In-Context Reinforcement Learning based Retrieval-Augmented Generation for Text-to-SQL

Rishit Toteja
Amazon
toteja@amazon.com

Arindam Sarkar
Amazon
arindsar@amazon.com

Prakash Mandayam Comar
Amazon
prakasc@amazon.com

Abstract

Text-to-SQL simplifies database interactions by enabling non-experts to convert their natural language (NL) questions to Structured Query Language (SQL) queries. With advancements in Large Language Models (LLM), in-context learning (ICL) has emerged as a popular choice for building Text-to-SQL systems. Real world, industry-scale databases, often comprise thousands of tables and hundreds of columns, and makes passing the entire schema as context to an LLM infeasibly expensive. This requisites access to the correct database and the set of tables. Recently Retrieval Augmented Generation (RAG) based methods have been proposed for retrieving relevant subset of databases and tables for a given query. However, we observe that the existing methods of synthetic query generation can generate predominantly simple queries which might not be sufficiently representative of complex, real world queries, thus, negatively affecting the quality of the generated SQL. To address this, we propose an *innovative* in-context reinforcement learning (ICRL) based framework which refines the question generation process by enhancing the model's ability to produce intricate queries that practitioners may pose during inference. In contrast to the existing approaches, our framework ensures the generation of synthetic SQL queries which are diverse and complex. We demonstrate the effectiveness of our approach via multiple experiments comparing against the representative state-of-the-art models on public benchmark datasets and observe substantial improvements in performance and scalability. Our method achieves 15-20% higher recall in database/table retrieval task compared to the existing state-of-the-art models for schema identification and upto 2% higher execution accuracy for SQL generation.

1 Introduction

The complexity of formulating effective database queries demands significant manpower and technical expertise, underscoring the need for innovative Text-to-SQL solutions to bridge the gap between natural language and data management. Recently Large Language Models (LLMs) finetuned for SQL generation have shown state-of-the-art results on the representative Text-to-SQL benchmarks [1,2]. In typical real world systems, databases are constantly evolving, and to accommodate new fields or relationships, the LLMs need to be continually finetuned to maintain the quality of generation. However, the most performant LLMs have parameters in billion-scale (Zhao et al., 2023), and finetuning these models is expensive and require technical expertise. In-context and few-shot learning has emerged a popular alternative, and has been shown to be extremely effective on the Text-to-SQL tasks by works like DIN-SQL (Pourreza and Rafiei, 2023). For syntactically correct SQL generation, the LLM needs to be schema aware. Industry scale databases often consisting of thousands of tables, and hundreds of columns, passing the entire schema as context to LLM is prohibitively expensive. This requisites a framework which can fetch relevant schemas for correct SQL generation. In contrast to using RAG for NLP tasks, where relevant knowledge retrieval can be done via similarity search in embedding space, as the table schemas might not be syntactically relevant for a natural language query. Towards this, schema routing was proposed in DBCopilot (Wang et al., 2024b) for effective synthetic data generation. It leverages the foreign key linkage between tables to perform random walks and generating corresponding synthetic natural language queries to aid in table retrieval for a query. This approach was shown to have state-of-the-art performance in database/table retrieval.

However, simply generating synthetic queries based on table relationships is not guaranteed to

[1] https://yale-lily.github.io/spider/
[2] https://bird-bench.github.io/

be representative of human generated queries, and consequently might under-represent the complex queries involving diverse operators (Figure A.2). Simple prompting based approach might generate questions which require trivial SQL operators, for e.g., "What is the most expensive book based on purchase price?". Consequently for complex queries this is susceptible to fetching irrelevant schemas (examples in Table 5). In this work, we propose a *novel* in-context Reinforcement Learning based framework to iteratively improve the quality of generated synthetic queries from a base LLM by employing a Feedback LLM which generates instructions to modify the base generation to maximize a reward function which encourages generation of complex queries. The proposed ICRL approach refines the preceding example to additionally generate synthetic NL queries like "What is the most expensive book based on purchase price for books written by authors whose last name starts with 'S', and what are the author and title of that book?" (refer to A.4 for more examples). This augmentation results in significant gains over the representative models for schema retrieval task, outperforming both finetuned and ICL based models. When the proposed RAG mechanism is utilized for few-shot SQL generation, it outperforms the state-of-the-art ICL based models on SQL generation as well, further concretizing the importance of correct schema retrieval for correct SQL generation.

