R^3 : "This is My SQL, Are You With Me?" A Consensus-Based Multi-Agent System for Text-to-SQL Tasks

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have demonstrated strong performance on various tasks. To unleash their power on the Text-to-SQL task, we propose R^3 (Review-Rebuttal-Revision), a consensus-based multi-agent system for Text-to-SQL tasks. R^3 outperforms the existing single LLM Text-to-SQL systems as well as the multi-agent Text-to-SQL systems by 1.3% to 8.1% on Spider and Bird. Surprisingly, we find that for Llama-3-8B, R^3 outperforms chain-of-thought prompting by over 20%, even outperforming GPT-3.5 on the development set of Spider.

1 Introduction

Text-to-SQL, the task of converting natural language to SQL queries, enables non-technical users to access databases with natural language (Deng et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023). Recently, Large Language Models (LLMs) have made significant progress on various tasks (Touvron et al., 2023; OpenAI, 2023).

Although researchers have proposed various methods to enhance the reasoning abilities of LLMs (Wei et al., 2022; Yao et al., 2023; Besta et al., 2024), However, they are still facing challenges with Text-to-SQL tasks (Li et al., 2023; Hong et al., 2024). The LLM-based multi-agent system leverages collective intelligence from a group of LLMs and have achieved exceptional performance across various tasks (Park et al., 2023; Hong et al., 2023; Xu et al., 2023), but little work explores using them on Text-to-SQL. The existing multi-agent Text-to-SQL system first decomposes the task into multiple subtasks which are then accomplished step-by-step by agents (Wang et al., 2023). While achieving remarkable performances, such a decomposition-based system necessitates extensive manual prompt engineering and logic design.

We propose R^3, a consensus-based multi-agent system for Text-to-SQL tasks. The proposed system draws inspiration from the peer-review mechanism, featuring one agent as the SQL-writer and several reviewers automatically generated by the LLM. Once the generated SQL query is tested to be executable, the system will step into a review process, where we use the execution results to guide the SQL-writer and reviewers to refine the SQL. Through rounds of "review", "negotiation or rebuttal", and "revision", SQL-writer and reviewers will finally achieve consensus and deliver a solution with collective agreement (see Figure 1).

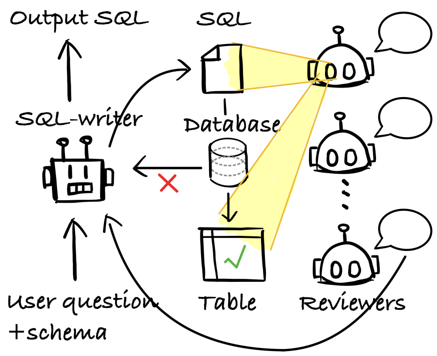

Figure 1: R^3 Architecture. n reviewer agents, each with distinct characteristics, are created to review the generated SQL and its execution result. The process continues until the master node (SQL-writer) and the other nodes reach a consensus, at which point the system outputs the final SQL.

We test R^3 on the popular Spider and Bird benchmarks. R^3 outperforms the existing single LLM as well as the multi-agent Tex-to-SQL systems by 1.3% to 8.1% on Spider and Bird. Surprisingly, we find that for Llama-3-8B, R^3 outperforms chain-of-thought prompting by over 20%, even outperforming GPT-3.5 on the Spider-Dev set. Our contributions can be summarized as follows:

1. To the best of our knowledge, R^3 is the first Text-to-SQL system to use the execution result for SQL refinements, and the first Text-to-SQL system to equip agents with memory sequences

to enhance SQL generation.

2. ***R***[3] offers a consensus-based multi-agent system for Text-to-SQL tasks. Using very succinct prompts, it achieves strong performance compared to other systems. In addition, it effectively helps open-source LLMs such as Llama-3-8B on SQL generation.

3. We provide a detailed error analysis of ***R***[3] on the existing Text-to-SQL benchmarks, shedding light on future research on the Text-to-SQL task.

