Multi-Hop Table Retrieval for Open-Domain Text-to-SQL

Anonymous ACL submission

Abstract

The open-domain text-to-SQL task aims to retrieve question-relevant tables from massive databases and then generate SQL. However, the performance of current methods is limited by single-hop retrieval, and existing multi-hop retrieval of other tasks cannot be directly applied due to two primary challenges: error cascades and the tendency to retrieve tables similar to the retrieved ones but irrelevant to the question. Therefore, we propose the multi-hop table retrieval with rewrite and beam search (MURRE). To reduce error cascades, MURRE employs beam search to select multiple tables at each hop. To avoid retrieving similar but irrelevant tables, we remove the retrieved information from the question, guiding the retriever to focus on unretrieved tables. We conduct experiments on two open-domain text-to-SQL datasets, achieving an average improvement of 5.7% over the previous state-of-the-art results.[1]

1 Introduction

Text-to-SQL, a significant task of natural language processing, reduces the difficulty of accessing databases and aids in efficient data querying, which has broad applications across many fields (Qin et al., 2022). Unlike the previous text-to-SQL task that provides the question-relevant database tables[2], a more realistic scenario is open-domain text-to-SQL, which should handle vast amounts of databases to convert user questions into SQL. Specifically, the open-domain text-to-SQL task involves two main steps: retrieving relevant tables and generating SQL based on the retrieved tables and the user question (Kothyari et al., 2023).

Considering the extensive knowledge embedded in LLM parameters, CRUSH (Kothyari et al., 2023) bridges the semantic gap between natural language user questions and structured database tables by rewriting user questions into potentially relevant tables. However, CRUSH performs table retrieval in a single hop, limiting its performance because the retrieval of certain tables could depend on others, similar to multi-hop retrieval in the open-domain question answering (QA) task (Feldman and El-Yaniv, 2019; Xiong et al., 2021). Nevertheless, adapting previous multi-hop retrieval methods to the open-domain text-to-SQL task presents two significant challenges, as illustrated in Figure 1.

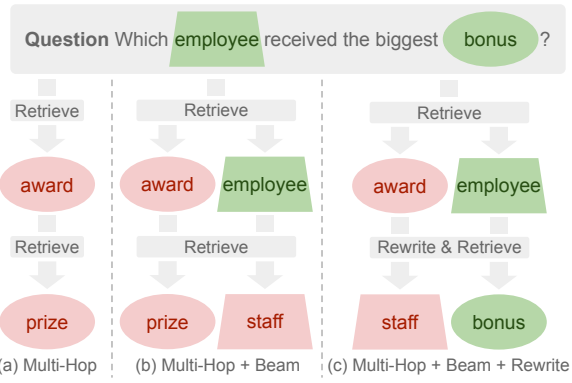

Figure 1: The two challenges of existing methods. The same shapes denote similar tables. The green represents the relevant tables, and the red represents the irrelevant. Multi-hop retrieval (a) gets irrelevant tables in the first hop, resulting in error cascades. Multi-hop retrieval employing the beam search paradigm (b) retrieves tables in the second hop similar to the retrieved tables in the first hop but irrelevant to the question.

First, the retriever could retrieve irrelevant tables, leading to ***error cascades*** in subsequent retrievals, as shown in Figure 1(a). Second, the retriever could easily ***retrieve tables that are similar to those retrieved in previous hops but irrelevant to the question***, as illustrated in the right part of Figure 1(b). This occurs because most multi-hop methods add the retrieved documents into the user question as the supplementary (Lee et al., 2022; Shao et al., 2023). Conversely, in open-domain text-to-SQL, the user question typically contains all necessary

[1] Our code and data are publicly available upon acceptance.

[2] For brevity, we refer to database tables relevant to the question as **relevant tables** in this paper.

information for retrieval with no need for the supplementary. Consequently, adding retrieved tables to the question can cause retrieved tables similar to those already retrieved but irrelevant to the question, resulting in limited retrieval performance. For instance, in the right part of Figure 1(b), the irrelevant table "*staff*" is retrieved because it is similar to the table "*employee*" retrieved in the first hop.

To solve the above challenges, we propose a method called **MU**lti-hop table **R**etrieval with **R**ewrite and b**E**am search (MURRE) to enhance the retrieval performance for the open-domain text-to-SQL. As illustrated in Figure 1(c), our method retrieves tables through the multi-hop retrieval, adopting the beam search paradigm to maintain multiple retrievals per hop, inspired by Zhang et al. (2024). To address the error cascades challenge, we employ the beam search paradigm to consider multiple possible tables at each hop, mitigating the impact of retrieving irrelevant tables in previous hops. For the second challenge, we prompt LLMs to remove the retrieved table information from the user question, forming a new question for the next hop, which reduces the retrieval of tables similar to the previously retrieved ones but irrelevant.

To validate the effectiveness of MURRE, we conduct experiments on two datasets, SpiderUnion and BirdUnion, which are open-domain versions of text-to-SQL datasets Spider (Yu et al., 2018) and Bird (Li et al., 2023b). MURRE achieves an average improvement of 5.7% compared to the previous state-of-the-art (SOTA) results, demonstrating its effectiveness. Additionally, our case studies indicate that MURRE enhances the performance of the open-domain text-to-SQL by mitigating the two challenges mentioned above.

Our contributions are as follows:

- To mitigate the impact of error cascades in multi-hop retrieval, we propose utilizing the beam search paradigm, which reduces the impact of retrieving question-irrelevant tables.
- To address the challenge of retrieving tables that are similar to previously retrieved ones but are irrelevant to the question, we propose removing the retrieved information from the user question, guiding the retriever to find new relevant tables.
- To demonstrate the effectiveness of MURRE, we conduct experiments on the SpiderUnion and BirdUnion datasets, achieving an average improvement of 5.7% compared with the previous SOTA results, proving its effectiveness.

2 Methodology

2.1 Task Definition

Our work mainly focuses on the open-domain text-to-SQL task, which can be formally defined as: Given a user question q, database tables $T = \{t_i\}$ and a number of retrieved tables N, suppose the tables relevant to q are $T^q = \{t_i^q\}$, MURRE aims to retrieve N tables T_N^q, where $T^q \subseteq T_N^q$.

2.2 Overview

The overview of MURRE is illustrated in Figure 2. MURRE comprises multiple hops, with each hop consisting of several beams. Each hop can be divided into two phases: **Retrieve** (§2.3) and **Rewrite** (§2.4). In each hop, we first **retrieve** the relevant tables by calculating their probability that is relevant to the question. Next, using the original user question and the retrieved tables in each beam, we **rewrite** the user question to exclude the information of the retrieved tables, generating the user question used for the subsequent hop. MURRE repeats the Retrieve and Rewrite phases until reaching the maximum hop limit H or meeting the early stop condition (see §2.4). After the multi-hop retrieval, we **score** (§2.5) each table based on its probability that is relevant to the user question and select the top-N tables as the input for generating the SQL. In Appendix A, we explore how to handle the situation when no table in the database is relevant to the user question.

