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

Anonymous ACL submission

Abstract

Retrieving relevant tables from extensive databases for a given query is essential for accurately answering questions in tasks such as text-to-SQL and open-domain table question answering. Top-k retrieval strategies based on similarity scores between the query and tables are common in existing table retrieval methods. However, the number of required tables varies across queries and cannot be known in advance. Such strategies may either retrieve an undersized set of tables, preventing the model from gathering all that was needed, or retrieve too large a pool of tables, leading to the incorporation of unnecessary ones. To address this issue, we present an adaptive table retrieval method that adjusts the number of tables retrieved according to the requirements of each query. We adopt an adaptive thresholding mechanism to selectively retrieve tables and integrate a sliding-window re-ranking algorithm to efficiently process large candidate sets. Extensive experiments on Spider, BIRD, and Spider 2.0 show that our method effectively addresses the limitations of the top-k retrieval strategies, improving performance in both retrieval and downstream tasks. Our code and data are available at link.

1 Introduction

Recent advances in large language models (LLMs) have improved performance on tasks that require structured reasoning over tabular data (Gao et al., 2023; Yang et al., 2024; Xie et al., 2024). These improvements are important for 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; Kong et al., 2024).

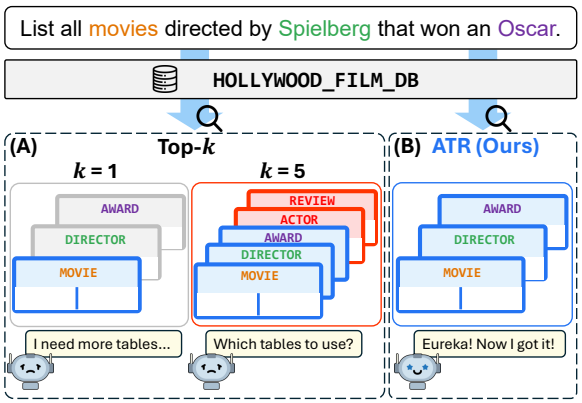

Figure 1: Rather than rely on a rigid retrieval strategy, ATR retrieves only relevant tables whether few or many. Gray indicates tables required by the query but not retrieved, red denotes irrelevant tables, and blue highlights retrieved relevant tables.

Existing table retrieval methods rely on a top-k retrieval strategy that selects the top k tables ranked by query-table similarity (Chen et al., 2024c; Zhang et al., 2025). However, this fixed cut-off ignores the fact that the number of tables required by each query is unknown in advance and can vary significantly. Even in the same database, ground-truth tables can range from a single table to several hundred. For example, the problem is acute in Spider 2.0 (Lei et al., 2025), an 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, top-k can miss necessary tables or retrieve too many tables to answer according to the size 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 retrieve two irrelevant ones. Subsequently, a small k sacrifices recall and a large k inflates latency and injects noise, degrading downstream performance (Kothyari et al., 2023). Fig-

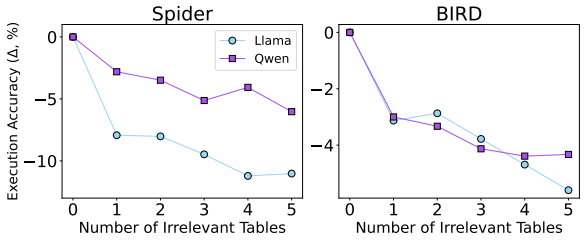

Figure 2: Retrieving irrelevant tables introduces noise, degrading performance on the text-to-SQL task. Execution accuracy consistently decreases as more irrelevant tables are added to the ground-truth tables.

ure 2 quantifies how the execution accuracy decreases as irrelevant tables are added.

To overcome the 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 required tables through an adaptive threshold mechanism that retrieves only those tables with logits exceeding the threshold. It then applies a sliding-window reranking module to reorder large candidate sets efficiently. Training is guided by a relevance calibration and a semantic grouping loss that jointly 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). Across all three benchmarks, ATR consistently outperforms top-k baselines retrieving fewer irrelevant tables, more essential ones, and boosting both retrieval metrics and downstream text-to-SQL execution accuracy. The improvements hold whether a query requires a single table or hundreds, demonstrating its robustness.