2 Background

Early work in Text-to-SQL models the problem as a sequence-to-sequence task and proposed encoder-decoder architectures (Yu et al., 2018a). (Qi et al., 2022) introduce a novel architecture, by modifying the attention layer of encoder of T5 and including relation embeddings into the key and value entries. In contrast to training shallow Seq2Seq models, recently LLMs like GPT-4 (OpenAI) have demonstrated to be effective in both zero-shot and few-shot scenarios as shown in DAIL-SQL (Gao et al., 2024). Further performance improvement is observed with supervised finetuning, which enhances LLMs using additional task-specific training data to make it more suitable for domain-specific SQL generation by finetuning LLMs like CodeLlama 34B and 7B released by Defog [3] achieve highest performance. Owing to the cost implication of finetuning LLMs, there is increased interest in prompting based techniques for Text-to-SQL tasks given the schema and relevant examples in context (Guo et al., 2023; Wang et al., 2024c). LLMs performance depends a lot on the demonstrations chosen for in-context learning as shown in FUSED (Wang et al., 2024a).

RAG for Large Databases RAG enhances the performance of LLMs (Lewis et al., 2020) on knowledge-intensive NLP tasks like Text-to-SQL by combining the strengths of pre-trained models with knowledge contained in specialized data stores, (e.g., database/table metadata). The hybrid approach helps to decrease the amount of context given to LLMs. With the arrival of large context length LLMs like Gemini 1.5 [4] and Claude [5] , it is possible to feed entire database schemas directly as context. However, we show that in many cases this approach fails to identify the correct set of tables relevant for solving a user question. Thus for massive databases containing thousands of tables, there is a need for intelligent retrieval for determining a high recall subset of the database/tables to enhance the SQL generation (Kothyari et al., 2023).

LLMs with Iterative Feedback LLMs exhibit a remarkable capability for improving from feedback (Kwon et al., 2023; Wang and Li, 2023). (Madaan et al., 2023) proposes a method for refining model outputs through an iterative process of self-feedback. (Du et al., 2024) introduces a novel method to improve the response generation of LLMs by incorporating multiple rounds of debate between different agents. Here we propose an in-context reward guided refinement of the base LLM, which iteratively improves the model output.

3 Methodology

Given a NL query, we first identify the most relevant schemas from diverse databases. To limit the schema search space, we reduce the scope from a large array of databases D to a smaller superset $S \subseteq D$. Our goal is to identify S so it retains high recall of the databases and tables in the ground truth SQL query. For efficient retrieval, we construct a graph to represent all databases (Wang et al., 2024b), with each traversal corresponding to a subset of schemas, and use the traversals to generate synthetic data to be stored in a knowledge