2 Architecture

SQL-Writer (SW). We task SW agents to: (1) compose the original SQL query based on the user question and database schema; (2) ensure that the SQL query is executable, and correct it when errors occur; (3) respond to reviewer agents' feedback and revise the SQL query accordingly. Specifically, we prompt SW agent through Prompt 1 in Appendix A.9. For task (1), we feed the Prompt 1 to SW agent directly. Given a user question x and the database schema $\mathcal{S}$, task (1) can be formalized as:

$$y = \texttt{LLM}(x, \mathcal{S}),$$

where y is the generated SQL query. For (2) and (3), we maintain a dialogue history $\mathcal{H}$ initially set to $\mathcal{H} = [(x, \mathcal{S}), y]$. Specifically, if an error e occurs, we append e to the history $\mathcal{H} \leftarrow \mathcal{H} + e$ and get y' through:

$$y' = \texttt{LLM}(\mathcal{H}).$$

We then concatenate y' with the history $\mathcal{H} \leftarrow \mathcal{H} + y'$. In addition, considering the length limitation of LLMs' context window, we truncate the history $\mathcal{H}$ when the prompt is longer than the context limit.

Reviewers (REs). We generate the reviewer agent's professions using an LLM (see Prompt 3 in Appendix A.9) based on the database schema and the content of the SQL query, for instance, "Senior Database Engineer specialized in writing various clauses" and "Data Analyst in the automotive industry", etc. We incorporate these professions in the system prompt for the reviewer agent to make them focus on different aspects of the SQL query. These reviewer agents are prompted to provide their professional comments based on the database schema, the user's question, the predicted SQL, and its execution result in the table format.

Overall Architecture. After several rounds of "negotiation" between the SQL-writer and reviewer agents, we decide whether there is a consensus by checking if the SQL-writer agent generates the same SQL query as in the previous round. When there is a consensus, we terminate the negotiation loop and output the final SQL query. Algorithm 1 depicts the overall process of our system.

Given x (user question), $\mathcal{S}$ (schema)
$y = \texttt{LLM}(x, \mathcal{S})$
$\mathcal{H} = [(x, \mathcal{S}), y]$
$i = 0$
$j = 0$
while $i <=$ MaxReviewTurns **do**
    while $j <=$ MaxDebugTurns **do**
        Try:
            $\mathcal{T} = \texttt{Database}(y)$
            break
        Except Exception as e:
        $j \leftarrow j + 1$
            $\mathcal{H} \leftarrow \mathcal{H} + e; \quad y' = \texttt{LLM}(\mathcal{H})$
            $\mathcal{H} \leftarrow \mathcal{H} + y'$
    end
    $r = \texttt{LLM}(x, \mathcal{S}, y, \mathcal{T})$
    $\mathcal{H} \leftarrow \mathcal{H} + r; \quad y'' = \texttt{LLM}(\mathcal{H})$
    $\mathcal{H} \leftarrow \mathcal{H} + y''$
    if $y == y''$ **then**
        break
    else
        $y \leftarrow y''$
    end
    $i \leftarrow i + 1$
end

Algorithm 1: *R*[3]-Loop

Appendix A.9 provides the detailed prompts we use in ***R***[3]. In addition, we incorporate:

1. Program of Thoughts (PoT) (Chen et al., 2023) to prompt the SQL-writer agent to generate Python code before SQL query (see Prompt 2 in Appendix A.9). Therefore, the agents may leverage Python in their reasoning process for better SQL query generation.

2. k-shots example selection based on similarity of the user question embeddings. Specifically, when our system infers the SQL query in the test set, we select the k most similar use questions and their corresponding SQL queries from the training set (k-shots) and use them for in-context learning.

3 Experiments and Results

We conduct experiments on two cross-domain Text-to-SQL benchmarks, Spider and Bird detailed in Table 5 in Appendix A.1. We employ test-suite execution evaluation[1] (Zhong et al., 2020), the standard evaluation protocol for Spider, and the official SQL

[1]github.com/taoyds/test-suite-sql-eval

Model	Method	SD	ST	BD
GPT-3.5	- (Li et al., 2023)	72.1	-	37.22
	C3 (Dong et al., 2023)	81.8	82.3	-
	MAC (Wang et al., 2023)	80.6	75.5	50.56
	$\boldsymbol{R}^3$ 5-shot	81.4	81.1	52.15
GPT-4	DAIL (Gao et al., 2023)	83.6	86.6	-
	PET (Li et al., 2024)	82.2	87.6	-
	DIN (Pourreza and Rafiei, 2023)	82.8	85.3	50.72
	MAC (Wang et al., 2023)	86.8	82.8	59.39
	$\boldsymbol{R}^3$ 5-shot	**88.1**	**89.9**	**61.80**

Table 1: Execution accuracy across various models and methods. We use the GPT-3.5-Turbo in our experiment. "SD", "ST", "BD" represent Spider-Dev, Spider-Test, Bird-Dev, respectively. For detailed description of baseline methods mentioned above, see Appendix A.2.