Our contributions can be summarized as follows:

- We show that top-k retrieval ignores the variation in required tables, leading to over-selection of irrelevant tables or omission of essential ones.
- We propose ATR, which uses query-specific thresholding to fetch exactly the necessary tables with relevance calibration and semantic grouping loss and combines sliding-window re-ranking to scale to a large table corpus.
- Experiments on Spider, Bird, and Spider 2.0 show that ATR consistently outperforms strong top-k baselines—using fewer tokens yet achieving higher execution accuracy—thereby improving both efficiency and downstream text-to-SQL performance.

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). Kothyari et al. (2023) and Zhang et al. (2025) propose table retrieval methods that rewrite user queries to enhance table retrieval accuracy. Chen et al. (2024c) and Wu et al. (2025) introduce methods that capture inter-table relationships, enhancing the coherence of retrieved tables. Li et al. (2025) propose a retrieval method that dynamically weights multiple fields of semi-structured data according to query semantics. Despite these advances, all of the above still rely on a top-k cut-off: with a small k they may omit essential tables, whereas a large k retrieves many irrelevant ones, degrading downstream accuracy and efficiency.

Adaptive Retrieval Strategy Recent RAG research has shown growing interest in adaptive retrieval methods, which adjust the size of text chunks based on query (Mallen et al., 2023; Jiang et al., 2023; Asai et al., 2023; Jeong et al., 2024). These approaches typically assess query complexity and selectively increase the retrieval budget accordingly. However, such methods are designed exclusively for the text domain and do not consider structured tabular data. Moreover, they typically involve iterative interactions with an LLM, increasing inference costs. We propose ATR, an adaptive table retrieval method capable of retrieving a query-dependent number of tables without relying on interactions with a generator.

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). Recent trends in text-to-SQL increasingly adopt retrieval-augmented approaches that integrates table retrieval with text-to-SQL generation to handle large-scale database scenarios (Kothyari et al., 2023; Kong et al., 2024; Chen et al., 2024c). Additionally, large-scale datasets recently introduced by Chen et al. (2024b) 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 downstream text-to-SQL performance and efficiency in resource-intensive contexts.

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 $\mathcal{C}'$

3 Problem Definition

We formulate 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 collectively satisfies the informational need expressed by the query q.

A retrieval function f ranks tables in corpus 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 $s(q, t_i)$.

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

4 Adaptive Table Retrieval

ATR adaptively selects the number of tables that each query needs. Whereas standard top-k retrieval always returns a fixed k tables, ATR infers a query-specific number of tables k_q to retrieve:

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

ATR—a transformer encoder—uses the query, the candidate tables, and two special tokens as input. We use the hidden states from each table and special tokens to infer $\hat{\mathcal{C}}_q$. k_q is decided by a comparison of logits from the hidden states between each table and the special token. To effectively capture both query-table and inter-table relevance, we use two complementary objectives: a *relevance calibration* loss that sharpens query-table alignment, and a *semantic grouping* loss that pulls the embeddings of joinable tables closer together. For efficient inference over large table corpora, we propose a sliding-window re-ranking algorithm that refines the ranking without exhaustively scoring every table at once.

4.1 Adaptive Thresholding

ATR learns to separate relevant tables from irrelevant tables for each query through an adaptive-thresholding mechanism inpired by Zhou et al. (2021) We prepend a *threshold token* T_{th} to the input sequence, followed by the natural language query and the candidate tables. Every table begins with a *table token* T_{tab} and is encoded together with its metadata (database, table name, column names).

ATR computes logits $logit_{T_{th}}$ and $logit_{T_{tab}}$ from the hidden states of special tokens T_{th} and T_{tab}, respectively. While training, we enforce that $logit_{T_{tab}}$ is bigger than $logit_{T_{th}}$ when the table is relevant, and is smaller than $logit_{T_{th}}$ otherwise. The loss for this adaptive thresholding is defined 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 single 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 query-irrelevant tables below $logit_{T_{th}}$ by treating the threshold token as their correct class. The threshold logit thus becomes a query-dependent decision boundary that distinguishes relevant tables from irrelevant tables. The hyper-parameters α and β weight the relative contributions of the two loss terms.

4.2 Relevance Calibration

Adaptive thresholding assigns a query-specific cutoff 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:

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

where σ denotes the sigmoid function.

This relevance calibration loss aligns each query with its relevant tables, giving ATR a signal to distinguish them from irrelevant ones and thereby improving its discriminative capability.