[3] https://huggingface.co/defog/

[4] https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/

[5] https://www.anthropic.com/news/claude-3-family

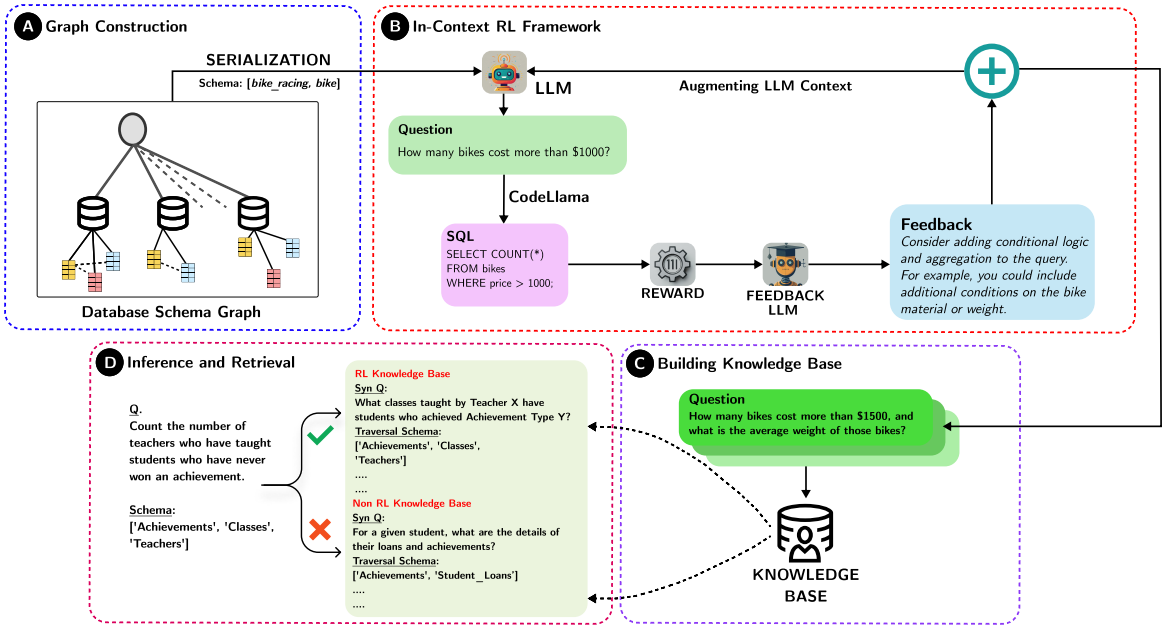

Figure 1: Overview of the proposed In-Context RL based RAG architecture for schema retrieval.

base (KB) for the RAG mechanism.

Schema Graph Construction The database/table schema graph G is constructed by initializing a root node R, with type-1 edges to each database, type-2 edges to tables, and type-3 edges representing foreign key relationships. We generate all possible traversals on G via fixed length random-walks where each traversal represents a unique path from R through G, as detailed in Algorithms A.1, A.2.

Synthetic Question Generation In order to identify S from the entire Database represented in Graph G, we begin by selecting a particular traversal from the graph. This step, referred to as serialization, involves mapping out a specific path through the graph, starting from the root node and following the edges through various databases and tables. Once a traversal is serialized, we carefully prompt a LLM to generate the corresponding natural language question(s) and SQL solution(s), and store the triplet in a KB keyed by the NL question.

3.1 In-Context RL Framework

To ascertain that the generated synthetic questions are relevant as well as sufficiently complex to reflect the nature of human queries, we propose an in-context reinforcement learning (ICRL) framework to iteratively improve the LLM's generation of synthetic questions. For each traversal, we formulate the interactive process as a Markov Decision Process (MDP). The **state** (s_t) at time t includes the current context provided to the LLM and its parameters (θ_t), comprising of schema information and previously generated questions. The **action** (a_t) is the generated synthetic question (q_{s_t}) by the LLM. For a pretrained LLM with frozen θ_t, the policy determining the action $\pi(a_t|s_t, ..)$ is the probability of generating a sequence of tokens given the context.

Reward Function To encourage the creation of synthetic questions with the desired complexity, we use a **reward** function based on keywords (k_j) from the intermediate SQL query generated by a LLM when given the synthetic question as input. Specifically, we define four keyword buckets: B_1 (data retrieval and filtering), B_2 (data modification), B_3 (conditional logic), and B_4 (aggregation and sorting). The complexity score of each bucket is calculated as:

$$c(B_i) = \frac{\sum_{k_j \in S_t} (k_j \in B_i)}{\sum_{B_i} \sum_{k_j \in S_t} (k_j \in B_i)} \quad (1)$$

Details of the carefully curated keyword scores are provided in A.3. These are selected to closely mimic the SQL operators used by human practitioners for complex NL queries. For instance, a query with multiple JOIN/GROUP BY statements is likely to be more complex compared to a query with only SELECT/AND/OR operators. The resulting reward function $R(S_t)$ is based on bucket frequencies and their weights:

$$R(S_t) = \sum_i f(B_i, S_t) \cdot c(B_i) \quad (2)$$

Table 1: Table retrieval recall (%) for the Spider and Bird Dev datasets.