2.3 Retrieve

The Retrieve phase aims to identify B tables corresponding to the user question $q^{h,b}$, where B is the beam size and $q^{h,b}$ denotes the user question at hop h and beam b. We embed the user question and each table into vectors and then compute their relevance probability to the user question. Let $\texttt{Emb}(x)$ represent the embedding vector of x, the probability is expressed as Equation 2.1.

$$\hat{P}(t_i|q^{h,b}) = \texttt{Norm}(\frac{\texttt{Emb}(t_i) \cdot \texttt{Emb}(q^{h,b})}{|\texttt{Emb}(t_i)||\texttt{Emb}(q^{h,b})|}) \quad (2.1)$$

We use the cosine similarity between the question and table vectors to calculate the probability, adopting `Norm` to ensure the conditional probabilities sum to one. Detailed representations of the table and the normalization method are provided in Appendix B and Appendix C, respectively. We select the B tables with the highest probabilities as the retrieval results for the current hop question $q^{h,b}$.

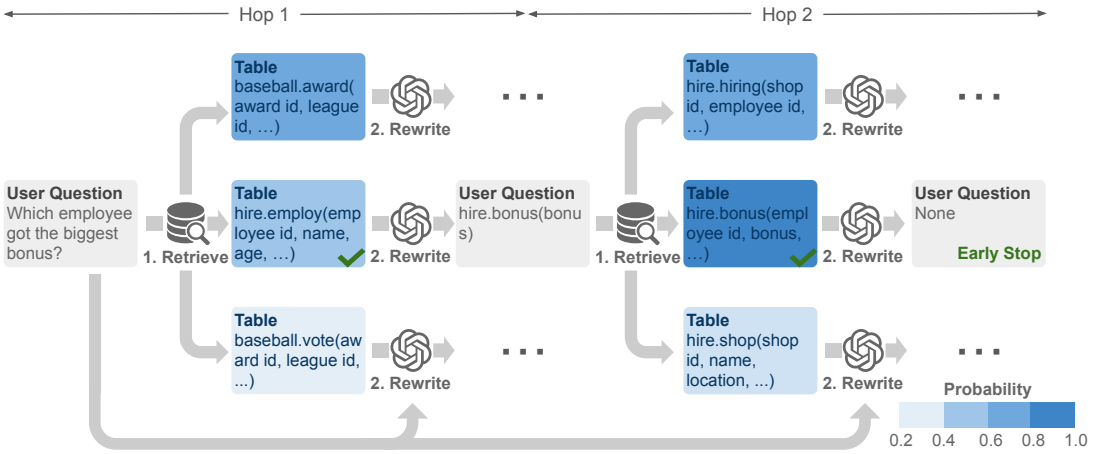

Figure 2: An overview of MURRE. Each hop consists of: (*i*) **Retrieve** tables similar to the question; (*ii*) **Rewrite** the question by removing the retrieved table information from the user question and representing the unretrieved information in the form of a table with LLM. We employ the beam search paradigm maintaining multiple retrievals at each hop. The color depth represents the probability that the table is relevant to the question of the hop, and ✓ represents the relevant table. We demonstrate an example which early stops at hop 2 for brevity.

In the previous hop of retrieval, there are B beams and each beam corresponds to B retrieved tables obtained at the current hop, resulting in a total of $B \times B$ retrieval results. Following the beam search paradigm, we then choose B results from these for the subsequent rewriting phase, with the selection method detailed in §2.5.

2.4 Rewrite

The Rewrite phase is designed to mitigate the retrieval of tables that are similar to previously retrieved ones yet irrelevant to the user question. To achieve the above goal, this phase employs LLMs to rewrite the user question, by removing the information of retrieved tables in $\texttt{Path}^{h,b}$ (which includes all retrieved tables from hop 1 to hop h on the path of beam b) from the user question q, and express the unretrieved information in the form of a table following the previous work (Kothyari et al., 2023), where the rewritten question is used to guide the retriever for the next hop.

Considering that different user questions require varying numbers of tables, to prevent additional hops from introducing errors, we instruct LLMs to assess whether the retrieved tables in $\texttt{Path}^{h,b}$ are sufficient to answer q, i.e., early stop. In detail, we prompt the LLM to generate a special mark (e.g., "None" in Figure 2) during rewriting to indicate that the retrieved tables are sufficient to answer the user question q, ceasing further retrieval as soon as this early stop mark is produced. The prompts used for rewriting are detailed in Appendix B.

2.5 Score

The aforementioned multi-hop retrieval process maintains B paths at each hop, containing tables that could exceed or fall short of the required number of tables N. Therefore, this phase aims to score all the tables based on their probabilities to retain the most relevant tables used for generating SQL, which consists of two parts: (*i*) scoring the retrieval path $\texttt{Score\_Path}(\texttt{Path}^{h,b})$; (*ii*) scoring the retrieved table $\texttt{Score\_Table}(t_i)$.

Retrieval Path Score First, we address the calculation of the retrieval path score, where each node on the path corresponds to the retrieved table on a beam of one hop. As discussed in §2.3, the score of each retrieval path represents the probability that the last table in the path is retrieved, given its corresponding question at the last hop. Following the derivation in Appendix D, the score of a retrieval path is computed as the product of all the probabilities $\hat{P}$ in the path.

Table Score Building on the retrieval path score, we describe the calculation to score a retrieved table t_i. Since each table has multiple scores across various hops and beam retrievals, we propose a table scoring algorithm to effectively integrate these scores. Detailed information is provided in Appendix E. Considering the potential interrelation of question-relevant tables, we aim to ensure that all retrieved tables collectively are the most relevant to the question. Thus, the higher the retrieval path score, the higher the score of the tables in the path.

Dataset	#table				
	1	2	3	4	All
SpiderUnion	395	214	43	6	658
BirdUnion	364	943	207	20	1534

Table 1: Statistics on the number of the relevant table for each question in the SpiderUnion and BirdUnion. **#table** denotes the number of the relevant table. **All** refers to the total number of questions in the dataset.

Specifically, let $\texttt{Path}_{t_i}$ denote all retrieval paths containing table t_i, we calculate the table score as: $\texttt{Score\_Table}(t_i) = \max_{t \in Path_{t_i}} \texttt{Score\_Path}(t)$. Finally, we select $T_N^Q = \{t_1, ..., t_N\}$ with the highest $\texttt{Score\_Table}(t_i)$ as our retrieval results. Appendix F elaborates on enhancing the table scoring algorithm to address ambiguous entities or synonyms in user questions or tables.

