MR-SQL: Multi-Level Retrieval Enhances Inference for LLM in Text-to-SQL

Zhenhe Wu[1,2†*], Zhongqiu Li[2*], Mengxiang Li[2], Jie Zhang[2], Zhongjiang He[2], Jian Yang[1], Yu Zhao[2], Ruiyu Fang[2], Yongxiang Li[2], Zhoujun Li[1(✉)], and Shuangyong Song[2(✉)]

[1] Beihang University
{wuzhenhe,jiaya,lizj}@buaa.edu.cn
[2] Institute of Artifcial Intelligence (TeleAI), China Telecom Corp Ltd
{lizq48,zhangj157,hezj,zhaoy11,fangry,liyx25,songshy}@chinatelecom.cn,
limengx@126.com

Abstract. Large language models (LLMs) with in-context learning (ICL) have notably boosted the performance in text-to-SQL, with prior efforts concentrating on exclusive SQL prompts to enhance reasoning ability. However, it is still challenging to further enhance the operational efficiency and inference performance of LLMs. To tackle this challenge, we propose MR-SQL, a multi-level retrieval-based LLM framework consists of three specially designed retrievers. The retrievers collaborate to retrieve valuable information for the target question, which not only reduce schema size and minimize the interference noise, but also enhance the reasoning capability of LLMs through more similar Chains of Thought (CoT). Concretely, Table-Retriever and Column-Retriever retrieve concise tables and columns from original large databases with redundant schema information. Example-Retriever select similar few-shot examples for more targeted CoT. Experiment results indicate that MR-SQL increases the execution accuracy on the BIRD and Spider validation sets by +2.54% and +1.15% respectively.

Keywords: LLMs · In-context learning · Prompt engineering.

1 Introduction

Text-to-SQL is a task of converting natural language questions into SQL queries to obtain the answers from the database. It has attracted widespread research attention and application in database querying [19, 22, 21]. Early methods utilize pre-trained models to encode the input sequence [16]. Some researchers decode queries by abstract syntax trees [30, 7, 25], while others use predefined sketches [8]. Recent works focus on extracting the question-to-SQL patterns generalized by training an encoder-decoder model with text-to-SQL corpus [9, 11, 12, 32]. More recently, Large Language Models (LLMs) [20, 1, 17, 2, 23, 28, 14, 27] have

† Work done during the internship at TeleAI.
* These authors contributed equally to this work.

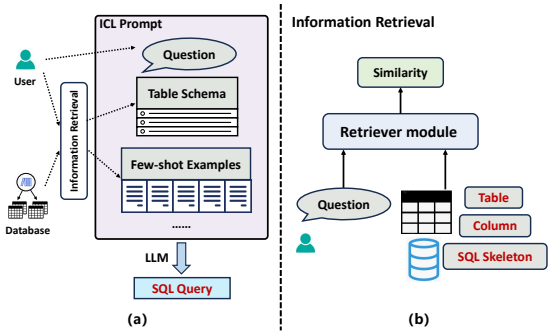

Fig. 1: (a) Pre-retrieve valuable information from databases for ICL prompt. (b) The schematic diagram of the retriever module.

shown remarkable improvement for various NLP tasks, and there has been growing interest in using LLMs to explore novel approaches for guiding SQL generation [24, 3, 6, 5, 18, 26, 15].

Different from prior studies, the fundamental solution in LLM-based text-to-SQL has primarily focused on using exclusive SQL generation prompt approaches to obtain a fully correct SQL query [6]. Existing approaches tend to utilize LLMs or simple embedding similarity calculations to achieve schema linking, yet the effects are not particularly outstanding. Due to the varying complexity and association methods of databases in different datasets, it is essential to learn from the training set how to more adaptively select schemas that contain relevant information. In addition, previous research often select few-shot examples of in-context learning by calculating the similarity of masked questions or SQL queries, while we believe that on the basis of using SQL skeleton to retrieve few-shot examples, refining the CoT approach by introducing SQL skeleton can benefit from retrieval effectiveness. Therefore, we aim to design multi-level retrievers to pre-retrieve valuable information from the raw databases. Fig 1 illustrates the flowchart and the schematic diagram of the retriever module for this approach.