Model	Spider-Dev				Bird-Dev			
	R@1	R@2	R@5	R@10	R@1	R@2	R@5	R@10
DBCopilot	49.94	81.01	85.34	85.34	34.30	56.73	61.02	61.02
RAG (BM25)	42.71	54.07	66.83	77.31	26.92	35.33	46.61	54.82
RAG (embedding)	64.92	76.52	89.24	93.99	43.54	59.32	79.79	90.61
RAG (emb.) + SXFMR	64.74	79.36	90.35	94.36	43.87	51.10	61.73	70.20
RAG (emb.) + ICRL (iter=1)	71.12	81.50	91.61	95.90	50.19	65.18	82.07	92.24
RAG (emb.) + ICRL (iter=2)	**71.44**	**81.64**	**91.66**	**95.94**	**51.63**	**66.03**	**82.40**	**92.24**

Table 2: LLM Aided RAG (Prompting LLM to perform aggregation on top-k retrieved schemas).

Top-K (To be merged)	Spider-Dev (R@1)		Bird-Dev (R@1)	
	RAG	RAG + ICRL	RAG	RAG + ICRL
5	86.21	**89.10**	73.01	**75.22**
10	88.91	**91.10**	79.07	**80.89**
15	89.98	**91.80**	80.70	**82.98**
20	90.59	**92.50**	80.96	**82.26**
All DBs provided in-context (Claude)		61.62		80.76

Table 3: EX (Execution Accuracy) and I/P (input) tokens on the Spider-Dev dataset.

Approach	Approach	I/P Tokens	EX (%)
DB schemas present	DIN-SQL	9916.56	74.2
	DBCopilot	199.24	74.4
	FUSED	778.25	74.9
	DAIL-SQL	922.21	75.5
	Ours (0-shot)	249.24	**75.9**
	Ours (1-shot)	387.26	**76.6**
DB schemas inferred	DBCopilot	93.62	64.12
	Ours (0-shot)	196.39	65.2
	Ours (1-shot)	318.93	**69.6**

where $f(B_i, S_t)$ is the frequency of bucket B_i within the SQL query S_t.

As shown in (Figure 1), initially, the base LLM generates a synthetic question q_{s_0} relevant to the schema, with a corresponding SQL query S_0. The reward $R(S_0)$ is calculated based on its complexity and keyword distribution. Subsequently, the Feedback-LLM receives in-context examples, the synthetic question q_{s_t}, and the reward signal $R(S_t)$. This Feedback-LLM generates textual feedback to guide the base LLM in modifying q_{s_t}. Once incorporated in the context, this feedback effectively modifies the generation policy, and as we show in our experiments, improves the base LLM's generation for the next iteration. At each iteration t, the state s_{t+1} updates with the initial context and additional feedback. The framework iteratively refines the synthetic question q_s based on the reward signal to get the final $q_{s_{\text{final}}}$, which is indexed in the knowledge base.

LLM Aided Schema Pooling We further improve the recall of schema retrieval by leveraging the reasoning capabilities of LLM, wherein given the user query, first the top-k schemas are fetched from the KB, and then given this context, a LLM is prompted to select the most relevant schemas from the candidates.

3.2 SQL Generation

Once schema is selected, we prompt a LLM to generate the SQL query. The LLM is provided with examples from KB in form (q, D, S) where q is user question, D is database schema, and S is retrieved SQL query. These along with the chosen schema, are used to produce the final SQL query.

4 Experiments and Results

We use Claude (Sonnet) for synthetic question generation and schema retrieval. Cohere-Embed-English-v3 [6] is used for generating embeddings for indexing in knowledge-base. Additionally, we compare the embedding based retrieval with BM25, a standard ranking function in search engines for document relevance, and SXFMR (Reimers and Gurevych, 2019) which applies contrastive learning to Transformer-based models for generic embedding retrieval on related text pairs. For SQL generation, we use GPT-3.5 Turbo model via the official API (OpenAI).

