memory costs due to the self-attention mechanism, leading to potential instability.

To demonstrate the effectiveness of our design, we compared our sliding-window strategy against a baseline that processes all retrieved tables simultaneously without a sliding window. As shown in Table 10, the efficiency gain is most pronounced in the Spider 2.0 dataset, which contains the largest and most complex schemas. Under this setting, ATR achieves an 80.5% reduction in average peak memory and completely prevents the massive memory spikes exceeding 1GB that are observed in the baseline. These findings confirm that the sliding-window strategy effectively bounds the peak memory per forward pass, ensuring stability and preventing out-of-memory errors even for extremely large-scale retrieval tasks.

H Case Study

We investigate the limitations of top-k approaches through qualitative analysis on specific examples from the BIRD and Spider 2.0 datasets. In these examples, relevant tables are retrieved using Contriever ($k = 5$) and ATR. SQL queries are generated using Qwen2.5-Coder-32B-Instruct.

As illustrated in Table 12, the top-k approach results in retrieving unnecessary tables when the number of required tables is fewer than k. These irrelevant tables can be noise, leading to confusion and incorrect SQL generation. Conversely, as illustrated in Table 13, when the query necessitates more tables than k, top-k retrieval fails to retrieve all essential tables, again resulting in incorrect SQL outputs. In contrast, ATR adaptively retrieves appropriate tables based on a query, effectively retrieving all necessary tables while minimizing irrelevant ones. This demonstrates ATR's clear advantage of providing precise and optimized input for the generator, significantly improving the accuracy and reliability of the generated SQL query.

To further demonstrate the robustness of our dy-

Algorithm 1 Adaptive Table Retrieval

Require: Query q, List of Table $\mathcal{C}$, Model M, Size of Window W, Size of Retention R, Number of Table C

Ensure: List of Ranked Table $\mathcal{C}'$

Variables:
  $\mathcal{C}' \leftarrow \emptyset, \mathcal{C}_{retain} \leftarrow \emptyset, idx \leftarrow C$
  $thr_{rank} \leftarrow 0, thr_{finalized} \leftarrow False$
while $idx > 0$ **do**
  if $\mathcal{C}_{retain} = \emptyset$ **then**
    $\mathcal{C}_{window} \leftarrow \mathcal{C}[idx - W :]$
    $idx \leftarrow idx - W$
  else
    $\mathcal{C}_{window} \leftarrow \mathcal{C}[idx-(W-R) : idx]+\mathcal{C}_{retain}$
    $idx \leftarrow idx - (W - R)$
  end if
  $logit, score \leftarrow M(q, \mathcal{C}_{window})$
  $\mathcal{C}_{retain} \leftarrow Decend_{score}(\mathcal{C}_{window})[: R]$
  if not $thr_{finalized}$ **then**
    $thr_{rank} \leftarrow Rank(logit) + idx$
    if $Rank(logit) > R$ **then**
      $thr_{finalized} \leftarrow True$
    end if
  end if
  $\mathcal{C}' \leftarrow \mathcal{C}' + Ascend_{score}(\mathcal{C}_{window} \setminus \mathcal{C}_{retain})$
end while
$\mathcal{C}' \leftarrow \mathcal{C}' + Ascend_{score}(\mathcal{C}_{retain})$
$\mathcal{C}' \leftarrow Reverse(\mathcal{C}')[: thr_{rank} - 1]$
return $\mathcal{C}'$

namic thresholding mechanism, we present three additional edge cases drawn from complex enterprise scenarios. As summarized in Table 11, these include: (1) cross-database joins, where queries require combining tables across different underlying datasets; (2) ambiguous table references, where identical table names appear in multiple databases and demand context-aware disambiguation; and (3) large-scale retrieval scenarios, where the number of gold tables far exceeds typical k values, causing fixed top-k approaches to structurally fail.