In this paper, we propose a multi-level retrieval-based text-to-SQL framework named MR-SQL, which mainly contains three independent retrievers to separately calculate similarity between *question* and certain SQL data types (*tables*, *columns*, *SQL skeleton*). Fig 2 shows the whole framework of MR-SQL. Table-Retriever aims to retrieve tables that are most relevant to the question from the massive tables in database. Column-Retriever further retrieves columns in the previous retrieved tables to reduce the number of selected columns. Example-Retriever is used for searching few-shot examples having similar SQL skeleton with questions from the training set. We conduct comprehensive evaluations on two cross-domain text-to-SQL datasets BIRD and Spider, experimental results indicate MR-SQL outperforms several baselines.

To summarize, our contributions are as follows:

– We propose MR-SQL, a novel retrieval-based framework for LLMs in text-to-SQL, which contains three independent retriever modules to pre-process

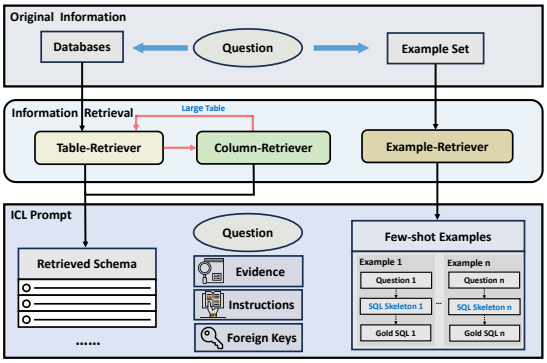

Fig. 2: Framework of the MR-SQL. Table-Retriever and Column-Retriever collaborate to retrieve relevant tables and columns for schema linking. Example-Retriever works for targeted example selection. ICL prompt includes original information such as question, foreign keys, etc. We further introduce SQL skeletons into CoT prompting.

the original information for accurate schema linking and to select few-shot examples with high reference significance.
- We introduce SQL skeleton into CoT, which benefits from the effectiveness of Example-Retriever and achieves better SQL generation.
- Experimental results demonstrate that our MR-SQL outperforms several baselines on BIRD and Spider datasets.

2 Methodology

2.1 Proposed Model

We propose a multi-level retrieval-based text-to-SQL framework for pre-processing original information and selectively retrieving valuable information, which consists of Table-Retriever, Column-Retriever and Example-Retriever. Table-Retriever filters out irrelevant tables which reduces the first interference at the database tables level. Column-Retriever aims to continuously reduce the interference caused by columns and obtain appropriate numbers of relevant columns. Table-Retriever and Column-Retriever jointly complete SQL schema linking, which reduces schema size in ICL prompt and minimize noise interference. Example-Retriever selects few-shot examples with similar SQL skeleton for questions, which helps provide targeted syntactic guidance in the CoT process. Simultaneously, we introduce SQL skeletons generation into CoT process, which benefits from the retrieval method and enhances the effectiveness of LLMs.

2.2 Table-Retriever

Table-Retriever is a module for retrieving highly correlated tables for each question. Omar Khattab *et al.* [10] discover that a model employing contextualized