Evaluation Metrics We employ standard evaluation metrics used for text-to-SQL task. For schema retrieval, we compute table recall, measuring the percentage of top-k schemas retrieved that match the gold schema. We consider Execution Accuracy (EX) for evaluation of generated SQL query. All experiments were performed on Spider-Dev (Yu et al., 2018b) and Bird-Dev (Li et al., 2023) datasets (details in A.1).

Schema Retrieval Recall The feedback-based model variants achieve superior recall compared to the representative baselines, including DB-Copilot which is a finetuned model. Specifically, on Spider-Dev, R@1 improved by 21.5%, and on Bird-Dev, R@1 increased by 17.3%, demonstrating the effectiveness of ICRL approach. It can be observed

[6] https://cohere.com/blog/introducing-embed-v3

from Table 2, directly providing all databases as context to the LLM does not always yield optimal performance, and might be attributed to distraction in presence of irrelevant tokens (Shi et al., 2023).

Execution Accuracy (EX): When the gold database schema is provided as context, our approach achieves high execution accuracy (EX), as shown in Table 3. Both zero-shot and 1-shot variants of our model achieve around $\sim$2% higher EX, surpassing all the baselines. Compared to our model, DIN-SQL and DAIL-SQL, current state-of-the-art models in this setting, use significantly more tokens. In contrast, we achieve substantial cost reductions, requiring 25.6x and 2.38x fewer tokens than DIN-SQL and DAIL-SQL, respectively. When the gold schema context is absent, our schema retrieval method outperforms DBCopilot, achieving 69.6% EX in the 1-shot scenario.

5 Conclusion

While LLMs are trained on vast amounts of public data, they are unable to readily handle domain specific/confidential industry scale databases. The impracticality and cost of finetuning LLMs with dynamic databases underscores the importance of efficient schema retrieval methods as an important step in Text-to-SQL applications. In this work, we propose a novel in-context reinforcement learning based RAG framework for efficient schema and in-context example retrieval for Text-to-SQL tasks. Our approach requires no specialized finetuning, and is based on composable prompting based modules, and outperforms representative state-of-the-art baselines for both schema retrieval and SQL generation tasks. While we benchmark the presented approach on the Text-to-SQL, the approach is generalizable to other problems requiring iterative refinement on top of LLMs as well.

6 Limitations

Since we are not using fine-tuned LLMs for SQL generation, they may still lack information or understanding about the specific databases in context. Apart from this, while powerful, the proposed model may not inherently understand or provide meaningful interpretations of the database schemas they are working with, especially if those schemas do not have natural language descriptions.

References

BM25. Okapi bm25 — Wikipedia, the free encyclopedia.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. 2023. Prompting gpt-3.5 for text-to-sql with desemanticization and skeleton retrieval. In *PRICAI 2023: Trends in Artificial Intelligence: 20th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2023, Jakarta, Indonesia, November 15–19, 2023, Proceedings, Part II*, page 262–274, Berlin, Heidelberg. Springer-Verlag.

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, Singapore. Association for Computational Linguistics.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. 2023. Reward design with language models. In *The Eleventh 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, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

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*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

OpenAI. Models overview - openai. https://platform.openai.com/docs/models/overview/.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems*, volume 36, pages 36339–36348. Curran Associates, Inc.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. RASAT: Integrating relational structures into pretrained Seq2Seq model for text-to-SQL. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3215–3229, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR.

Danqing Wang and Lei Li. 2023. Learning from mistakes via cooperative study assistant for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10667–10685, Singapore. Association for Computational Linguistics.

Dingzirui Wang, Longxu Dou, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. 2024a. Improving demonstration diversity by human-free fusing for text-to-SQL. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1193–1207, Miami, Florida, USA. Association for Computational Linguistics.