Case 1: Cross-Database Joins	
Query	*"Could you provide, for the United States, France, China, Italy, Spain, Germany, and Iran, the total number of confirmed COVID-19 cases as of April 20, 2020, along with the number of cases per 100,000 people based on their total 2020 populations calculated by summing all relevant population entries from the World Bank data"*
Gold	`covid19_jhu_csse.summary`, `world_bank_wdi.indicators_data`
ATR	`covid19_jhu_csse.summary`, `world_bank_wdi.indicators_data`
Top-5	`world_bank_health_population.health_nutrition_population`, `world_bank_health_population.health_nutrition_population`, `world_bank_health_population.health_nutrition_population`, `world_bank_global_population.population_by_country`, `world_bank_wdi.series_times`
Case 2: Ambiguous Table References	
Query	*"Calculate the average sales per order for each customer within distinct RFM segments, considering only 'delivered' orders"*
Gold	`E_commerce.orders`, `E_commerce.order_items`, `E_commerce.customers`
ATR	`orders` (E_commerce), `order_items` (E_commerce), `customers` (E_commerce)
Top-5	`orders` (electronic_sales), `orders` (E_commerce), `leads_closed` (E_commerce), `order_payments` (electronic_sales), `orders` (delivery_center)
Case 3: Large-Scale Retrieval	
Query	*"Could you tell me the average number of engaged sessions per user for December 2020, counting only those sessions where the event parameter 'session_engaged' is equal to '1' and using 'user_pseudo_id' combined with the 'ga_session_id' to identify distinct sessions?"*
Gold	`ga4.events_20201201` ... `ga4.events_20201231` (31 daily tables)
ATR	`events_20201201`, `events_20201202`, ..., `events_20201231` (31/31 retrieved)
Top-20	`events_20201201`, ..., `events_20201211` (11/31 retrieved)

Table 11: Edge-case retrieval results. Green denotes correctly retrieved gold tables; red denotes irrelevant or missed tables.

Question

Please list player names which are higher than 180.

Tables Retrieved by ATR	**Tables Retrieved by Top-k**
"Player"	"Player" "Match" "League" "Team" "superhero"

SQL Generated by ATR	**SQL Generated by Top-k**

`SELECT player_name FROM Player`
`WHERE height > 180`
	`SELECT T1.player_name FROM Player AS T1`
	`INNER JOIN Match AS T2`
	`ON T1.player_api_id = T2.home_player_1`
	`WHERE T1.height > 180`

Retrieved Table Schema

"Player": *id*, *player_api_id*, *height*, *weight*, *birthday*, *player_name*, *player_fifa_api_id*
"Match": *id*, *home_player_1*, *stage*, *goal*, *season*, *country_id*, *league_id*, *match_api_id*, ...
"League": *id*, *country_id*, *name*
"Team": *id*, *team_api_id*, *team_fifa_api_id*, *team_long_name*, *team_short_name*
"superhero": *id*, *superhero_name*, *full_name*, *gender_id*, *eye_colour_id*, *hair_colour_id*, ...

Table 12: A case study illustrating a scenario where the retrieval size k exceeds the number of tables required to resolve the query. As the top-k approach retrieves unnecessary tables, it propagates noise, degrading downstream SQL generation. Irrelevant tables are in red, and unnecessary tables that do not contribute to answering the query are highlighted in yellow. Retrieved table schema displays schema information for all retrieved tables.

Question

Please calculate the monthly average levels of PM10, PM2.5 FRM, PM2.5 non-FRM, volatile organic emissions, SO2 (scaled by a factor of 10), and Lead (scaled by a factor of 100) air pollutants in California for the year 2020.