Model	Method	SD	ST
GPT-3.5	- (Li et al., 2023)	72.1	-
Llama-3-8B	CoT	52.1	53.5
	$\boldsymbol{R}^3$ 0-shot	72.8	72.6
Llama-3-70B	$\boldsymbol{R}^3$ 0-shot	79.7	80.3

Table 2: Execution accuracy comparison when we equip Llama-3 models with $\boldsymbol{R}^3$on Spider-Dev ("SD") and Spider-Test ("ST").

	GPT-3.5-Turbo		GPT-4	
	Spider	Bird	Spider	Bird
CoT	78.2	37.22	79.7	53.30
PoT	78.5	36.96	80.0	54.61
1R-Lp + CoT	78.3	44.13	82.3	57.89
1R-Lp + PoT	79.3	46.35	85.4	58.34
$\boldsymbol{R}^3$: 3R-Lp + PoT	**81.4**	**52.15**	**88.1**	**61.80**

Table 3: Ablation Studies on Spider-Dev and Bird-Dev (Execution Accuracy). The 1-Reviewer Loop (1R-Lp) represents that only one reviewer agent participates in the discussion, while the 3-Reviewers Loop (3R-Lp) represents three in the dicussion, which is also the default configuration of $\boldsymbol{R}^3$. We conduct all the experiments here under the 5-shot setting.

execution accuracy evaluation for Bird[2]. Table 1 compares $\boldsymbol{R}^3$'s performance with existing baseline methods when we employ different foundation LLMs. Our best performed system achieves 88.1%, 89.9%, and 61.8% on the Spider-Dev, Spider-Test, and Bird-Dev respectively, surpassing the existing multi-agent Text-to-SQL systems. In addition, we test our system with open-source Llama-3 models on Spider and report the results in Table 2. To our surprise, with the help of $\boldsymbol{R}^3$, zero-shot Llama-3-8B outperforms GPT-3.5 performance reported by Li et al. (2023) on Spider-Dev set. This demonstrates the effectiveness of our proposed $\boldsymbol{R}^3$ system.

3.1 Ablation Studies

We conduct an ablation study on the impact of CoT, PoT with one or three reviewer agents in the discussion and report the results in Table 3. The results in Table 3 show that the n-Reviewer(s) Loop (nR-Lp) plays a major role in performance improvement, with the 3R-Lp configuration significantly outperforming the 1R-Lp setup. The proposed $\boldsymbol{R}^3$ system achieves a 10.54% improvement over the baseline GPT-4 + CoT. We provide the statistical significant test for these results in Appendix A.4. Appendix A.3 provides a sensitivity analysis of the impacts of the k value in k-shots.

We conducted case studies on 244 instances from the Spider-Dev dataset where the CoT fail but $\boldsymbol{R}^3$succeed when combined with Llama-3-8B. The findings are as follows:

1. **Corrected non-executable SQL queries, 51%.** The LLM equipped with memory module (see Section 2) excels at correcting non-executable SQL queries.

2. **Refinement based on reviewers' comments, 44%.** The nR-Lp functions as an enhanced Self-Consistency (SC) (Wang et al., 2022). On the one hand, it avoids the hallucinations caused by high temperatures (Renze and Guven, 2024). On the other hand, the nR-Lp considers feedback from all agents, unlike the voting process in SC, which consistently disregards minority opinions.

3. **Refinement based on the output table, 27.5%.** LLMs may not experts in SQL writing, but they are full-skilled data readers. The information that reviewers observe from execution results greatly assists the SQL-writer in refining the SQL.

3.2 Error Analysis

In total, GPT-4+$\boldsymbol{R}^3$ fails to generate the gold SQL queries for 123 instances in Spider-Dev. Table 4 shows the error case distribution for our system on Spider-Dev (more in Appendices A.6 and A.7). Note that though we have spotted issues with the