4.3 Semantic Grouping

Relationships between tables—especially joinability—are critical for accurate multi-table retrieval (Chen et al., 2024c; Wu et al., 2025). To consider these dependencies between tables, ATR adds a contrastive learning objective (Hadsell et al., 2006) 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 t_i table and let g_i denote its joinability group; the semantic grouping loss L_{SG} is defined as follows:

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

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

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 1: Data statistics. Number of queries (#Q), number of databases (#DB), number of tables (#T), and the number of tables required per query (Min/Max) are reported. Evaluations on Spider and BIRD are performed using development sets.

Finally, the ATR objective function can be 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 allow ATR to adaptively retrieve multiple tables in consideration of the relevance between tables and the relevance between the query and the tables.

4.4 Sliding Window Re-ranking

Since the encoder used in ATR has a quadratic complexity with respect to input length, directly processing large numbers of tables is computationally impractical. To mitigate this inefficiency, ATR uses a *sliding window re-ranking* 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_{tab} 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. Pseudo-algorithm for this sliding window re-ranking appears in Algorithm 1. Since ATR re-ranks subsets of tables within overlapping windows, the method avoids the cost of re-ranking the full list at once.

		Spider				BIRD				Spider 2.0			
		Contriever		UAE		Contriever		UAE		Contriever		UAE	
	k	R	CR	R	CR	R	CR	R	CR	R	CR	R	CR
Bi-Encoder	3	94.0	89.2	93.0	88.0	72.9	55.1	79.0	63.5	37.6	24.8	40.6	27.6
	5	97.4	95.5	97.8	96.4	82.1	68.6	87.2	77.3	45.1	32.6	48.5	34.9
	10	99.2	98.7	99.4	99.0	96.1	92.6	97.6	95.1	56.5	44.8	60.7	49.0
JAR	3	96.5	93.6	96.5	94.1	87.4	76.3	86.3	74.8	42.3	26.9	41.6	27.1
	5	98.5	97.2	99.1	98.5	92.5	85.9	91.1	82.9	48.6	34.0	47.9	33.8
	10	**99.5**	**99.2**	99.5	99.2	97.6	**96.0**	97.2	95.2	55.1	41.4	56.4	43.9
ATR (Ours)		**99.5**	**99.2**	**99.6**	**99.4**	**98.2**	**96.0**	**98.6**	**97.1**	**72.4**	**64.4**	**75.4**	**68.7**

Table 2: Retrieval performance comparison with baseline methods, evaluated using Recall (R) and Complete Recall (CR). ATR consistently outperforms all baselines across datasets. The best and second-best scores for each metric are highlighted in **bold** and underlined, respectively.

5 Experiments

In this section, we introduce the datasets, metrics, models, and baselines used to evaluate the performance of ATR. The experiments aim to validate the effectiveness of ATR that overcomes the inherent limitations of existing table retrieval strategies and to quantify the impact of improved retrieval performance on the downstream task.

5.1 Setups

Dataset We evaluate ATR on three datasets: Spider (Yu et al., 2018), BIRD (Li et al., 2023), and Spider 2.0 (Lei et al., 2025). Spider is a widely used benchmark for text-to-SQL, and BIRD is a realistic text-to-SQL dataset reflecting practical query scenarios. We adopt the "union" setting for these datasets, merging all databases into a single corpus (Kothyari et al., 2023; Chen et al., 2024c; Zhang et al., 2025). Spider 2.0 is a benchmark composed of complex, real-world enterprise text-to-SQL workflows derived from large-scale database systems. Specifically, we use Spider 2.0-Lite[1], a subset featuring multiple SQL dialects, and apply the union setting by grouping databases based on dialect. We denote that ATR training utilizes only the training sets of Spider and BIRD. Dataset statistics are summarized in Table 1, with detailed preprocessing methods provided in Appendix B.

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 $\mathcal{C}$. Retrieving all the ground-truth tables is critical for the table retrieval task, we report *recall* and *complete recall* by comparing $\hat{\mathcal{C}}_q$ with the ground-truth set $\mathcal{C}_q$ following Zhang et al. (2025). In text-to-SQL, 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.

Models We utilize ModernBERT-large (Warner et al., 2024), a bidirectional encoder-only transformer model, as a backbone model for ATR. For table embedding models, we use Contriever (Izacard et al., 2021)[2] and UAE-Large-V1 (Li and Li, 2024).[3] For SQL generation, we utilize Llama-3.1-8B/70B-Instruct (Grattafiori et al., 2024) and Qwen2.5-Coder-7B/32B-Instruct (Hui et al., 2024) as generators.