3 Experiments

3.1 Experiment Setup

Dataset To evaluate the effectiveness of MURRE, we validate it on two open-domain text-to-SQL datasets: SpiderUnion (Kothyari et al., 2023) and BirdUnion, which mix all tables of Spider (Yu et al., 2018) and Bird (Li et al., 2023b). Table 1 shows the number of questions requiring different numbers of tables. Detailed descriptions of Spider and Bird are provided in Appendix G.

Metric We employ recall and complete recall as evaluation metrics for retrieval, and Execution Accuracy (EX) (Yu et al., 2018) for text-to-SQL. Recall (***r*@**) measures the proportion of relevant tables retrieved from all relevant tables, following previous work (Kothyari et al., 2023). Unlike other open-domain tasks (e.g., open-domain QA), it is crucial to retrieve all relevant tables to generate correct SQL for the open-domain text-to-SQL. Hence, we introduce complete recall (***k* =**), which is the proportion of examples retrieving all relevant tables. For text-to-SQL, we use execution match (EX), following previous work (Gao et al., 2023a), to evaluate the correctness of the execution results of predicted SQL compared to gold SQL.

Model We utilize SGPT (Muennighoff, 2022) to embed tables and user questions without additional fine-tuning, following the previous work (Kothyari et al., 2023). For the Rewrite phase and SQL generation, we use `gpt-3.5-turbo`[3] to predict. The detailed descriptions of SGPT and `gpt-3.5-turbo` are provided in Appendix H.

Comparing System In our experiments, we compare MURRE with the following methods: (*i*) Single-hop, which retrieves with the user question in a single hop; (*ii*) CRUSH (Kothyari et al., 2023), which retrieves in a single hop with hallucinating the user question into the table format.

Implement Details We set the beam size to 5, as it provides the best performance with the smallest size (see §3.4.1). The maximum hop (abbreviated as max hop) is set to 3 because over 98% of questions in the SpiderUnion and BirdUnion datasets require $\leq$ 3 tables (see Table 1). We rewrite user questions using the 9-shot prompt and 8-shot prompt on the SpiderUnion and BirdUnion respectively, since the table scales of BirdUnion are larger than that of SpiderUnion.

3.2 Main Result

The main results of our experiments are presented in Table 2. Compared to CRUSH, MURRE demonstrates significant improvements across various datasets and models of different scales, with an average enhancement of 5.7% in recall and complete recall compared to the previous SOTA, which validates the effectiveness of our method. From the table, we can also see that:

The improvement of MURRE on BirdUnion is more significant than on SpiderUnion. Since the questions in BirdUnion typically require more tables (see Table 1), requiring multi-hop retrieval of MURRE more to obtain multiple relevant tables, thereby enhancing retrieval performance.

As the number of top-ranked tables grows, the retrieval performance of MURRE slows down. Improving metrics with a high number of top-ranked tables necessitates retrieving relevant tables that are highly dissimilar to the user question, making the metrics challenging to enhance. Especially, for some metrics (e.g., $k = 20$, r@20), the performance of MURRE declines slightly because removing retrieved table information at each hop leads to a greater focus on retrieving tables that are highly dissimilar to the user question.

Text-to-SQL Experiments We conduct text-to-SQL experiments on the SpiderUnion and BirdUnion datasets using the user question and retrieved tables as the input, as shown in Table 3.

[3] Document for `gpt-3.5-turbo`

Dataset	Model	Method	$k=3$	$k=5$	$k=10$	$k=20$	r@3	r@5	r@10	r@20
SpiderUnion	SGPT-125M	Single-hop	54.3	66.0	75.4	82.2	63.0	73.1	80.7	86.3
		CRUSH†	60.2	71.3	80.7	**86.8**	68.9	76.3	**83.4**	**88.9**
		MURRE	**65.0**	**74.2**	**81.0**	85.3	**70.2**	**77.5**	82.3	86.9
	SGPT-5.8B	Single-hop	76.3	86.8	94.1	**97.6**	84.0	91.5	96.2	**98.7**
		CRUSH†	68.2	80.1	88.4	92.2	75.5	85.1	91.2	94.5
		MURRE	**86.0**	**93.5**	**96.7**	97.3	**89.3**	**94.3**	**96.8**	97.5
BirdUnion	SGPT-125M	Single-hop	39.0	50.3	62.1	70.9	54.0	63.2	73.3	80.9
		CRUSH†	42.1	56.1	70.2	77.7	60.2	70.0	79.5	**86.1**
		MURRE	**51.4**	**62.7**	**72.9**	**78.3**	**64.8**	**72.7**	**79.6**	84.2
	SGPT-5.8B	Single-hop	55.3	67.3	79.4	86.4	72.9	80.8	88.6	92.8
		CRUSH†	52.2	63.5	78.4	88.1	70.0	77.9	87.5	93.0
		MURRE	**69.1**	**80.1**	**88.7**	**92.7**	**81.0**	**87.6**	**92.6**	**95.4**

Table 2: The main results on complete recall and recall of MURRE, compared with Single-hop and CRUSH on SpiderUnion and BirdUnion, using SGPT-125M and SGPT-5.8B as the embedding models. k refers to the complete recall, and r refers to the recall. † denotes our run since the performance difference led by the API change. The best results of different datasets and models are annotated in **bold**.

Model	Method	SpiderUnion r@3	SpiderUnion r@5	SpiderUnion r@10	SpiderUnion r@20	BirdUnion r@3	BirdUnion r@5	BirdUnion r@10	BirdUnion r@20
SGPT-125M	Single-hop	43.9	50.0	53.2	54.1	11.2	13.4	17.1	18.5
	CRUSH	47.3	50.9	**55.9**	**59.6**	14.5	**17.1**	19.0	**20.5**
	MURRE	**50.3**	**54.1**	54.7	57.4	**16.0**	16.8	**19.4**	20.0
SGPT-5.8B	Single-hop	54.9	60.6	61.6	63.7	16.9	17.3	18.3	20.4
	CRUSH	49.8	56.4	60.3	60.8	16.8	18.0	19.7	21.1
	MURRE	**62.5**	**64.4**	**64.4**	**66.7**	**20.9**	**21.8**	**22.0**	**22.4**

Table 3: EX for predicted SQL based on the input, including the user question and varying numbers of retrieved tables on SpiderUnion and BirdUnion using SGPT-125M and SGPT-5.8B embeddings, compared with Sigle-hop and CRUSH. The best results with different models are annotated in **bold**.

The performance of MURRE in text-to-SQL surpasses both Single-hop and CRUSH, exhibiting a trend similar to the retrieval performance, thereby further validating the effectiveness of our retrieval method. As the number of input tables increases, the EX improvement decelerates beyond the top 10 tables due to the presence of too many irrelevant tables, which hinders the model from focusing on the relevant ones. This also underscores the necessity of MURRE in enhancing retrieval performance on a small number of top-ranked tables in the open-domain text-to-SQL task.