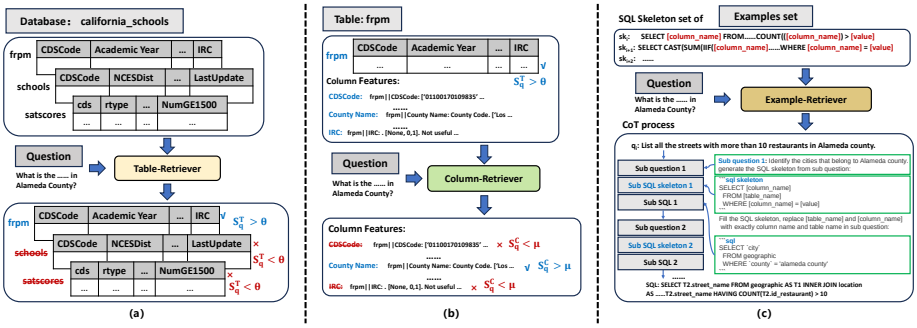

Fig. 3: (a) The workflow of Table-Retriever. The module calculate similarity of question with tables and retrieve highly relevant tables for question. (b) The workflow of Column-Retriever. The module retrieve highly relevant columns for question. (c) The workflow of Example-Retriever and CoT process.

late interaction over deep LMs is efficient for retrieval. In our model, we use BERT as encoders and MaxSim-based late interaction to calculate the similarity of question q and table t. We first convert the tables into continuous text by directly concatenating table name, column names and column types as $\{t' = name : n||c_1 : ty_1|c_2 : ty_2|...|c_n : ty_n\}$, n is table name, c_i is column name, ty_i is data type of c_i. We use q as the input of BERT $\mathcal{B}_Q^T$, which computes a contextualized representation of each token. Then, we pass the output representations through a 1D-CNN layer $\mathcal{C}$ with no activations, which is used for dimension reduction. Following the settings of ColBERT [10], we typically fix the output size m to be much smaller than BERT's fixed hidden dimension, which we discuss later. After that, we normalize the output embeddings so each has L2 norm equal to one:

$$O_q^T = \sigma\left(\mathcal{C}\left(\mathcal{B}_Q^T\left(q\right)\right)\right) \tag{1}$$

where q is the input question. We use converted table as the input of $Bert_T$, the rest of steps are the same as above, so we can get the output representations of table as follow:

$$O_t^T = \sigma\left(\mathcal{C}\left(\mathcal{B}_T^T\left(t'\right)\right)\right) \tag{2}$$

where t' is the converted input table. Next, we use the output embeddings O_q^T and O_t^T for late interaction. We calculate the dot-product similarity between each token in O_q^T and every embedding in O_t^T, then take the maximum value. We sum these maximum values to get the final similarity score between question q and table t:

$$S_{q,t}^T = \sum_{i\in[|O_q^T|]} \max_{j\in[|O_t^T|]} O_{q_i}^T \cdot O_{t_j}^T \tag{3}$$

where $S_{q,t}^T$ is the final similarity score of Table-Retriever. Fig 3 (a) shows the process of table retrieval.

2.3 Column-Retriever

Column-Retriever is the downstream module of Table-Retriever. Given the retrieved tables output by Table-Retriever, Column-Retriever can retrieve highly correlated columns for each question, such as those that are identical or semantically similar to certain entities in the question, and can further filters out redundant information of schema. The framework of Column-Retriever is the same as Table-Retriever, which is designed to calculate the similarity of question q and column c. We convert column features into continuous text c' by concatenating table name t_{name}, column name c_{name}, column description c_{desc}, examples $[e_1...e_i]$, value description v_{desc} and other knowledge k, as $\{c' = t_{name}||c_{name} : c_{desc}[e_1...e_i]v_{desc}k\}$. Then we use q and c' as the input of $\mathcal{B}_Q$ and $\mathcal{B}_C$, and obtain output embeddings O_q^C and O_c^C through similar process with Table-Retriever. In the following operation, we acquire the similarity score by the sum of MaxSim value in the same way, where $S_{q,c}^T$ is the final similarity score of Column-Retriever. Fig 3 (b) shows the process of column retrieval.

Specialized handling of Large tables In practical applications, some tables may have too many columns that the converted tables t' are so long. Since we use BERT as our encoder, which is not able to handle over 512 tokens, we need a specialized design to handle large tables. In our method, if the length of t' is over 512, we firstly use Column-Retriever to perform coarse filtering with smaller threshold μ', which can shorten the table by reducing the number of columns. Then we pass the shortened table back to Table-Retriever. All the following steps are the same as before.