Tables Retrieved by ATR	**Tables Retrieved by Top-k**
"pm25_frm_daily_summary"	"pm25_frm_daily_summary"
"lead_daily_summary"	"lead_daily_summary"
"pm10_daily_summary"	"pm10_daily_summary"
"pm25_nonfrm_daily_summary"	"pm25_frm_hourly_summary"
"so2_daily_summary"	"pm10_hourly_summary"
"voc_daily_summary"	"pm25_nonfrm_daily_summary"
	"so2_daily_summary"
	"voc_daily_summary"

SQL Generated by ATR

```
WITH pm10_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(arithmetic_mean) AS avg_pm10
FROM `bigquery-public-data.epa_historical_air_quality.pm10_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
pm25_frm_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(arithmetic_mean) AS avg_pm25_frm
FROM `bigquery-public-data.epa_historical_air_quality.pm25_frm_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
pm25_nonfrm_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(arithmetic_mean) AS avg_pm25_nonfrm
FROM `bigquery-public-data.epa_historical_air_quality.pm25_nonfrm_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
voc_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month,
AVG(arithmetic_mean) AS avg_voc
FROM `bigquery-public-data.epa_historical_air_quality.voc_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
so2_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month,
AVG(arithmetic_mean * 10) AS avg_so2_scaled
FROM `bigquery-public-data.epa_historical_air_quality.so2_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
lead_avg AS (
SELECT
EXTRACT(MONTH FROM date_local) AS month,
AVG(arithmetic_mean * 100) AS avg_lead_scaled
FROM `bigquery-public-data.epa_historical_air_quality.lead_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month)
SELECT
pm10_avg.month, pm10_avg.avg_pm10, pm25_frm_avg.avg_pm25_frm,
pm25_nonfrm_avg.avg_pm25_nonfrm,
voc_avg.avg_voc, so2_avg.avg_so2_scaled, lead_avg.avg_lead_scaled
FROM pm10_avg JOIN pm25_frm_avg ON pm10_avg.month = pm25_frm_avg.month
JOIN pm25_nonfrm_avg ON pm10_avg.month = pm25_nonfrm_avg.month
JOIN voc_avg ON pm10_avg.month = voc_avg.month
JOIN so2_avg ON pm10_avg.month = so2_avg.month
JOIN lead_avg ON pm10_avg.month = lead_avg.month
ORDER BY pm10_avg.month;
```

SQL Generated by Top-k

```
WITH pm10_monthly_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(sample_measurement) AS avg_pm10
FROM `bigquery-public-data.epa_historical_air_quality.pm10_hourly_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
pm25_frm_monthly_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(sample_measurement) AS avg_pm25_frm
FROM `bigquery-public-data.epa_historical_air_quality.pm25_frm_hourly_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
pm25_nonfrm_monthly_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(arithmetic_mean) AS avg_pm25_nonfrm
FROM `bigquery-public-data.epa_historical_air_quality.pm25_nonfrm_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
volatile_organic_emissions_monthly_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(arithmetic_mean) AS avg_volatile_organic_emissions
FROM `bigquery-public-data.epa_historical_air_quality.nox8_hourly_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020 AND
parameter_name = 'Nitric oxide'
GROUP BY month),
so2_monthly_avg AS (
SELECT EXTRACT(MONTH FROM date_local) AS month, AVG(arithmetic_mean) * 10 AS avg_so2_scaled
FROM `bigquery-public-data.epa_historical_air_quality.so2_daily_summary`
WHERE
state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month),
lead_monthly_avg AS (
SELECT
EXTRACT(MONTH FROM date_local) AS month, AVG(arithmetic_mean) * 100 AS avg_lead_scaled
FROM `bigquery-public-data.epa_historical_air_quality.lead_daily_summary`
WHERE state_name = 'California' AND EXTRACT(YEAR FROM date_local) = 2020
GROUP BY month)
SELECT
m.month, COALESCE(p10.avg_pm10, 0) AS avg_pm10,
COALESCE(p25frm.avg_pm25_frm, 0) AS avg_pm25_frm,
COALESCE(p25nonfrm.avg_pm25_nonfrm, 0) AS avg_pm25_nonfrm,
COALESCE(voe.avg_volatile_organic_emissions, 0) AS avg_volatile_organic_emissions,
COALESCE(so2.avg_so2_scaled, 0) AS avg_so2_scaled,
COALESCE(lead.avg_lead_scaled, 0) AS avg_lead_scaled
FROM (SELECT DISTINCT EXTRACT(MONTH FROM date_local) AS month
FROM `bigquery-public-data.epa_historical_air_quality.pm10_hourly_summary`
WHERE EXTRACT(YEAR FROM date_local) = 2020
) m LEFT JOIN pm10_monthly_avg p10 ON m.month = p10.month
LEFT JOIN pm25_frm_monthly_avg p25frm ON m.month = p25frm.month
LEFT JOIN pm25_nonfrm_monthly_avg p25nonfrm ON m.month = p25nonfrm.month
LEFT JOIN volatile_organic_emissions_monthly_avg voe ON m.month = voe.month
LEFT JOIN so2_monthly_avg so2 ON m.month = so2.month
LEFT JOIN lead_monthly_avg lead ON m.month = lead.month
ORDER BY m.month;
```