3.3 Ablation Studies

To demonstrate the effectiveness of our method, we conduct ablation experiments on SpiderUnion, with results presented in Table 4. We select SpiderUnion corresponding to Spider to perform subsequent experiments because Spider is the mainstream dataset for the text-to-SQL task. Since SGPT-125M and SGPT-5.8B exhibit similar trends across different datasets and methods (as shown in Tables 2 and 3), we select SGPT-125M as the embedding model employed in the subsequent experiments to balance the embedding speed and the retrieval recall (Muennighoff et al., 2023).

The Effectiveness of Rewrite To demonstrate the effectiveness of Rewrite which removes the retrieved table information from the user question, we compare its performance against the standard multi-hop retrieval method in open-domain QA, which is directly splicing the user question with retrieved tables at each hop without rewriting. Compared with MURRE, the performance of splicing methods drops significantly and consistently, underscoring the effectiveness of Rewrite which alleviates retrieved tables similar to previously retrieved tables but irrelevant to the user question.

The Effectiveness of Tabulation To prove the effectiveness of transforming questions into a tabular format (abbreviated as tabulation), we rewrite the questions at each hop into natural language questions targeting to query unretrieved information.

Method	$k=3$	$k=5$	$k=10$	r@3	r@5	r@10
MURRE	**65.0**	**74.2**	**81.0**	**70.2**	**77.5**	**82.3**
w/o rewrite	46.2 (−18.8)	56.7 (−17.5)	67.2 (−13.8)	50.6 (−19.6)	60.7 (−16.8)	70.0 (−11.6)
w/o tabulation	54.6 (−10.4)	64.9 (−9.3)	75.5 (−5.5)	63.4 (−6.8)	72.5 (−5.0)	80.9 (−1.4)
w/o early stop	52.6 (−12.4)	64.9 (−9.3)	71.0 (−10.0)	57.1 (−13.1)	67.0 (−10.5)	72.2 (−10.1)

Table 4: The ablation results on evaluating MURRE, compared with splicing the question and previously retrieved tables (denoted as *w/o rewrite*), rewriting to natural language question (denoted as *w/o tabulation*), and without employing the mechanism of early stop (denoted as *w/o early stop*) on SpiderUnion with SGPT-125M. k refers to the complete recall, and r refers to the recall. The best results are annotated in **bold**.

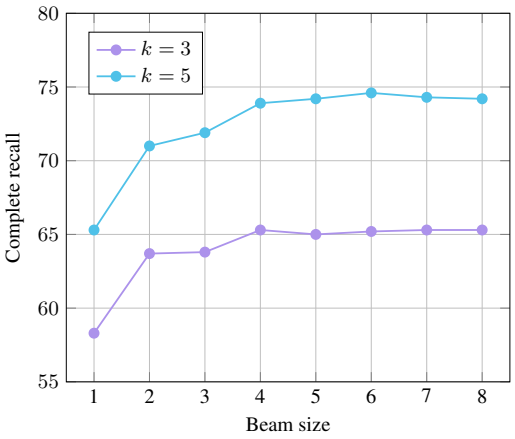

Figure 3: The complete recall with different beam sizes on SpiderUnion with SGPT-125M.

Max Hop	#table 1	2	3	≥ 4	All
1	73.7	59.8	25.6	**50.0**	66.0
2	73.2	77.6	**58.1**	**50.0**	73.4
3	**74.2**	**78.0**	**58.1**	**50.0**	**74.2**
4	**74.2**	**78.0**	**58.1**	**50.0**	**74.2**

Table 5: Complete recall $k=5$ of MURRE with varying maximum hops. We categorize the SpiderUnion dataset based on the number of relevant tables (denoted as **#table**) per question. **All** represents the undivided SpiderUnion dataset. The best results with different tables are annotated in **bold**.

The results indicate that, compared to rewriting to natural language, rewriting to tabulated questions significantly enhances performance, validating the effectiveness of tabulation in MURRE.

The Effectiveness of Early Stop To verify the effectiveness of early stop in MURRE, we compare the results without using this mechanism, where the model does not generate the special early stop mark. The performance without the early stop is significantly degraded, which proves that incorporating the early stop mechanism in MURRE effectively guarantees the performance.

3.4 Analysis

In the analysis experiments, we study how different parameters affect the MURRE performance. We use $k=5$ as the evaluation metric, with detailed explanations provided in Appendix I. Additionally, we discuss the the efficiency of MURRE in Appendix J and the impact of SQL hardness on the performance in Appendix K.

3.4.1 How Does Beam Size Affect the Performance?

To observe the impact of different beam sizes on the retrieval performance, we compare the performance of our method using SGPT-125M as the embedding on the SpiderUnion dataset under the setting of different beam sizes, as shown in Figure 3. When our method does not employ beam search, i.e., with a beam size of 1, performance degrades rapidly, indicating that beam search mitigates the effects of error cascade. As the beam size increases, the complete recall shows a significant upward trend until reaching a beam size of 5, beyond which performance either improves slightly or declines. This indicates that while employing beam search enhances the performance of our method, a beam size greater than 5 introduces too many irrelevant tables, resulting in higher computational costs without further performance gains.

3.4.2 How Does the Number of Hops Affect the Performance?

To verify the effectiveness of multi-hop in our method, we conduct experiments on the SpiderUnion dataset, divided based on the number of relevant tables, using SGPT-125M as the embedding. We compare the complete recall $k=5$ with varying numbers of maximum hops, as shown in Table 5. The results indicate the following: (*i*) The overall trend indicates that MURRE achieves the best performance when the number of maximum hops is greater than or equal to the required number of tables. (*ii*) For questions requiring 1 or 2 tables,

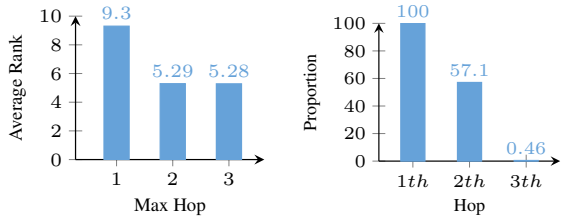

Figure 4: The left part is the average rank of relevant tables with different numbers of max hops on the SpiderUnion with MURRE. The right part is the proportion of questions that are not early stopped with different hops on the SpiderUnion with MURRE.