Baseline We compare ATR with the two bi-encoder baselines—Contriever and UAE-Large-V1—and with the re-ranking method JAR (Chen et al., 2024c), which explicitly encodes table joinability. Contriever and UAE-Large-V1 embed the query and each table independently, flattening the table into text and ranking candidates by cosine similarity between their vector representations. Both ATR and JAR adopt a two-stage re-ranking pipeline: a bi-encoder first retrieves the top 50 ta-

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

[2]https://huggingface.co/facebook/contriever-msmarco

[3]https://huggingface.co/WhereIsAI/UAE-Large-V1

	k	Spider		BIRD		Spider 2.0	
		Llama (8B/70B)	**Qwen (7B/32B)**	**Llama (8B/70B)**	**Qwen (7B/32B)**	**Llama (8B/70B)**	**Qwen (7B/32B)**
Contriever	3	54.6 / 64.8	65.4 / 66.3	21.7 / 40.4	35.4 / 46.0	**1.1** / 3.4	0.5 / 3.9
	5	53.3 / 65.0	65.7 / 68.6	22.4 / 44.5	37.0 / 49.8	**1.1** / 3.2	**1.4** / 3.7
	10	54.6 / 67.1	66.0 / 70.1	24.3 / 47.4	39.1 / 53.0	**1.1** / 3.5	0.9 / 4.1
JAR	3	53.9 / 65.9	66.6 / 68.8	25.4 / 45.3	40.1 / 49.7	0.7 / 3.7	0.9 / 4.1
	5	54.8 / 66.6	67.5 / 69.7	26.8 / 46.4	43.4 / 52.0	**1.1** / 3.9	0.7 / 3.7
	10	56.0 / 66.8	66.6 / 69.2	26.7 / 47.5	43.0 / 53.0	0.9 / 3.0	**1.4** / 4.4
ATR (Ours)		**58.7 / 67.8**	**69.7 / 71.5**	**28.6 / 49.9**	**45.0 / 53.3**	**1.1 / 4.4**	**1.4 / 5.7**
Oracle		66.6 / 70.8	75.6 / 75.2	31.8 / 53.5	50.6 / 58.0	4.4 / 7.2	3.5 / 7.4

Table 3: 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.

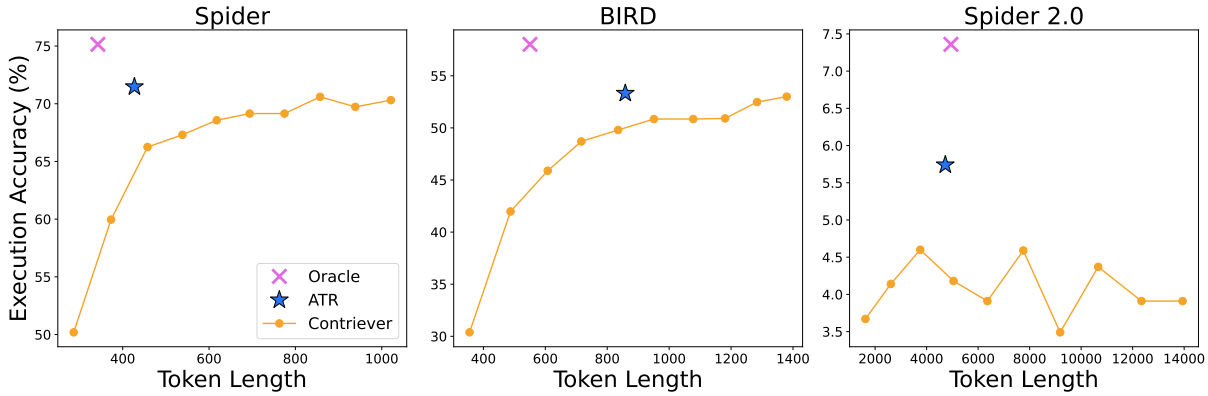

Figure 3: Comparison of execution accuracy and average token length for the text-to-SQL task across different retrieval methods. ATR achieves higher accuracy with fewer tokens compared to the best-performing top-k approach.

bles, and then they are re-ranked using standard approaches (Glass et al., 2022; Sun et al., 2023; Qin et al., 2024).