[2]bird-bench.github.io/

Error Types	Question, Gold & Prediction	Explanation
Gold Error (30.5%)	**Q:** What are the Asian countries which have a population **larger than that of any country in Africa**? **Gold:** ❌ … `AND population > (SELECT min(population) FROM country WHERE Continent = "Africa")` **Pred:** ✅ … `AND population > (SELECT max(population) FROM country WHERE Continent = "Africa")`	Judged as incorrect because of the incorrect gold SQL query.
Logic (29.8%)	**Q:** How many owners **temporarily do not** have any dogs? **Gold:** ✅ `SELECT count(*) FROM Owners WHERE owner_id NOT IN (SELECT owner_id FROM Dogs)` **Pred:** ❌ `SELECT (SELECT COUNT(DISTINCT owner_id) FROM Owners) - (SELECT COUNT(DISTINCT owner_id) FROM Dogs WHERE date_departed IS NULL)`	The predicted SQL query wrongly assumes that all owners have had dogs.
Ambiguity (13.2%)	**Q:** What are the **names** of all makers with more than 3 models? **Gold:** ✅ `SELECT T1.FullName ... HAVING count(*) > 3;` **Pred:** ✅ `SELECT T1.Maker ... HAVING count(*) > 3;`	Both `FullName` and `Maker` columns hold the information for “names”.
Inaccuracy (11.3%)	**Q:** What are the **arriving date** of the dogs who have gone through a treatment? **Gold:** ✅ `SELECT T1.date_arrived, FROM ...` **Pred:** ❌ `SELECT T1.date_arrived, T1.Name FROM ...`	The selected `Name` is not asked by the question.
DB Value (10.6%)	**Q:** Which city and country is the **Alton** airport at? **Gold:** ✅ `SELECT ... WHERE AirportName = "Alton" ;` **Pred:** ❌ `SELECT ... WHERE AirportName LIKE "%Alton%" ;`	Our framework notices there is a space for `Alton` in the DB, therefore employing a fuzzy match.
Others (4.6%)		

Table 4: Error Analysis of $\boldsymbol{R}^3$ on Spider-Dev. We make the part in the question red when it is either annotated incorrectly in the gold SQL query (Gold) or predicted incorrectly in the predicted SQL query (Pred).

gold SQL queries, we still adopt the original set to calculate the performance of our system to ensure a fair comparison.

Gold Error. We notice that though the annotation quality of Spider is good, there are still cases where the gold SQL queries are not correct. Specifically, among the 151 examples, 30.5% are due to incorrect gold SQL queries (4.5% of all the examples in Spider-Dev). To facilitate future research, we catalog the instances with incorrect gold SQL, correct the errors, and share the details[3].

Ambiguity. We observe that there are a few questions involving ambiguities, a phenomenon spotted on a wide range of NLP tasks (Plank, 2022; Deng et al., 2023). In Table 4.3, both `FullName` and `Maker` columns hold the information for the “name of makers”, except that `FullName` holds the full names while `Maker` holds the name abbreviations. Therefore, both the gold and predicted SQL queries should be considered correct if there is no further clarifications. Such ambiguous requests may be common in real-world applications as the lay users may not be familiar with the database schema. This requires future research on interactive Text-to-SQL systems that can understand and deal with such ambiguities in user questions.

[3] visible-after-review.com

Dirty Database Value. We observe that due to the Database (DB) setup for Spider, certain DB values may deviate from what is asked in the question. For instance, in Table 4.5, $\boldsymbol{R}^3$ notices a space for `Alton` in DB, therefore employing a fuzzy match. But this deviates the SQL query's execution results from the gold SQL query's results.

Explanations of “Logic” and “Inaccuracy” errors can be found in Appendix A.5. Our findings indicate that the existing evaluation protocols for Text-to-SQL generation may not authentically capture the capabilities of these sophisticated systems. Therefore, we advocate for a reassessment and enhancement of Text-to-SQL evaluation methods. We provide further error analysis of $\boldsymbol{R}^3$ on Bird in Appendix A.7.

4 Conclusion

$\boldsymbol{R}^3$ significantly enhance the performances of LLMs on the Text-to-SQL task. We conduct a comprehensive error analysis and identify persistent issues with the current Text-to-SQL evaluation. This underscores the necessity for our community to develop a refined evaluation protocol that more effectively captures nuances in SQL generation and accurately reflects model performance.

Limitations

Due to the scope of the study, we only test a limited number of LLMs. The performance gap between 1R-Lp and 3R-Lp demonstrates that the number of reviewers is a worthwhile topic of research. However, this work does not delve into this much.

Ethical Statements

In this paper, we propose strategies to improve the SQL generation capabilities of LLMs. To the best of our knowledge, we do not expect our system would have negative impacts on the society.

References

AI@Meta. 2024. Llama 3 model card.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-SQL: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2166–2187, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Naihao Deng, Xinliang Zhang, Siyang Liu, Winston Wu, Lu Wang, and Rada Mihalcea. 2023. You are what you annotate: Towards better models through annotator representations. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12475–12498, Singapore. Association for Computational Linguistics.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

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

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv preprint arXiv:2406.08426*.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023. Can llm already serve as a database interface. *A big bench for large-scale database grounded text-to-sqls. CoRR abs/2305.03111*.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, et al. 2024. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv preprint arXiv:2403.09732*.

Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.

R OpenAI. 2023. Gpt-4 technical report. *arXiv*, pages 2303–08774.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Barbara Plank. 2022. The "problem" of human label variation: On ground truth in data, modeling and evaluation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10671–10682, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Matthew Renze and Erhan Guven. 2024. The effect of sampling temperature on problem solving in large language models. *arXiv preprint arXiv:2402.05201*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. 2023. Language agents with reinforcement learning for strategic play in the werewolf game. *arXiv preprint arXiv:2310.18940*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suites. *arXiv preprint arXiv:2010.02840*.

A Appendix

A.1 Dataset Descriptions

	Spider-Dev	Spider-Test (Yu et al., 2018)	Bird-Dev (Li et al., 2023)
#QA	1,034	2147	1,534
#Domain	138	-	37
#DB	200	206	95
DB Size	879.5 MB	906.5 MB	1.76 GB

Table 5: Statistics of two Text-to-SQL benchmarks we use in our experiments. "#QA", "#Domain" and "#DB" refer to the number of samples, domains and databases, respectively.

A.2 Baseline

Experiments in this work was based on LLMs including GPT-3.5-Turbo, GPT-4 (OpenAI, 2023) and Llama-3 (AI@Meta, 2024). As for the compared methods, the raw performance for GPT-3.5 ("-") was evaluated by Li et al. (2023); C3 employs schema linking filtering (Dong et al., 2023); DAIL selects few-shot demonstrations based on their skeleton similarities (Gao et al., 2023), and "SC" represents Self-Consistency (Wang et al., 2022); PET uses cross-consistency (Li et al., 2024); DIN decomposes the text-to-SQL task into smaller subtasks (Pourreza and Rafiei, 2023); MAC, as previously mentioned, is the first to apply a Multi-Agent system to Text-to-SQL tasks (Wang et al., 2023).

A.3 Effects of k in k-shot.

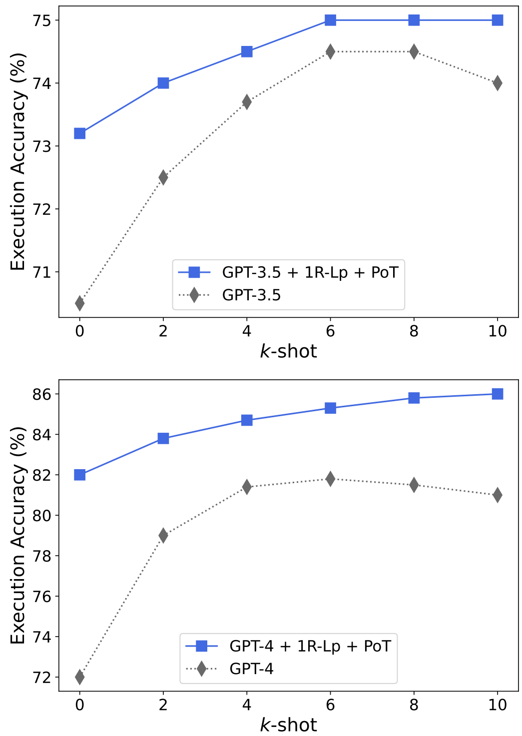

Figure 2: k-shot Sensitivity Analysis.

We test various k values on 200 random samples from Spider-Dev. As shown in Figure 2, compared to CoT, the performance of the $\boldsymbol{R}^3$ system remains relatively stable regardless of the number of examples, which corroborates our previous findings from the 0-shot experiments with Llama-3.

A.4 Significance Test

We divided the generated SQL by several strategies in Table 3 into 10 equal parts and calculated the execution accuracy for each. To test whether our strategy can indeed improve execution accuracy, we conduct a significance test between the "CoT" and "3R-Lp+PoT" strategies. The null hypothesis of the test is that the median execution accuracy obtained by the two strategies is the same. The Mann-Whitney U Test (Mann and Whitney, 1947) is a non-parametric statistical method used to compare whether there is a significant difference in the medians of two independent samples. Compared to the Analysis of Variance (ANOVA), it does not require the data to be normally distributed, making it suitable for small samples or data with unknown distribution.

The p-value of the test is 0.0024, which is below the commonly accepted significance level of 0.05. Therefore, we have reason to reject the null hypothesis, indicating that the "3R-Lp+PoT" strategy leads to a significant performance improvement.