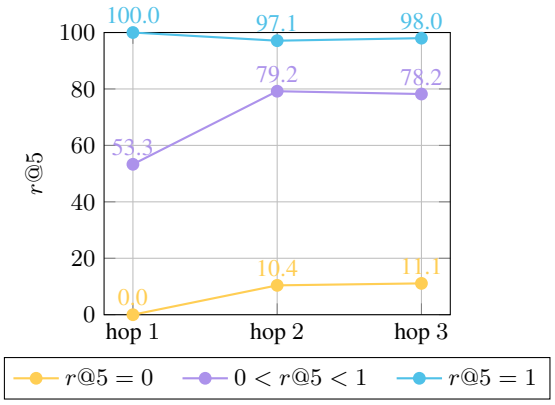

Figure 5: The $r@5$ during multiple hops, categorizing according to the $r@5$ in the first hop which falls into different intervals, demonstrates that the cascading effect of our method is not significant.

the best performance is achieved at a maximum hop of 3. Since MURRE can improve the performance of the questions that retrieve irrelevant tables in previous hops by removing the retrieved information, guiding the model to obtain unretrieved relevant tables in subsequent hops. (*iii*) The performance for questions requiring a 1 table slightly decreases with 2 hops because removing retrieved information from such questions could easily introduce errors. However, this error is mitigated and eliminated at a maximum hop of 3. (*iv*) The performance of requiring ≥ 4 tables remains unchanged across multiple hops because improving complete recall $k = 5$ necessitates that the top 5 retrieved tables include all relevant tables, which is challenging.

3.4.3 Can MURRE Reduce the Average Rank of Relevant Tables?

To verify the effectiveness of MURRE in improving the average rank of relevant tables, we calculate the average rank at different maximum hops, as shown on the left part of Figure 4. From the figure, we can see that MURRE significantly enhances the average rank of relevant tables, with the most notable improvement occurring at a maximum hop of 2. This is because most questions in SpiderUnion require 1 or 2 tables (see Table 1), requiring two hops to obtain unretrieved relevant tables. Conversely, the improvement at a maximum hop of 3 is weak, not only because of the limited number of questions requiring ≥ 3 tables but also due to the early stop mechanism, which causes most questions to cease retrieval before the third hop.

As illustrated in the right part of Figure 4, the proportion of different hops performed by our method is almost the same as the proportion of the table corresponding to the different table numbers in Table 1. Besides, most user questions have stopped retrieving in the third hop, so it is reasonable to set the maximum hop steps to 3.

3.4.4 How Does the Previous Errors Affect Subsequent Performance?

To examine the error cascades in MURRE, we analyze the number of errors with different recalls that occurred in different hops. We compare the performance of $r@5$ during multiple hops, categorizing the questions into $r@5 = 0$, $0 < r@5 < 1$, and $r@5 = 1$ based on the results in the first hop, as illustrated in Figure 5. We use SGPT-125M as the embedding model and conduct experiments on the SpiderUnion dataset, with a beam size of 5 and a maximum hop of 3.

The error cascades in MURRE is minimal and is significantly compensated by the performance improvements brought about by multi-hop retrieval. We analyze these three categories of questions separately. (*i*) For questions whose $r@5 = 0$ in hop 1, the top 5 tables retrieved are all irrelevant. MURRE improves performance because removing the retrieved information from the user question can eliminate the interference of irrelevant tables retrieved in the previous hops. (*ii*) For questions with $0 < r@5 < 1$ in hop 1, there are irrelevant tables among the 5 retained tables. Our method significantly enhances performance in hop 2 by extracting the retrieved table information from the user question and focusing on retrieving unretrieved relevant tables. Performance in hop 3 is slightly reduced, mainly because most questions in SpiderUnion have ≤ 2 relevant tables (see Table 1), and additional hops may introduce errors. (*iii*) For questions whose $r@5 = 1$ in hop 1, performance in hops 2 and 3 is slightly reduced as all relevant tables are retrieved, causing subsequent hops to introduce a small amount of error.

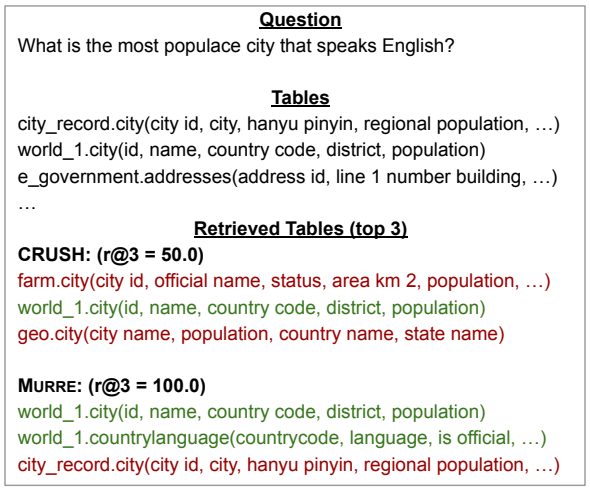

Figure 6: A case study comparing MURRE with CRUSH. Green indicates relevant tables, while red indicates irrelevant ones. Each table is represented as "database name.table name(column names)". r denotes recall.

3.5 Case Study

We demonstrate a case study with MURRE compared with CRUSH, as shown in Figure 6. We can see that CRUSH fails to retrieve the table *"world_1.countrylanguage"* within the top 3 results due to its single-hop retrieval limitation, as the retrieval of table *"world_1.countrylanguage"* relies on the table *"world_1.city"*. In contrast, MURRE employs multi-hop retrieval with a beam size of 3, increasing the probability of selecting relevant tables at each hop. Additionally, we eliminate the retrieved information in *"world_1.city"* from the question, which aids the model in retrieving *"world_1.countrylanguage"* that was previously missed. A detailed comparison with additional methods is provided in Appendix L.

4 Related Work

4.1 Text-to-SQL

The text-to-SQL task aims to convert user questions into SQL queries, facilitating efficient database access (Qin et al., 2022). LLM-based methods have become mainstream in text-to-SQL due to their superior performance with minimal annotated data (Li et al., 2023a; Gao et al., 2023a). For example, Li and Xie (2024) propose creating test cases and using LLMs to predict execution results, determining the correctness of SQL from candidates. However, these methods do not focus on open-domain text-to-SQL and exist a gap with real-world applications. Therefore, CRUSH (Kothyari et al., 2023) proposes to guess potentially relevant tables for retrieval. DBCopilot (Wang et al., 2024) trains a schema router to identify relevant tables. Chen et al. (2024b) propose a re-ranking relevance method by fine-tuning DTR models.

However, existing methods are constrained by: *(i)* only designing to retrieve with the single hop; *(ii)* requiring fine-tuning, which is resource-intensive and domain-specific. To solve these problems, we propose a multi-hop table retrieval method for open-domain text-to-SQL.

4.2 Retrieval for Open-Domain QA

Existing retrieval methods for open-domain QA leverage in-context learning and the knowledge embedded in LLM parameters, demonstrating effectiveness across many benchmarks (Gao et al., 2023b; Chen et al., 2024a). Some studies emphasize iterative retrieval and generation, enhancing the performance by using multi-hop retrieval. For example, ITER-RETGEN (Shao et al., 2023) proposes to splice the question and LLM generation to retrieve for the next iteration. To reduce the calculation cost of multi-hop retrieval, Adaptive-RAG (Jeong et al., 2024) proposes to adopt multi-hop retrieval by splicing the question, retrieved documents, and generated answers in each hop for complex questions that are predicted first.