5.2 Table Retrieval Performance

Across all tested values of k, ATR outperforms the top-k baselines on the in-domain datasets (Spider and BIRD) and the out-of-domain dataset (Spider 2.0) as shown in Table 2. On Spider with $k = 3$, ATR improves complete recall by 10.0% over Contriever and 5.6% over JAR. The performance gains are even larger on BIRD, reaching gains of 40.9% and 19.7% relative to the same baselines. With $k = 10$, ATR still outperforms UAE and JAR on BIRD by 2.0% and 1.9%, respectively. Although no query in Spider or BIRD requires more than four tables and the baselines already use conservative k values, our method still surpasses them. The performance improvement becomes more evident on Spider 2.0, where the number of ground-truth tables varies from one to hundreds. With $k = 10$, ATR improves complete recall by 19.6% over Contriever and 23.0% over JAR. These findings confirm that the top-k retrieval strategy cannot be generalized 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. For both ATR and JAR, we re-rank the 50 candidates retrieved by Contriever.

Table 3 shows execution accuracy across three datasets and several generators. The results demonstrate that ATR achieves substantial improvements over retrieval baselines. With Qwen2.5-Coder-32B and $k = 10$, ATR improves on JAR by 2.3% on

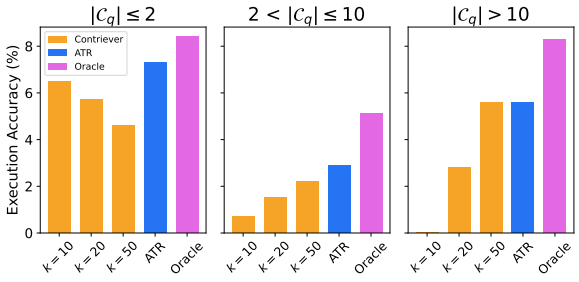

Figure 4: Execution accuracy on the Spider 2.0 dataset across varying numbers of required tables. $|\mathcal{C}_q|$ denotes the number of tables required per query.

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

Table 4: ATR completes retrieval faster than existing adaptive document retrieval strategies and achieves superior performance. Acc. denotes execution accuracy.

Spider and 0.3% on BIRD; the performance gains increase to 3.1% and 2.0%, respectively, when the small 7B model is used. On the more demanding Spider 2.0, ATR surpasses Contriever and JAR by 1.6% and 1.3%. The trend is consistent for Llama-3.1-70B: ATR outperforms JAR by 1.0% on Spider and 2.4% on BIRD with $k = 10$. These results confirm earlier findings that stronger retrieval performance translates into higher downstream accuracy (Kothyari et al., 2023; Chen et al., 2024c).

6 Analysis

This section provides an in-depth analysis of ATR, focusing on token efficiency, robustness to varying numbers of required ground-truth tables, and comparisons with state-of-the-art adaptive document retrieval methods. Additionally, we provide ablation and statistical analyses to empirically validate the effectiveness of table representations.

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 whether ATR retrieves more relevant tables and fewer irrelevant tables than top-k methods, and how this difference 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 3 shows execution accuracy against the average number of input tokens used to generate SQL queries. ATR achieves higher execution accuracy with fewer tokens than the best top-k baseline on all three datasets. Specifically, ATR uses 430 fewer tokens than the best-performing top-k configuration ($k = 8$) 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 upper bound. These results demonstrate that ATR is a robust method that retrieves accurately ground-truth tables while minimizing the retrieval of irrelevant tables.

6.2 Performance Analysis by Number of Required Tables

The top-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 they require: two or fewer, between three and ten, and more than ten.

Figure 4 illustrates the limitations of a fixed k approach. Top-k retrieval achieves its best performance for 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 in accuracy 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 Methods

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.

Table 4 shows that ATR consistently outperforms adaptive document retrieval baselines in both retrieval and end-to-end task. Compared with 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 end-to-end processing time by 6.7 seconds on Spider and 10.6 seconds on BIRD, while achieving comparable execution accuracy. These results demonstrate the effectiveness of ATR through improved table representation learning and the efficiency from operating without iterative retrieval interactions with LLMs.

6.4 Ablation Study

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

As illustrated in Table 5, removing the BCE loss lowers both recall and complete recall, confirming that explicit query-table alignment is crucial for distinguishing relevant tables from irrelevant ones. Similarly, the removal of contrastive loss reduces retrieval performance, indicating that inter-table joinability enhances the discriminative quality of table embeddings, which contributes to improved retrieval accuracy. These findings indicate that both BCE and contrastive loss components are essential for learning robust table representations, thereby contributing to superior retrieval capabilities of ATR compared to existing table retrieval methods.

6.5 Statistical Analysis of Table Representations

ATR assigns logits on tokens T_{th} for the threshold and T_{tab} 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	
	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 training strategies for table representation. Both loss functions contribute significantly to the retrieval performance of ATR.