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

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

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594, New Orleans, Louisiana. 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. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

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 Appendix

A.1 Dataset Description

We used two development sets for our experiments, Spider and Bird. Spider and Bird are cross-domain datasets in English widely used for benchmarking. Bird tries bridges the gap between text-to-SQL research and real-world applications by dealing with large and messy database values. The statistics below include the total size and the distribution of queries by difficulty levels.

Table 4: Statistics of Spider-Dev and Bird-Dev

Dataset	Easy (Simple)	Medium (Moderate)	Hard (Challenging)	Extra
Spider (2147)	470	857	463	357
Bird (1534)	925	465	144	-

A.2 Construction of the Database Graph

The database graph G starts with a root node R. An edge of type-1 connects R to each database D_i. From each database D_i, edges of type-2 connect to its constituent tables T_{ij}. Additionally, edges of type-3 represent relationships between tables within the same database, specifically foreign key constraints. If a table T_{ij} in database D_i references another table T_{ik}, a type-3 edge connects T_{ij} to T_{ik}. (Algorithm A.1). Once the database graph is constructed, we proceed to generate traversals on this graph following Algorithm A.2.

Ground Truth		Simple Retrieval		ICRL Augmented Retrieval	
Question	**Gold Schema**	**Synthetic Question**	**Schema (R@1)**	**Synthetic Question**	**Schema (R@1)**
What is the title of the book written by George Orwell that has the lowest sale price?	[‘Book’, ‘Author_book’, ‘Author’]	Which books have the highest sale price?	[‘Author_Book’, ‘Book’]	What are the titles of books written by authors whose books have a sale price between $10 and $20, ordered by the highest sale price?	[‘Book’, ‘Author_book’, ‘Author’]
What are the distinct ids of customers who bought lemon flavored cake?	[‘items’, ‘receipts’, ‘goods’]	What are the details of customers with a specific customer ID?	[‘Customers’]	Which items of a specific flavor and food type were ordered by customers with IDs between 100 and 200 on a given date range?	[‘items’, ‘receipts’, ‘goods’]
What is the least common detention type? Show the code and the description.	[‘Detention’, ‘Ref_Detention_Type’]	What is the description for a given detention type code?	[‘Ref_Detention_Type’]	What is the most common type of detention given, and for those detentions, retrieve the detention summary and other details where the summary contains the word ‘behavior’?	[‘Detention’, ‘Ref_Detention_Type’]
What are the prices and sizes of all products whose price is above the mean?	[‘Products’]	Which food items have a price above a certain amount?	[‘goods’]	What is the most expensive product of a certain color and size range, and how does its price compare to the average price of products in that category?	[‘Products’]
Which make has more than one team?	[‘team’]	What is the number of drivers per team?	[‘team_driver’]	Which car owners also sponsor at least two teams, and retrieve the team names and car makes for those teams?	[‘team’]

Table 5: Comparison of retrieved queries and schemas across the different retrieval methods.

A.3 Complexity Scores

To encourage the LLM to generate diverse and composite questions, we meticulously designed a reward function based on keyword categories. These scores can be further adjusted to suit specific use cases. Notably, these settings yielded the best results in our experiments.

Data retrieval and filtering `SELECT` (1), `FROM` (1), `JOIN` (2), `INNER JOIN` (3), `LEFT JOIN` (3), `RIGHT JOIN` (3), 4 (score to be confirmed), `ON` (2), `WHERE` (2), `GROUP BY` (3), `HAVING` (3), `ORDER BY` (2), `DISTINCT` (2), `LIMIT` (1).

Data modification `INSERT` (2), `UPDATE` (3), `DELETE` (4)

Conditional logic `AND` (1), `OR` (1), `NOT` (1), `IN` (2), `BETWEEN` (2), `LIKE` (2), `CASE` (3), `WHEN` (2), `THEN` (2), `ELSE` (2), `END` (1)

Aggregation and sorting `AVG` (3), `SUM` (3), `COUNT` (3), `MIN` (3), `MAX` (3), `ASC` (1), `DESC` (1).