However, these methods are unsuitable for open-domain text-to-SQL due to error cascades introduced by multi-hop retrieval. Also, unlike the user questions in open-domain QA that require supplementary documents, adding retrieved tables to our questions tends to retrieve similar tables among hops. To solve these challenges, we select multiple tables at each hop and exclude the retrieved information from the question for next-hop retrieval.

5 Conclusion

In the paper, we propose MURRE to address the challenge that multi-hop retrieval in other open-domain tasks cannot be directly applied to open-domain text-to-SQL. Compared with previous methods, MURRE employs the beam search paradigm to reduce the impact of error cascades in multi-hop retrieval and removes the retrieved information from the question at each hop to obtain unretrieved tables. Experimental results demonstrate the effectiveness of MURRE on two open-domain text-to-SQL datasets. Our method achieves new SOTA results compared with the previous methods, with an average of 5.7% improvement.

Limitations

We discuss the limitations of our work from the following two aspects. (*i*) Considering the applicability, the multi-turn text-to-SQL task is common in real scenarios (Yu et al., 2019a,b), while we do not discuss the solutions of open-domain multi-turn text-to-SQL. We leave improving our method to apply to multi-turn text-to-SQL for future work. (*ii*) From the performance perspective, our method does not consider the performance improvement brought by the text-to-SQL feedback (Trivedi et al., 2023; Yu et al., 2023). We leave the retrieval recall improvement leveraging the results of text-to-SQL for future work. handles ambiguous entities or synonyms within the natural language questions or database schemas. Although our method achieves significant improvements, future work can improve our method from the aspects of applicability and recall further.

Ethics Statement

Every dataset and model used in the paper is accessible to the public, and our application of them adheres to their respective licenses and conditions.

References

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024a. Benchmarking large language models in retrieval-augmented generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17754–17762.

Peter Baile Chen, Yi Zhang, and Dan Roth. 2024b. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. *Preprint*, arXiv:2404.09889.

Yair Feldman and Ran El-Yaniv. 2019. Multi-hop paragraph retrieval for open-domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2296–2309, Florence, Italy. Association for Computational Linguistics.

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

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023b. Retrieval-augmented generation for large language models: A survey. *ArXiv*, abs/2312.10997.

Kazuma Hashimoto, Iftekhar Naim, and Karthik Raman. 2024. How does beam search improve span-level confidence estimation in generative sequence labeling? In *Proceedings of the 1st Workshop on Uncertainty-Aware NLP (UncertaiNLP 2024)*, pages 62–69, St Julians, Malta. Association for Computational Linguistics.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *NAACL*.

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*. Association for Computational Linguistics.

Hyunji Lee, Sohee Yang, Hanseok Oh, and Minjoon Seo. 2022. Generative multi-hop retrieval. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press.

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. 2023b. 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*.

Zhenwen Li and Tao Xie. 2024. Using llm to select the right sql query from candidates. *Preprint*, arXiv:2401.02115.

Niklas Muennighoff. 2022. Sgpt: Gpt sentence embeddings for semantic search. *ArXiv*, abs/2202.08904.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022.

A survey on text-to-sql parsing: Concepts, methods, and future directions. *ArXiv*, abs/2208.13629.

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*. Association for Computational Linguistics.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Tianshu Wang, Hongyu Lin, Xianpei Han, Le Sun, Xiaoyang Chen, Hao Wang, and Zhenyu Zeng. 2024. Dbcopilot: Scaling natural language querying to massive databases. *Preprint*, arXiv:2312.03463.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. 2023. Self-evaluation guided beam search for reasoning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Wenhan Xiong, Xiang Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Scott Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oguz. 2021. Answering complex open-domain questions with multi-hop dense retrieval. In *International Conference on Learning Representations*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

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*. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

Zichun Yu, Chenyan Xiong, Shi Yu, and Zhiyuan Liu. 2023. Augmentation-adapted retriever improves generalization of language models as generic plug-in. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Jiahao Zhang, Haiyang Zhang, Dongmei Zhang, Yong Liu, and Shen Huang. 2024. End-to-end beam retrieval for multi-hop question answering. In *2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *Preprint*, arXiv:2303.18223.

A How to Adress the Scenarios Where No Relevant Tables Exist

In this section, we discuss how our method can be improved to solve the situation that there are no tables that are relevant to the user question in the given databases (Yu et al., 2019a). According to the existing open-domain database tables, we can manually annotate or synthesize relevant and irrelevant questions to train a discriminator. For each user question, before retrieving the relevant tables, we can use the discriminator to determine whether the question is irrelevant to the existing tables. And if the question is irrelevant, we can directly output the feedback, and no longer retrieve tables and generate SQL (Jeong et al., 2024).

B Prompts for Rewrite

In the section, we show the prompts we use to rewrite the question on SpiderUnion (see Table 6) and BirdUnion (see Table 7). Each table is represented in the form of "database name.tabel name(column name, column name, ...)" following Kothyari et al. (2023). We only show the first two examples here limited by pages. The code and the whole prompt will be public in the future.

C Normalization Method

In this section, we show how to normalize the cosine similarity into a probability distributed between 0 and 1 present in Equation 2.1. We define the cosine similarity between the question q vector and the table t_i vector as s, which is distributed between -1 and 1. And we use Equation C.1 to normalize the cosine similarity s.

$$Norm(s) = \frac{s+1}{2} \tag{C.1}$$

Moreover, $Norm(s)$ is proportional to s, that is, the greater the cosine similarity s, the greater $\hat{P}(t_i|q)$, that is, the greater the probability that the table t_i is retrieved by q.

D Score of the Retrieval Path

In this section, we prove the calculation process of the retrieval path probability present in §2.5. First of all, we define the retrieval path $\texttt{Path}^{h,b}$ as Equation D.1, where $q^{h,b}$ represents the user question of hop h and beam b, and $t_{p_h}^{q^{h,b}}$ represents the table retrieved by $q^{h,b}$ ranked at p_h.

$$\texttt{Path}^{h,b} = ((q^{1,b}, t_{p_1}^{q^{1,b}}), ..., (q^{h,b}, t_{p_h}^{q^{h,b}})) \tag{D.1}$$

According to the discussion in §2.3, the score of each retrieval path $\texttt{Path}^{h,b}$ can be regarded as the probability that the last table in the path $t_{p_h}^{q^{h,b}}$ is retrieved in the case of the user question at last hop $q^{h,b}$. Formally, it can be summarized as:

$$\begin{aligned}
&\texttt{Score_Path}((q^{1,b}, t_{p_1}^{q^{1,b}}), ..., (q^{h,b}, t_{p_h}^{q^{h,b}})) \\
=&\hat{P}((q^{1,b}, t_{p_1}^{q^{1,b}}), ..., (q^{h,b}, t_{p_h}^{q^{h,b}})) \\
=&\hat{P}(t_{p_h}^{q^{h,b}}|q^{h,b}) \cdot \hat{P}((q^{1,b}, t_{p_1}^{q^{1,b}}), ..., (q^{h-1,b}, t_{p_{h-1}}^{q^{h-1,b}})) \\
=&... \\
=&\prod_{j=1}^{h} \hat{P}(t_{p_j}^{q^{j,b}}|q^{j,b})
\end{aligned} \tag{D.2}$$

Therefore, we multiply all $\hat{P}$ on the retrieval path as the score of the path.