A.5 Additional Error Types

Logic. In Table 4.2, we present an example of the logic error made by $\boldsymbol{R}^3$. We notice that LLMs may solve the problems using a more complicated logic, which is prone to mistakes. For instance, in Table 4.2, instead of directly counting the owners who do not own dogs, the LLMs try to substract the number of dog owners from the total number of owners. This ignores the possibility that some owners may have never had any dogs before. This addresses an issue with the multi-agent system that if the system comes up with a complicated initial SQL query, the following discussion process may try to polish the complicated SQL query instead of switching to an easier solution. In cases like Table 4.2, there is no way to reach a perfect SQL query with the substraction logic.

Inaccuracy. We observe that the LLMs may incorporate more information than what is asked by the end user. For instance, in Table 4.4, the user does not ask for the name of the dogs but the LLMs present such information along with the asked arriving date. We hypothesize that since such extra information can potentially be helpful to the end user, LLMs may be biased towards including it.

A.6 Spider Error Cases

Error Type	Question, Gold & Prediction	Reason
DB Value	**Q:** Find the last name of the students who currently live in the state of North Carolina but have not registered in any degree program. **Gold:** `SELECT ... WHERE T2.state_province_county = 'NorthCarolina' EXCEPT ...` **Pred:** `SELECT ... WHERE T2.state_province_county = 'North Carolina' EXCEPT ...`	The filtering condition in the question does not match the database value, string "NorthCalifornia" in database do not have a space in between.
Gold Error	**Q:** What are the first names of all players, and their average rankings? **Gold:** `SELECT avg(ranking), T1.first_name FROM players AS T1 JOIN rankings AS T2 ON T1.player_id = T2.player_id GROUP BY T1.first_name` **Pred:** `SELECT avg(ranking), T1.first_name FROM players AS T1 JOIN rankings AS T2 ON T1.player_id = T2.player_id GROUP BY T1.player_id`	The individuals in the table can be uniquely determined by column player_id not first_name, when `GROUP BY`.
Gold Error	**Q:** Find the id and cell phone of the professionals who operate two or more types of treatments. **Gold:** `SELECT T1.professional_id, T1.cell_number FROM Professionals AS T1 JOIN Treatments AS T2 ON T1.professional_id = T2.professional_id GROUP BY T1.professional_id HAVING count(*) >= 2` **Pred:** `SELECT T1.professional_id, T1.cell_number FROM Professionals AS T1 JOIN Treatments AS T2 ON T1.professional_id = T2.professional_id GROUP BY T1.professional_id HAVING COUNT( DISTINCT T2.treatment_type_code) >= 2`	The gold only finds professionals who have two or more records in the treatment table does not ensure that the records are for different types of treatments
Ambiguity	**Q:** What are the names and ids of all makers with more than 3 models? **Gold:** `SELECT T1.FullName , T1.Id FROM CAR_MAKERS AS T1 JOIN MODEL_LIST AS T2 ON T1.Id = T2.Maker GROUP BY T1.Id HAVING count(*) > 3;` **Pred:** `SELECT T1.Maker , T1.Id FROM CAR_MAKERS AS T1 JOIN MODEL_LIST AS T2 ON T1.Id = T2.Maker GROUP BY T1.Id HAVING count(*) > 3;`	Both column "Maker" and column "FullName" can answer the question about the "names of makers" in the query.
Imprecise	**Q:** What are the arriving date and the departing date of the dogs who have gone through a treatment? **Gold:** `SELECT DISTINCT T1.date_arrived, T1.date_departed FROM Dogs AS T1 JOIN Treatments AS T2 ON T1.dog_id = T2.dog_id` **Pred:** `SELECT DISTINCT T1.date_arrived, T1.date_departed, T1.Name FROM Dogs AS T1 JOIN Treatments AS T2 ON T1.dog_id = T2.dog_id`	The question do not require listing the specific names of the students, but only ask to list the students' arrival and departure dates. This falls under information redundancy.