A.4 Solving Complex User Questions using ICRL

We analyze the retrieval methods by their error rates across query complexity levels, proxied by the number of tables in the ground truth query schema. We compare three retrieval settings on Spider and Bird Dev sets: top 2, top 5, and LLM-aided RAG (top 1), i.e., the number of retrieved schemas from the knowledge base. Figure A.2 underscores the effectiveness of RL-based retrieval methods in achieving lower error rates for complex queries, which are common across industries, without compromising accuracy on simpler queries. Table A.5 highlights the effectiveness of ICRL augmented retrieval over simple retrieval in generating synthetic questions and retrieving relevant schemas. The ICRL approach retrieved schemas are more aligned with the ground truth and the formulated synthetic questions better capture the complexity of the original queries. This demonstrates that reinforcement learning feedback significantly helps in enhancing schema identification recall.

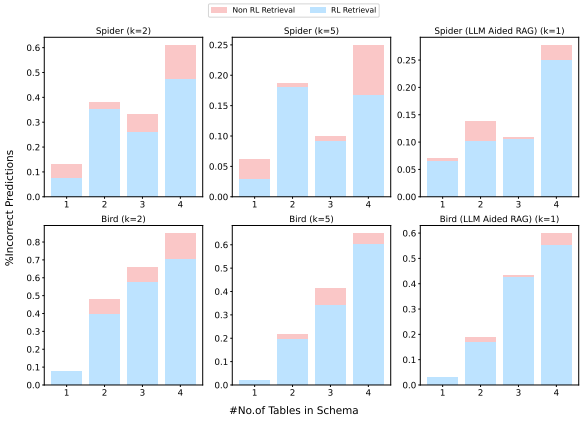

Figure 2: Comparison of Incorrect Distributions (RL and Non-RL Retrieval) on Spider and Bird Dev sets on top- k @2, @5

Algorithm 1 Construction of Database Schema Graph

Input: Metadata: List of databases and schemas
Output: Graph G representing the schema

Initialize root node R and graph G
G.add_node(R)
for each database D in databases **do**
    G.add_node(D)
    G.add_edge($R, D, 1$) ▷ Type-1 edge
    for each table T in D.tables **do**
        G.add_node(T)
        G.add_edge($D, T, 2$) ▷ Type-2 edge
        for each foreign key FK in T.foreign_keys **do**
            G.add_edge(T, FK.related_table, 3) ▷ Type-3 edge
        end for
    end for
end for
return G

Algorithm 2 Serialization of Graph with Cutoff

Input: Graph G, starting node R, cutoff length k
Output: List of all traversals from R with cutoff length k

function GET_ALL_TRAVERSALS(G, R, k)
    $traversals \leftarrow []$
    function SERIALIZE($node$, $path$, $depth$, $visited$)
        $path \leftarrow path + [node]$
        $traversals \leftarrow traversals + [path]$
        $visited[node] \leftarrow$ True
        if $depth < k$ **then**
            if G.has_children($node$) = True **then**
                $children \leftarrow G$.get_children($node$)
                if length($children$) > 1 and $edge\_type = 3$ **then**
                    $subsets\_children \leftarrow$ power_set($children$)
                    for each child c in $children$ **do**
                        for each subset s in $subsets\_children$ **do**
                            if $\forall n \in s$, visited[n] = False **then**
                                SERIALIZE(c, $path + s$, $depth +$ length(s) $+ 1$, $visited$)
                            end if
                        end for
                    end for
                else
                    for each child c in $children$ **do**
                        if visited[c] = False **then**
                            SERIALIZE(c, $path$, $depth + 1$, $visited$)
                        end if
                    end for
                end if
            end if
        end if
        $visited[node] \leftarrow$ False
        $path \leftarrow path - [node]$
    end function
    SERIALIZE(R, [], 0, {False} for all nodes in G)
    return $traversals$
end function
$k \leftarrow$ specified cutoff length
$samples \leftarrow$ GET_ALL_TRAVERSALS(G, R, k)