E Table Scoring Algorithm in MURRE

In this section, we detail the table scoring algorithm (Algorithm 1), which is discussed in §2.5.

F How to Handle Ambiguous Entities or Synonyms

In this section, we present how our method can be improved to handle ambiguous entities or synonyms within user questions or database tables. When scoring the retrieved tables, in the face of multiple similar tables retrieved due to ambiguous entities or synonyms, we can additionally consider the correlation between tables and select tables with higher relevance scores of other retrieved tables (Chen et al., 2024b).

G Dataset Details

In this section, we introduce in detail the source dataset of SpiderUnion and BirdUnion datasets which we use. Spider (Yu et al., 2018) is a multi-domain mainstream text-to-SQL dataset that contains 658 questions, with an average of 1.48 tables per question in the dev-set. Bird (Li et al., 2023b), as a text-to-SQL dataset, is closer to the actual scenario featuring its larger scale and more difficult questions. Bird contains 1534 questions, with an average of 1.92 tables per question in the dev-set.

H Model Details

In the section, we introduce the models SGPT and `gpt-3.5-turbo` used in our experiments. SGPT (Muennighoff, 2022) is the popular retrieval

Given the following SQL tables, your job is to complete the possible left SQL tables given a user's request.
Return None if no left SQL tables according to the user's request.

Question: Which models are lighter than 3500 but not built by the 'Ford Motor Company'?
Database: car_1.model list(model id, maker, model)
car_1.cars data(id, mpg, cylinders, edispl, horsepower, weight, accelerate, year)
car_1.car names(make id, model, make)
Completing Tables: car_1.car makers(id, maker, full name, country)

Question: Which employee received the biggest bonus? Give me the employee name.
Database: employee_hire_evaluation.evaluation(employee id, year awarded, bonus)
employee_hire_evaluation.employee(employee id, name, age, city)
Completing Tables: None
...

Table 6: The prompt we use for the SpiderUnion with `gpt-3.5-turbo`.

Given the following SQL tables, your job is to complete the possible left SQL tables given a user's request.
Return None if no left SQL tables according to the user's request.

Question: What was the growth rate of the total amount of loans across all accounts for a male client between 1996 and 1997?
Database: financial.client(client_id, gender, birth_date, location of branch)
financial.loan(loan_id, account_id, date, amount, duration, monthly payments, status)
Completing Tables: financial.account(account id, location of branch, frequency, date)
financial.disp(disposition id, client_id, account_id, type)

Question: How many members did attend the event 'Community Theater' in 2019?
Database: student_club.Attendance(link to event, link to member)
Completing Tables: student_club.Event(event id, event name, event date, type, notes, location, status)
...

Table 7: The prompt we use for the BirdUnion with `gpt-3.5-turbo`.

Single-hop, employing a decoder-only architecture and showing excellent performance on tasks such as sentence matching. `gpt-3.5-turbo` (Zhao et al., 2023) has undergone instruction fine-tuning and human alignment and has superior in-context learning and inference capability.

I The Evaluation Metric in Analysis Experiments

In this section, we explain the reasons for using complete recall $k = 5$ as the evaluation metric in the analysis experiments. The increasing trend of the performance in the text-to-SQL becomes slow or even drops when inputting retrieved tables more than 5 as shown in Table 3, and considering that SpiderUnion and BirdUnion require up to 4 tables for each question, so in the following analysis, we are mainly concerned with the performance of the top 5 retrieval results. Furthermore, complete recall $k = 5$ is a more strict indicator than $recall@5$, so we mainly utilize complete recall $k = 5$ as the evaluation metric in analysis experiments.

J Discussion on Efficiency

In this section, we discuss the comparison of efficiency between MURRE and CRUSH. Because of each user question, CRUSH needs to use LLM to predict the relevant tables once, and then retrieve all the tables according to the LLM prediction once. So the time complexity of CRUSH is shown in Equation J.1, where n is the number of user questions.

$$\begin{aligned} T(CRUSH) &= O(2 \cdot n) \\ &= O(n) \end{aligned} \tag{J.1}$$

Suppose that the number of hop is H and the beam size is B in MURRE. For each user question, MURRE needs to retrieve all the tables first, and input LLM for rewriting according to the retrieved top B tables. In the subsequent hops, each hop needs to retrieve B times and rewrite B times with LLM. Therefore, the time complexity of our method is present in Equation J.2.

$$\begin{aligned} T(\text{MURRE}) &= O((1 + B + (H - 1) \cdot B \cdot 2) \cdot n) \\ &= O(B \cdot H \cdot 2 \cdot n) \\ &= O((B \cdot H) \cdot n) \end{aligned} \tag{J.2}$$

Algorithm 1 The table scoring algorithm in MURRE

Input: The similarity corresponding to each table t in each hop h: $all\_paths = [[(table_{11}, score_{11}), \ldots, (table_{1H}, score_{1H})], \ldots, [(table_{P1}, score_{P1}), \ldots, (table_{PH}, score_{PH})]]$, the number of max hops H, the number of all paths P.

Output: The scores of each table t

Initialization : $table\_score \leftarrow \{\}$
for $each\_path$ in all_paths **do**
  $score \leftarrow 1$ ▷ Initialize the score
  for $example$ in $each\_path$ **do**
    $score = score \times example\,[1]$ ▷ Calculate the score of the path
  end for
  for $example$ in $each\_path$ **do**
    $table\_score\,[example\,[0]] \leftarrow max(score, table\_score\,[example\,[0]])$ ▷ Update the table score with the max path score
  end for
end for
return $table\_score$

It can be found that although our method has significantly improved the performance compared with CRUSH, our method is less efficient. However, existing work shows that using reasoning efficiency for improving the reasoning performance has a wide range of practical application value (Yao et al., 2023; Xie et al., 2023; Press et al., 2023; Hashimoto et al., 2024). Therefore, in practical applications, how to choose B and H in MURRE to achieve a balance between retrieval efficiency and effect should be carefully considered.