Table 6

A.7 BIRD Error Cases

Error Type	Question, Gold & Prediction	Reason
DB Value	**Q:** How many cards with unknown power that can't be found in foil is in duel deck A? **Gold:** `SELECT SUM(CASE WHEN power LIKE '%*%' OR power IS NULL THEN 1 ELSE 0 END) FROM cards WHERE hasFoil = 0 AND duelDeck = 'a'` **Pred:** `SELECT COUNT(*) FROM cards WHERE (power IS NULL OR power = '*') AND hasFoil = 0 AND duelDeck = 'a'`	Values in database cannot exact match with the question and evidence.
Gold Error	**Q:** How many artists have designed a card with a black border color and is available in both "arena" and "mtgo" printing type? **Gold:** `SELECT COUNT(CASE WHEN availability LIKE '%arena,mtgo%' THEN 1 ELSE NULL END) FROM cards` **Pred:** `SELECT COUNT(DISTINCT artist) FROM cards WHERE borderColor = 'black' AND availability LIKE '%arena,mtgo%'`	The gold missed one filtering condition.
Ambiguity	**Q:** Among black card borders, which card has full artwork? **Gold:** `SELECT id FROM cards WHERE borderColor = 'black' AND isFullArt = 1` **Pred:** `SELECT name FROM cards WHERE borderColor = 'black' AND isFullArt = 1;`	Both column "name" and column "id" can answer the question about the "the card has full artwork" in the query.
Logic	**Q:** Which cities have the top 5 lowest enrollment number for students in grades 1 through 12? **Gold:** `SELECT T2.City FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode GROUP BY T2.City ORDER BY SUM(T1.`Enrollment (K-12)`) ASC LIMIT 5` **Pred:** `SELECT T2.City, T1."Enrollment (K-12)" FROM frpm AS T1 JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode ORDER BY T1."Enrollment (K-12)" ASC LIMIT 5;`	The Pred query is incorrect because it lacks grouping and aggregation. It directly selects cities and their corresponding "Enrollment (K-12)" without summing up the enrollment numbers across all schools within each city. As a result, it returns the cities of the five schools with the lowest enrollment numbers, which does not accurately answer the question seeking the top 5 cities with the lowest total enrollment for students in grades 1 through 12.

Table 7

A.8 Interactive Correction Cases

Before	After	Reason
Q: What is the phone number of the school that has the highest number of test takers with an SAT score of over 1500?		

Before:
`SELECT schools.Phone`
`FROM satscores`
`JOIN schools ON`
`satscores.cds =`
`schools.CDSCode`
`WHERE satscores.NumGE1500`
`> 1500`
`GROUP BY schools.School`
`ORDER BY COUNT(*) DESC`
`LIMIT 1;`
	After:
	`SELECT schools.Phone`
	`FROM satscores`
	`JOIN schools ON`
	`satscores.cds =`
	`schools.CDSCode`
	`ORDER BY`
	`satscores.NumGE1500 DESC`
	`LIMIT 1;`
		The WHERE clause is checking if the number of test takers with an SAT score over 1500 is greater than 1500, which is not what the question is asking. The question is asking for the school with the highest number of test takers who scored over 1500. Additionally, the GROUP BY clause is grouping by the school name, which may not be unique. It would be better to group by the school code (CDSCode) which is unique for each school.

Q: List the top five schools, by descending order, from the highest to the lowest, the most number of Enrollment (Ages 5-17). Please give their NCES school identification number.		

Before:
`SELECT T1."School Name",`
`T2.NCESSchool`
`FROM frpm AS T1`
`JOIN schools AS T2 ON`
`T1.CDSCode = T2.CDSCode`
`ORDER BY T1."Enrollment`
`(Ages 5-17)" DESC`
`LIMIT 5;`
	After:
	`SELECT T2.NCESSchool`
	`FROM frpm AS T1`
	`JOIN schools AS T2 ON`
	`T1.CDSCode = T2.CDSCode`
	`ORDER BY T1."Enrollment`
	`(Ages 5-17)" DESC`
	`LIMIT 5;`
		The question asks for the top five schools with the most number of Enrollment (Ages 5-17), and their NCES school identification number. However, the SQL query provided returns the school name, which is not required by the question.

Q: find the package option of the tv channel that do not have any cartoon directed by Ben Jones.		

Before:
`SELECT`
`TV_Channel.Package_Option`
`FROM TV_Channel`
`LEFT JOIN Cartoon ON`
`TV_Channel.id =`
`Cartoon.Channel`
`WHERE Cartoon.Directed_by`
`IS NOT 'Ben Jones';`
	After:
	`SELECT`
	`TV_Channel.Package_Option`
	`FROM TV_Channel`
	`WHERE TV_Channel.id NOT IN`
	`(`
	`SELECT Cartoon.Channel`
	`FROM Cartoon`
	`WHERE Cartoon.Directed_by`
	`= 'Ben Jones');`
		The current query is selecting all channels where the director is not Ben Jones, but it should exclude channels that have any cartoons directed by Ben Jones. This can be achieved by using a subquery to find all channels that have cartoons directed by Ben Jones, and then excluding these channels in the main query. Here is the corrected SQL:

Q: List the earnings of poker players in descending order.		