Retrieved Table Schema

"pm25_frm_daily_summary": *arithmetic_mean*, *date_local*, *state_code*, *county_code*, ...
"lead_daily_summary": *arithmetic_mean*, *date_local*, *state_code*, *county_code*, ...
"pm10_daily_summary": *arithmetic_mean*, *date_local*, *state_code*, *county_code*, ...
"pm25_nonfrm_daily_summary": *arithmetic_mean*, *date_local*, *state_code*, *county_code*, ...
"so2_daily_summary": *arithmetic_mean*, *date_local*, *state_code*, *county_code*, ...
"voc_daily_summary": *arithmetic_mean*, *date_local*, *state_code*, *county_code*, ...
"pm25_frm_hourly_summary": *date_local*, *sample_measurement*, *state_code*, *county_code*, ...
"pm10_hourly_summary": *date_local*, *sample_measurement*, *state_code*, *county_code*, ...

Table 13: A case study illustrating a scenario where the retrieval size k is smaller than the number of tables required to resolve the query. As the top- k approach fails to retrieve all necessary tables, it produces inaccurate SQL, whereas ATR retrieves all essential tables, enabling correct SQL generation. Tables not retrieved but crucial for SQL generation are displayed in light gray.

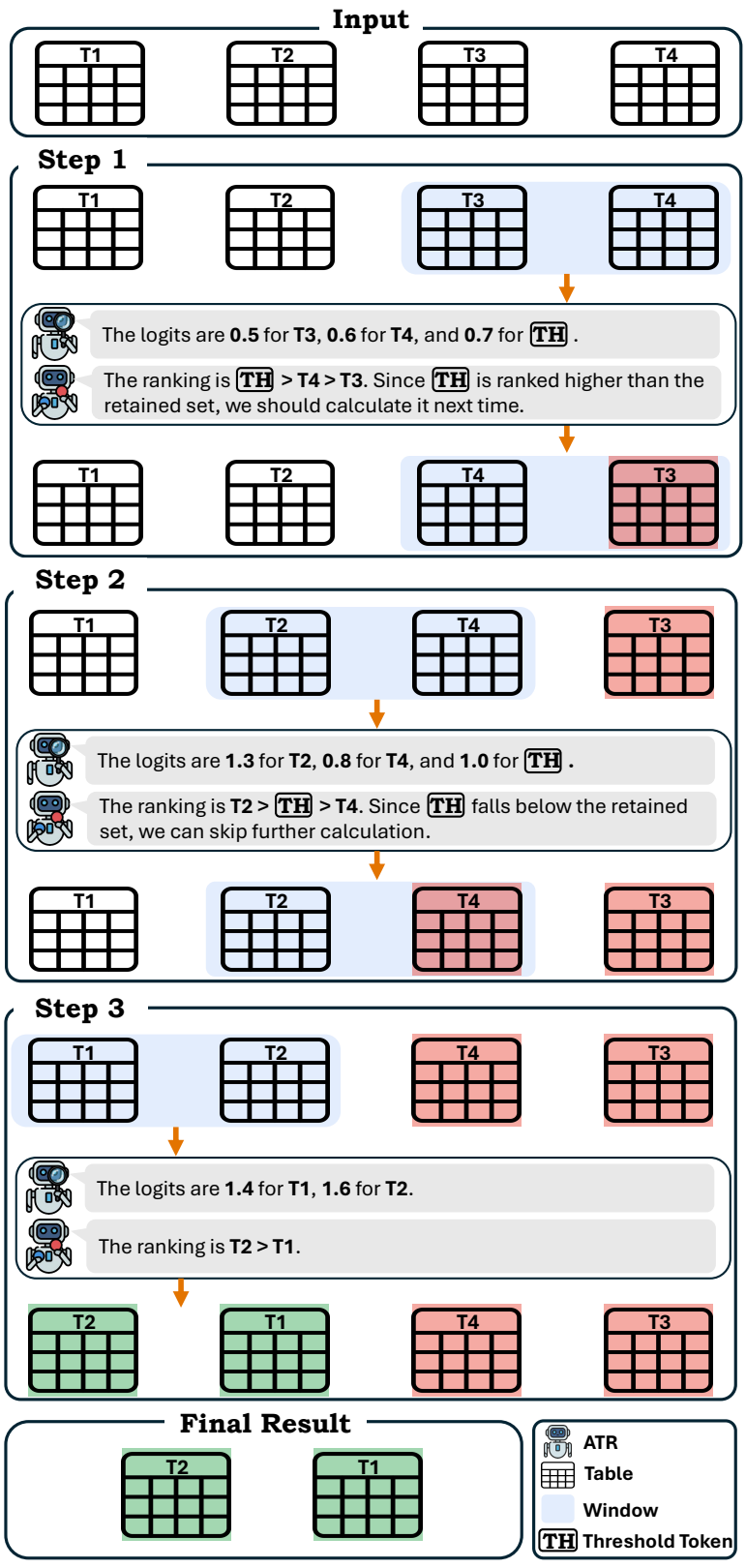

Figure 7: An illustrative example of the sliding window reranking process in ATR with four input tables, window size $W = 2$, and retention size $R = 1$. Tables highlighted in red indicate those whose rankings are finalized but excluded from the final set, while tables in green represent the tables that are ultimately retrieved. The input tables are initially ordered by query-table similarity in descending order from left to right. ATR processes tables starting from the lowest-ranked ones (rightmost) within the window. In each step, ATR compares the logits of table tokens with the threshold token (T_{th}) and retains the top-R tables for the next iteration.

```
[System]

You are a highly experienced data analyst with expert-level SQL skills. You have
been given a database schema, external knowledge and a question about the data.