2.4 Example-Retriever

According to prior studies [4], in-context learning is similar to the decision process of human beings by learning from analogy, so it is effective to select examples that are similar with the target question. In our method, we apply a Example-Retriever as the example selection module. The framework of Example-Retriever is the same as the retriever modules above, the input for BERT-based encoders are question q and SQL skeleton sk. sk is the original SQL which is masked specific content by [column_name], [table_name] and [value] token. As the retrievers above, $S_{q,sk}^T$ is the final similarity score of Example-Retriever. Before we select few-shot examples for in-context learning, we first translate all the SQL queries from our training set into SQL skeletons as a candidate set $SK = \{sk_1, sk_2...sk_n\}$. To conduct k-shot examples selection for a target question q, we apply Example-Retriever to retrieve top-k SQL skeletons from SK. Then we trace the source and find the original samples corresponding to these skeletons as our final selected k-shot examples.

Prompt Construction Inspired by the previous work [29, 33], we find it is efficient to decompose complex questions into multiple simple steps and provide the human like thinking process as detailed as possible. Thus, we further introduce SQL skeleton as an intermediate step in CoT, which conforms to human way of thinking. Fig 3 (c) illustrates our organization process. Given the selected

Datasets	Train	Dev	Test	DB	Table/DB	Row/DB
BIRD	9,428	1,534	1,789	95	7.3	549k
Spider	8,659	1,034	2,147	200	5.1	2k

Table 1: The statistics of BIRD and Spider datasets.

few-shot question q_i, we first decompose it into sub-questions as the way of [33]. Then, we generate SQL skeleton (original SQL masked by [column_name], [table_name] and [value]), which guides LLMs to think about the structures of SQL first. Next, we instruct the LLM to select correct identifiers from schema according to the sub-question and fill the SQL skeleton to generate the SQL query. After all the sub-question solved, we finally obtain the SQL query of q_i. In conclusion, generating and filling SQL skeleton provide more detailed inference steps for CoT, which enhance LLM performance.

3 Experimental Setup

Table 1 shows the statistics of two datasets.

Datasets: **BIRD** [13] represents a pioneering, cross-domain dataset on text-to-SQL. BIRD contains over 12,751 unique question-SQL pairs, 95 big databases with a total size of 33.4 GB. **Spider** [31] is a cross-domain text-to-SQL dataset. It consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables covering 138 different domains.

Evaluation Metrics: Following BIRD [13], we utilize Execution Accuracy (EX) and Valid Efficiency Score (VES) to evaluate models.

Baselines: We use GPT-4 [17], DIN-SQL [18], DAIL-SQL [6], C3-SQL [5] and MAC-SQL [26] as the baselines.

Hyperparameters: For three retrievers, we use the popular transformers library for pre-trained BERT. As we have analyzed in section 7.1, for achieving the best retrieval effects, hyper-parameter θ & μ should be neither too large, nor too small. Thus, we use grid-search strategy to tune the hyper-parameters. We tune θ in {11,12,13,14,15,16} and μ in {1,3,5,7,9,11}, we finally obtain the best result at θ=13 and μ=5.

Error Correction: Following DIN-SQL [18], error correction module is designed to automatically correct errors after generating SQL queries, because the generated SQL usually contains certain accidental errors such as missing keywords or syntax errors. Thus, we use an error correction module to optimize the initial SQL generation results by automatically amending specific errors.