7 Conclusion

In this work, we address the limitations of conventional table retrieval methods that rely on a top-k retrieval strategy. Such rigidity 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 query requirements. ATR leverages threshold embedding and table-level embedding to determine the optimal number of tables required for each query. Furthermore, ATR adopts relevance calibration loss 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, improved downstream accuracy, and enhanced inference efficiency. These results confirm ATR as a practical solution suitable for large-scale database retrieval applications.

Limitations

Although ATR demonstrates substantial improvements in both retrieval and downstream execution accuracy, two limitations remain. First, the sliding window re-ranking method effectively reduces computational complexity, but determining optimal window sizes and retention parameters may necessitate additional empirical tuning across different datasets or retrieval scenarios. Second, 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, by establishing a robust framework for adaptively retrieving relevant information efficiently without direct interaction with LLMs, our methodology provides a strong foundation for future studies aiming to generalize retrieval-augmented generation across diverse data types and broader application contexts.

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*.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

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

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

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, et al. 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.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 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.

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, 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, 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. 2025. 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.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822, Toronto, Canada. Association for Computational Linguistics.

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.

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.

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.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang,

Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. mGTE: Generalized long-context text representation and reranking models for multilingual text retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1393–1412. 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.

A Environment Setup

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.

B Dataset 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 dense 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.

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

		Spider		BIRD		Spider 2.0	
	k	R	CR	R	CR	R	CR
mGTE	5	94.8	89.9	88.5	76.8	44.3	30.1
	10	98.9	97.5	96.5	92.5	57.0	42.0
BGE-M3	5	96.7	93.7	78.9	64.2	40.4	26.8
	10	99.1	98.1	91.3	83.6	52.8	40.6
RankGPT	5	98.9	97.9	92.7	84.6	56.1	42.1
	10	**99.6**	**99.5**	97.5	94.6	64.0	52.2
RankZephyr	5	98.2	96.9	84.4	72.4	46.7	33.8
	10	99.5	99.2	96.5	93.1	57.3	45.5
ATR (Ours)		99.5	99.2	**98.2**	**96.0**	**72.4**	**64.5**

Table 7: Comparison of retrieval performance between ATR and text-based re-ranking methods. Candidate tables are initially retrieved using Contriever.

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 illustrated in Table 6. 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 strong retrieval performance when combined with advanced embedding models.

D Comparison with Text Re-rankers

In this section, we evaluate ATR's retrieval performance by comparing it against widely used text re-ranking methods. For cross-encoder-based re-ranking, we adopt mGTE-reranker (Zhang et al., 2024)[4] and BGE-M3 (Chen et al., 2024a)[5]. For LLM-based re-ranking, we leverage RankGPT (Sun et al., 2023) and RankZephyr (Pradeep et al., 2023)[6]. We use

[4] https://huggingface.co/Alibaba-NLP/gte-multilingual-reranker-base

[5] https://huggingface.co/BAAI/bge-reranker-v2-m3

[6] https://huggingface.co/castorini/rank_zephyr_7b_v1_full

gpt-4o-mini-2024-07-18 for RankGPT.

As shown in Table 7, ATR consistently achieves superior retrieval performance compared to cross-encoder re-rankers and LLM-based re-rankers. Specifically, ATR achieves improvement of 1.7% on Spider and 3.5% on BIRD in complete recall, compared to mGTE-reranker at $k = 10$. This performance gap widens on Spider 2.0, where ATR achieves a 19.0% higher complete recall compared to RankZephyr at $k = 10$. Given that ATR is substantially smaller than RankZephyr, these results demonstrate the capability of ATR to overcome the limitations of top-k retrieval methods and effectively learn robust table representations.

E 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.

As illustrated in Table 8, 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 9, when the query necessitates more tables than the k, top-k retrieval fails to retrieve all essential tables, again resulting in incorrect SQL outputs. In contrast, ATR adaptively retrieves an appropriate table 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.

F Prompt Template

We vary prompt templates based on the dataset and SQL dialect when performing text-to-SQL tasks. Fig. 5 shows the prompt templates used for the Spider and BIRD datasets. Fig. 6 to Fig. 8 are prompt templates used for the Spider 2.0 dataset.

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 8: 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 9: 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.

```
[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 5: 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 6: 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 7: 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 8: Prompt template for Spider 2.0 (SQLite dialect)