Your task is to generate a valid SQLite query that correctly answers the question,
respecting any conditions or filters implied in the prompt.
Your answer should consist only of the SQL code, without additional explanations or
commentary.

Follow this Output Format:
SQL: <YOUR_SQL>

[User]
Database Schema:
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(255),
  major VARCHAR(255)
);

CREATE TABLE courses (
  course_id INT PRIMARY KEY,
  course_name VARCHAR(255),
  instructor VARCHAR(255)
);

Question: How many students are currently listed in the students table?

[Assistant]
SQL: SELECT count (*) FROM students

[User]
Database Schema:
{table_str}

External Knowledge:
{external_knowledge}

Question: {query_str}
```

Figure 8: Prompt template for Spider and BIRD datasets.

```
[System]
You are a highly experienced data analyst with expert-level SQL skills. You have been given a
database schema, external knowledge and a question about the data.
Your task is to generate a valid query that correctly answers the question, respecting any
conditions or filters implied in the prompt.
Your answer should consist only of the SQL code, without additional explanations or commentary.

Follow this Output Format:
SQL: <YOUR_SQL>

**IMPORTANT**
Use backticks for table identifiers (`project.dataset.table`).

[User]
Database Schema:
CREATE TABLE `SALES.CUSTOMERS` (
  id INT64,
  customer_name STRING,
  email STRING,
  address STRING,
  join_date DATE
);

CREATE TABLE `SALES.ORDERS` (
  order_id INT64,
  customer_id INT64,
  order_date DATE,
  amount NUMERIC,
  status STRING
);

Question: Find the top 5 customers with highest purchase amount

[Assistant]
SQL: SELECT
  C.customer_name,
  SUM(O.amount) AS total_purchases
FROM
  `SALES.CUSTOMERS` AS C
JOIN
  `SALES.ORDERS` AS O ON C.id = O.customer_id
GROUP BY
  C.customer_name
ORDER BY
  total_purchases DESC
LIMIT 5;

[User]
Database Schema:
{database_schema}

External Knowledge:
{external_knowledge}

Question: {question}
```