4 Results and Analysis

4.1 Overall Results

In Table 2(a), we report the performance of MR-SQL and competitive baselines on dev set of BIRD, MR-SQL achieves at least 2.54% improvement in EX and

Model	BIRD	
	EX	VES
ChatGPT + CoT	36.64	42.30
GPT-4	46.35	49.77
DIN-SQL + GPT-4	50.72	58.79
DAIL-SQL + GPT-4	54.76	56.08
MAC-SQL + GPT-4	59.39	**66.39**
MR-SQL + GPT-4	**61.93**	**66.91**

(a)

Model	Spider	
	EX (dev)	EX (test)
C3 + ChatGPT	81.80	82.30
DIN-SQL + GPT-4	82.80	85.30
DAIL-SQL + GPT-4	84.40	86.60
MAC-SQL + GPT-4	86.75	82.80
MR-SQL + GPT-4	**87.30**	**87.70**
+ Generated Evidence	**87.90**	**88.10**

(b)

Table 2: Overall performances of MR-SQL and several baselines. (a) Execution accuracy (EX) and Valid efficiency score (VES) on dev set of BIRD dataset. (b) Execution accuracy(EX) on both dev and test set of Spider.

0.52% in VES than the state-of-the-art. Table 2(b) shows the EX of MR-SQL and other baselines on dev and test set of Spider, MR-SQL achieves further 0.55% and 1.10% improvement in dev and test set respectively. The results on two datasets demonstrate the high efficiency of MR-SQL framework.

Inspired by BIRD [13], external knowledge evidence is helpful for mapping the natural language instructions into counterpart database values. Thus, we propose an approach to generate evidence for Spider by utilizing GPT-4. With the generated extra evidence, MR-SQL reaches the new state of the art by at least 1.15% on the development set and by 1.50% on the test set, which proves the effectiveness of our generation approach.

4.2 Ablation Study

To study the impact of the modules in MR-SQL, we evaluate it by conducting a set of ablation studies. Row(1) represents the experiment results of the whole MR-SQL framework with GPT-4, while in the following rows, we start with GPT-4 and add Table-Retriever & Column-Retriever, SQL skeleton organization, Example-Retriever and error correction module row by row to compare the efficacy of each module in MR-SQL framework. For comparison, the last row(6) represent the whole MR-SQL. The results are shown in Table 3. In conclusion, the ablation study proves that all the modules in MR-SQL framework play important roles for performance enhancement. Compared with GPT-4, the MR-SQL make further improvement by 15.58% in EX and 17.14% in VES.

5 Discussion of Hyper-parameters

Here we study how θ & μ influence the performance of Table-Retriever and Column-Retriever on the development set of BIRD. Figure 4 (a) illustrates the trade-off between recall and reduction ratio for the Table-Retriever when the threshold θ is adjusted (fine statistic means the recall of gold tables, while coarse statistic means the recall of all gold tables for each question). Specifically, with the growth of θ, the reduction ratio of invalid tables increase, but the recall

Method	BIRD	
	EX	VES
(1) MR-SQL + GPT-4	**61.93**	**66.91**
(2) GPT-4	46.35 (↓ 15.58)	49.77 (↓ 17.14)
(3) + Table-Retriever & Column-Retriever	55.74 (↓ 6.19)	58.65 (↓ 8.26)
(4) + Example-Retriever	58.02 (↓ 3.91)	61.57 (↓ 5.34)
(5) + SQL Skeleton (CoT)	60.63 (↓ 1.30)	64.72 (↓ 2.19)
(6) + Error correction	**61.93** (↓ 0.00)	**66.91** (↓ 0.00)

Table 3: Results of ablation study on BIRD. "+" means adding module on the basis of the previous row.