K Impact of SQL Hardness

Method	Easy	Medium	Hard	Extra	All
Single-hop	70.5	71.1	55.8	51.3	66.0
MURRE	**71.8**	**76.0**	**73.3**	**73.1**	**74.2**

Table 8: Complete recall $k = 5$ of MURRE compared with the Single-hop in different SQL hardness levels on SpiderUnion. **Extra** denotes extra hard. **All** refers to the performance of the whole SpiderUnion dataset. The best results of different hardness are annotated in **bold**.

In this section, we show the performance of MURRE on SQL of different hardness levels. We categorize the SQL and its corresponding question according to the SQL hardness criteria (Yu et al., 2018) and calculate the retrieval performance of different hardness levels, as shown in Table 8. MURRE improves performance more significantly for more difficult SQL questions. Because more difficult SQL often requires more tables to operate and query, the Single-hop is challenging to retrieve all relevant tables merely in a single hop, while our method can retrieve more relevant tables with multi-hop retrieval by removing the retrieved information from the question at each hop.

L Detailed Case Study

We present one example in detail with MURRE compared with the Single-hop, MURRE without beam search and MURRE without Rewrite respectively in Table 9, Table 10, and Table 11. We demonstrate the example, with setting the beam_size to 3 and max hop to 3, while MURRE stops early at the second hop.

As shown in Table 9, the Single-hop retrieval fails to retrieve the table *"world_1.countrylanguage"* at top 3 limited by the single-hop retrieval since the retrieval of table *"world_1.countrylanguage"* relies on the table *"world_1.city"*. As displayed in Table 10, MURRE without beam search method is affected by error cascades, because the table *city_record.city* with the highest retrieval ranking in hop 1 is irrelevant to the question. Rewriting based on the irrelevant *city_record.city* table will lead to retrieval errors in subsequent hops. As present in Table 11, MURRE without Rewrite adds the retrieved tables directly to the user question, so that the subsequent retrieved tables are similar to the currently retrieved tables. For example, the irrelevant table *"city_record.hosting city"* retrieved in hop 2 is similar to the table

Question
What is the most populace city that speaks English?

Single-hop ($r@3 = 50.0$)
Retrieved Tables (top 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
world_1.city(id, name, country code, district, population)
e_government.addresses(address id, line 1 number building, town city, zip postcode, state province county, country)

MURRE ($r@3 = 100.0$)
Retrieved Tables, hop = 1 **(top** 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
world_1.city(id, name, country code, district, population)
e_government.addresses(address id, line 1 number building, town city, zip postcode, state province county, country)
Rewritten Questions
city_record.language(city id, language, percentage)
world_1.countrylanguage(countrycode, language, is official, percentage)
e_government.languages(language id, language name, language code, population)
Retrieved Tables, hop = 2 **(top** 3**)**
world_1.city(id, name, country code, district, population))
world_1.countrylanguage(countrycode, language, is official, percentage)
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
Rewritten Questions
None
None
None
(Early Stop)

Table 9: Detailed case study comparing MURRE with Single-hop. The green means the relevant table, while the red means irrelevant. Each table is expressed in the form of "database name.tabel name(column names)". r denotes recall.

"city_record.city" retrieved in hop 2, which are both about *"city"* information, but ignore the information of *"language"*. And our method focuses on retrieving tables about *"language"* by removing the information *"world_1.city"* in the retrieved tables, and successfully retrieves two relevant tables.

Question
What is the most populace city that speaks English?

MURRE without beam search ($r@3 = 0.0$)
Retrieved Tables, hop = 1 **(top** 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
world_1.city(id, name, country code, district, population)
e_government.addresses(address id, line 1 number building, town city, zip postcode, state province county, country)
Rewritten Question
city_record.language(city id, language, percentage)
Retrieved Tables, hop = 2 **(top** 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
city_record.hosting city(year, match id, host city)
city_record.match(match id, date, venue, score, result, competition)
Rewritten Question
None
(Early Stop)

MURRE ($r@3 = 100.0$)
Retrieved Tables, hop = 1 **(top** 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
world_1.city(id, name, country code, district, population)
e_government.addresses(address id, line 1 number building, town city, zip postcode, state province county, country)
Rewritten Questions
city_record.language(city id, language, percentage)
world_1.countrylanguage(countrycode, language, is official, percentage)
e_government.languages(language id, language name, language code, population)
Retrieved Tables, hop = 2 **(top** 3**)**
world_1.city(id, name, country code, district, population))
world_1.countrylanguage(countrycode, language, is official, percentage)
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
Rewritten Questions
None
None
None
(Early Stop)

Table 10: Detailed case study comparing MURRE with MURRE without beam search. The green means the relevant table, while the red means irrelevant. Each table is expressed in the form of "database name.tabel name(column names)". r denotes recall.

Question
What is the most populace city that speaks English?

MURRE without Rewrite ($r@3 = 50.0$)
Retrieved Tables, hop = 1 **(top** 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
world_1.city(id, name, country code, district, population)
e_government.addresses(address id, line 1 number building, town city, zip postcode, state province county, country)
Spliced Questions
What is the most populace city that speaks English?; city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
What is the most populace city that speaks English?; world_1.city(id, name, country code, district, population)
What is the most populace city that speaks English?; e_government.addresses(address id, line 1 number building, town city, zip postcode, state province county, country)
Retrieved Tables, hop = 2 **(top** 3**)**
city_record.hosting city(year, match id, host city)
county_public_safety.city(city id, county id, name, white, black, amerindian, asian, multiracial, hispanic)
world_1.country(code, name, continent, region, surface area, indepdent year, population, life expectancy, gnp, gnp old, local name, ...)
Spliced Questions
What is the most populace city that speaks English?; world_1.city(id, name, country code, district, population); city_record.hosting city(year, match id, host city)
What is the most populace city that speaks English?; world_1.city(id, name, country code, district, population); city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
What is the most populace city that speaks English?; city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp); city_record.hosting city(year, match id, host city)
Retrieved Tables, hop = 3 **(top** 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
city_record.hosting city(year, match id, host city)
world_1.city(id, name, country code, district, population)

MURRE ($r@3 = 100.0$)
Retrieved Tables, hop = 1 **(top** 3**)**
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
world_1.city(id, name, country code, district, population)
e_government.addresses(address id, line 1 number building, town city, zip postcode, state province county, country)
Rewritten Questions
city_record.language(city id, language, percentage)
world_1.countrylanguage(countrycode, language, is official, percentage)
e_government.languages(language id, language name, language code, population)
Retrieved Tables, hop = 2 **(top** 3**)**
world_1.city(id, name, country code, district, population))
world_1.countrylanguage(countrycode, language, is official, percentage)
city_record.city(city id, city, hanzi, hanyu pinyin, regional population, gdp)
Rewritten Questions
None
None
None
(Early Stop)

Table 11: Detailed case study comparing MURRE with MURRE without Rewrite. The green means the relevant table, while the red means irrelevant. Each table is expressed in the form of "database name.tabel name(column names)". r denotes recall.