Figure 9: Prompt template for Spider 2.0 (BigQuery dialect)

```
[System]
You are a highly experienced data analyst with expert-level SQL skills. You have been given
a database schema, external knowledge and a question about the data.
Your task is to generate a valid query that correctly answers the question, respecting any
conditions or filters implied in the prompt.
Your answer should consist only of the SQL code, without additional explanations or
commentary.

Follow this Output Format:
SQL: <YOUR_SQL>

**IMPORTANT**
Use double quotes for Enclose all identifiers (database, schema, table, column names, and
aliases) in double quotes (").
SQL function names (YEAR, TO_TIMESTAMP, etc.) and keywords (SELECT, FROM, etc.) should NOT
be enclosed in quotes.
When using table aliases, the alias itself must also be enclosed in double quotes. Example:
"USERS" "U“
When referencing through aliases with dot notation, both parts need quotes: "U"."email“
Make sure all column references in SELECT, WHERE, GROUP BY, ORDER BY clauses use double
quotes.

[User]
Database Schema:
"CREATE TABLE “SALES”.”CUSTOMERS” (
  “id” INTEGER,
  “customer_name” VARCHAR(100),
  “email” VARCHAR(100),
  “address” VARCHAR(200),
  “join_date” DATE
);

CREATE TABLE “SALES”.”ORDERS” (
  “order_id” INTEGER,
  “customer_id” INTEGER,
  “order_date” DATE,
  “amount” DECIMAL(10,2),
  “status” VARCHAR(20)
);
Question: Find the top 5 customers with highest purchase amount

[Assistant]
SQL: SELECT
  "C"."customer_name",
  SUM("O"."amount") AS "total_purchases"
FROM
  "SALES"."CUSTOMERS" "C"
JOIN
  "SALES"."ORDERS" "O" ON "C"."id" = "O"."customer_id"
GROUP BY
  "C"."customer_name"
ORDER BY
  "total_purchases" DESC
LIMIT 5;

[User]
Database Schema:
{database_schema}

External Knowledge:
{external_knowledge}

Question: {question}
```

Figure 10: Prompt template for Spider 2.0 (Snowflake dialect)

```
[System]
You are a highly experienced data analyst with expert-level SQL skills. You have been given a database
schema, external knowledge and a question about the data.
Your task is to generate a valid query that correctly answers the question, respecting any conditions
or filters implied in the prompt.
Your answer should consist only of the SQL code, without additional explanations or commentary.

Follow this Output Format:
SQL: <YOUR_SQL>

[User]
Database Schema:
CREATE TABLE CUSTOMERS (
  id INTEGER,
  customer_name TEXT,
  email TEXT,
  address TEXT,
  join_date TEXT
);

CREATE TABLE ORDERS (
  order_id INTEGER,
  customer_id INTEGER,
  order_date TEXT,
  amount REAL,
  status TEXT
);

Question: Find the top 5 customers with highest purchase amount

[Assistant]
SQL: SELECT
  C.customer_name,
  SUM(O.amount) AS total_purchases
FROM
  CUSTOMERS C
JOIN
  ORDERS O ON C.id = O.customer_id
GROUP BY
  C.customer_name
ORDER BY
  total_purchases DESC
LIMIT 5;

[User]
Database Schema:
{database_schema}

External Knowledge:
{external_knowledge}

Question: {question}
```

Figure 11: Prompt template for Spider 2.0 (SQLite dialect)