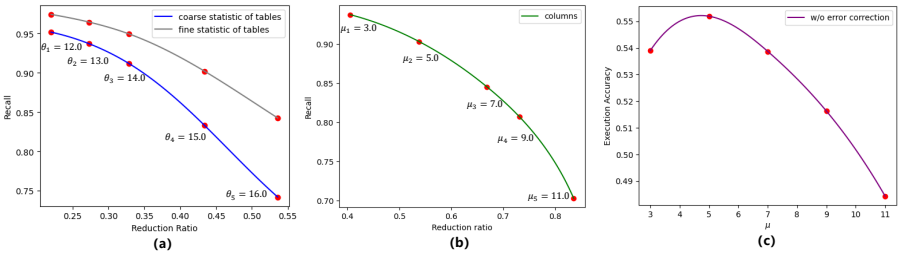

Fig. 4: (a) Recall and reduction ratio with different θ in Table-Retriever. (b) Recall and reduction ratio with different μ in Column-Retriever. (c) Execution accuracy of LLM with different μ while the θ is fixed.

of gold tables decrease. Similarly, as shown in Figure 4 (b), we fix θ=13.0 and modify μ. With the growth of μ, the reduction ratio of invalid column increase, while the recall of gold columns decrease. The appearance demonstrates that a higher confidence threshold may filter out both invalid and gold tables/columns, which will lead to a decrease in recall and an increase in reduction ratio.

Furthermore, we design a set of experiments to explore how confidence threshold of Table-Retriever and Column-Retriever influence the final performance of LLM. Here we use μ in Column-Retriever as the representative. Figure 4 (c) shows the execution accuracy of LLM with the tuning of μ while the θ is fixed, the settings of θ and μ is the same as Figure 4 (b). In order to study the impact for LLM clearly, we experiment without post-processing error correction module. We can easily find the execution accuracy first increase and then decrease with the growth of μ. As shown in table, when μ=5.0, we get the best LLM performance. The results indicates Table-Retriever and Column-Retriever with too small θ and μ may not decrease invalid tables and columns adequately, while too large θ and μ may lead to low recall of gold tables and columns.

6 Conclusion

In this paper, we propose a multi-level retrieval-based framework (MR-SQL) by constructing efficient SQL generation prompt to improve the LLMs' reasoning performance. We design three independent retrieval-based models to al-

leviate the drawback of redundant tables and columns which cause excessive redundancy, and retrieve similar samples for few-shot example selection. Then, we also introduce SQL skeleton in example organization to achieve more fine-grained SQL generation process. Through comprehensive experiments, the results demonstrate the effectiveness of retrieving and filtering valid information in advance for constructing LLM's prompt engineering, and the rationality of using skeleton to guide the correct SQL generation.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 62276017, 62406033, U1636211, 61672081), and the State Key Laboratory of Complex& Critical Software Environment (Grant No. SKLCCSE-2024ZX-18).

References

1. Bai, J., et al.: Qwen technical report. arXiv preprint arXiv:2309.16609 **abs/2309.16609** (2023), https://arxiv.org/abs/2309.16609
2. Chai, L., Liu, S., Yang, J., Yin, Y., Jin, K., Liu, J., Sun, T., Zhang, G., Ren, C., Guo, H., et al.: Mceval: Massively multilingual code evaluation. arXiv preprint arXiv:2406.07436 (2024)
3. Chang, S., Fosler-Lussier, E.: How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. CoRR **abs/2305.11853** (2023)
4. Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., Li, L., Sui, Z.: A survey for in-context learning. CoRR **abs/2301.00234** (2023)
5. Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Chen, L., Lin, J., Lou, D.: C3: zero-shot text-to-sql with chatgpt. CoRR **abs/2307.07306** (2023)
6. Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J.: Text-to-sql empowered by large language models: A benchmark evaluation. Proc. VLDB Endow. **17**(5), 1132–1145 (2024)
7. Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J., Liu, T., Zhang, D.: Towards complex text-to-sql in cross-domain database with intermediate representation. In: Korhonen, A., Traum, D.R., Màrquez, L. (eds.) ACL 2019. ACL (2019)
8. He, P., Mao, Y., Chakrabarti, K., Chen, W.: X-SQL: reinforce schema representation with context. CoRR **abs/1908.08113** (2019)
9. Hui, B., Geng, R., Wang, L., Qin, B., Li, Y., Li, B., Sun, J., Li, Y.: S^2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers. In: ACL 2022, (2022)
10. Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over BERT. In: Huang, J.X., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J., Liu, Y. (eds.) SIGIR 2020. ACM (2020)
11. Li, H., Zhang, J., Li, C., Chen, H.: RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql. In: Williams, B., Chen, Y., Neville, J. (eds.) Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI (2023)
12. Li, J., Hui, B., Cheng, R., Qin, B., Ma, C., Huo, N., Huang, F., Du, W., Si, L., Li, Y.: Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In: EAAI 2023 (2023)