Before:
`SELECT * FROM poker_player`
`ORDER BY Earnings DESC;`
	After:
	`SELECT Earnings FROM`
	`poker_player ORDER BY`
	`Earnings DESC;`
		The question only asks for the earnings of poker players. Therefore, we should only select the 'Earnings' column to make the result more concise and relevant to the question.

Q: How many countries speak both English and Dutch?		

Before:
`SELECT COUNT(DISTINCT`
`CountryCode)`
`FROM countrylanguage`
`WHERE Language = 'English'`
`AND CountryCode IN (`
`SELECT CountryCode`
`FROM countrylanguage`
`WHERE Language = 'Dutch')`
	After:
	`SELECT COUNT(*)`
	`FROM (`
	`SELECT CountryCode`
	`FROM countrylanguage`
	`WHERE Language = 'English'`
	`INTERSECT`
	`SELECT CountryCode`
	`FROM countrylanguage`
	`WHERE Language = 'Dutch')`
		The SQL query provided uses the 'IN' clause, which selects countries that speak either English or Dutch. However, the question asks for countries speaking both languages. Using IN results in a union of sets (either English or Dutch), not an intersection (both English and Dutch). This could be done with two separate queries joined by an 'INTERSECT' clause.

Table 8

A.9 Prompt Examples

```
Describe how you understand the question based on the evidence, and
 help me write an SQL to answer the question.
### EVIDENCE: {evidence}
### USER_QUESTION: {question}

### RELATED SQL:
{related_sql}

### DATABASE STRUCTURE:
{schema}
```

Prompt 1: CoT-SQL-Writer

````
Write an to answer the question.

Program of Thoughts (PoT) is a variant of Chain of Thought (CoT),
pre-generating Python code to assist in the creation of SQL. Please
 apply PoT (and PoT only) before generating an SQL.
In your python code, `Table %s` is stored in `db_dict['%s']`, `
db_dict` is of type dict[pandas.DataFrame].

### RELATED SQL:
{related_sqls}

### DATABASE STRUCTURE:
{schema}

### EXAMPLES:
QUESTION: What is %s in the earliest year and what year was it?
SQL:
earliest_year = db_dict[%s]['Year'].min()
year_filtered_data = step1_result[step1_result['Year'] ==
earliest_year]
result = year_filtered_data[[%s, 'Year']]
```sql
SELECT T1.%s, T2.Year FROM %s AS T1 JOIN %s AS T2 ON T1.Id = T2.Id
WHERE T2.Year = (SELECT min(YEAR) FROM %s);
```

QUESTION: Show names for all %s except for %s having a %s in year
2023.
SQL:
%s_2023 = db_dict['%s'][db_dict['%s']['year'] == '2023']
result = db_dict[%s][~db_dict[%s][%s].isin(%ss_2023[%s])]
```sql
SELECT name FROM %s EXCEPT SELECT T2.name FROM %s AS T1 WHERE T1.
year = 2023
```

QUESTION: Find the %s that %s is A and B?
SQL:
````

````
condition_a_data = db_dict[%s][db_dict['Cartoon'][%s] == 'A']
condition_b_data = db_dict[%s][db_dict['Cartoon'][%s] == 'B']
result = pd.merge(condition_a_data, condition_b_data, how='inner')
```sql
SELECT T1.%s FROM %s AS T1 WHERE %s = 'A'
INTERSECT
SELECT T1.%s FROM %s AS T1 WHERE %s = 'B'
```

### EVIDENCE: {evidence}
### USER_QUESTION: {question}
### SQL:
````

Prompt 2: PoT-SQL-Writer

````
You are the manager of a Database project. You are going to invite
{n} experts to review an SQL query.
Who would you invite?

considering:
(1) the domain of this database;
(2) the structure of this SQL.
Please write your invitation as a JSON format dictionary, Enclose
the JSON within ```json...```.

### DATABASE STRUCTURE:
{schema}

### QUESTION: {question}
### SQL:
{pred_sql}

### EXAMPLES:
```json
{
  "Reviewer PVsg": "Data Analyst in automotive industry",
  "Reviewer 2KtR": "Senior Database Engineer specialized in writing
   various clauses",
  "Reviewer LmN3": "Senior Database Engineer specialized in writing
   filtering conditions"
}
```
### INVITATION:
````

Prompt 3: Invitation