13. Li, J., et al.: Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In: NeurIPS 2023 (2023)
14. Li, X., et al.: Tele-flm technical report. CoRR **abs/2404.16645** (2024)
15. Li, Z., Wu, Z., Li, M., He, Z., Fang, R., Zhang, J., Zhao, Y., Li, Y., Li, Z., Song, S.: Scalable database-driven kgs can help text-to-sql. In: Proceedings of the ISWC 2024 Posters, Demos and Industry Tracks: (2024)
16. Liu, S., Peng, C., Wang, C., Chen, X., Song, S.: icsberts: Optimizing pre-trained language models in intelligent customer service. In: INNS DLIA@IJCNN (2023)
17. OpenAI: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023), https://arxiv.org/abs/2303.08774
18. Pourreza, M., Rafiei, D.: DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) NeurIPS 2023 (2023)
19. Qin, B., Hui, B., Wang, L., Yang, M., Li, J., Li, B., Geng, R., Cao, R., Sun, J., Si, L., Huang, F., Li, Y.: A survey on text-to-sql parsing: Concepts, methods, and future directions. CoRR **abs/2208.13629** (2022)
20. Rozière, B., et al.: Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950 (2023), https://arxiv.org/abs/2308.12950
21. Shao, J., Li, X.: Ai flow at the network edge. IEEE Network pp. 1–1 (2025). https://doi.org/10.1109/MNET.2025.3541208
22. Sun, R., et al.: Sql-palm: Improved large language model adaptation for text-to-sql. CoRR **abs/2306.00739** (2023)
23. Sun, T., Chai, L., Yang, J., Yin, Y., Guo, H., Liu, J., Wang, B., Yang, L., Li, Z.: Unicoder: Scaling code large language model via universal code. arXiv preprint arXiv:2406.16441 (2024)
24. Tai, C., Chen, Z., Zhang, T., Deng, X., Sun, H.: Exploring chain of thought style prompting for text-to-sql. In: EMNLP 2023 (2023)
25. Wang, B., Shin, R., Liu, X., Polozov, O., Richardson, M.: RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. In: ACL 2020 (2020)
26. Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Zhang, Q., Yan, Z., Li, Z.: MAC-SQL: A multi-agent collaborative framework for text-to-sql. CoRR **abs/2312.11242** (2023)
27. Wang, Z., Yao, Y., Mengxiang, L., He, Z., Wang, C., Song, S., et al.: Telechat: An open-source billingual large language model. In: Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10). pp. 10–20 (2024)
28. Wang, Z., et al.: Telechat technical report. CoRR **abs/2401.03804** (2024)
29. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: NeurIPS 2022 (2022)
30. Wu, H., Chen, Y., Liu, L., Chen, T., Wang, K., Lin, L.: Sqlnet: Scale-modulated query and localization network for few-shot class-agnostic counting. CoRR **abs/2311.10011** (2023)
31. Yu, T., et al.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: EMNLP 2018. ACL (2018)
32. Zheng, Y., Wang, H., Dong, B., Wang, X., Li, C.: HIE-SQL: history information enhanced network for context-dependent text-to-sql semantic parsing. In: Findings of the Association for Computational Linguistics: ACL 2022, (2022)
33. Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q.V., Chi, E.H.: Least-to-most prompting enables complex reasoning in large language models. In: ICLR 2023. OpenReview.net (2023)