SQL-CRAFT: Advancing Text-to-SQL Capabilities through Interactive Refinement and Enhanced Reasoning

Anonymous ACL submission

Abstract

Modern LLMs have become increasingly powerful, but they are still facing challenges in specialized tasks such as Text-to-SQL. We propose **SQL-CRAFT**, a framework to enhance LLMs' **SQL** generation **C**apabilities through inte**RA**ctive re**F**inemen**T** and enhanced reasoning. We leverage an Interactive Correction Loop (IC-Loop) for LLMs to interact with databases automatically, as well as Python-enhanced reasoning. We conduct experiments on two Text-to-SQL datasets, Spider and Bird, with performance improvements of up to 5.7% compared to the naive prompting method. Moreover, our method surpasses the current state-of-the-art on the Spider Leaderboard, demonstrating the effectiveness of our framework.

1 Introduction

Text-to-SQL – the task of converting natural language to SQL queries – enables non-technical users to access databases in natural language. Recently, Large Language Models (LLMs) have made significant progress on various tasks (Touvron et al., 2023; OpenAI, 2023), but little work explores the task of using LLMs on Text-to-SQL (Chen et al., 2023b).

Most LLMs are pre-trained on publically available corpora (OpenAI, 2023; Anil et al., 2023; Touvron et al., 2023) but there is a scarcity of SQL queries in the pre-training corpora of LLMs. For instance, in the public report from Github[1], the proportion of SQL languages is close to zero, while Python is at the top with a 17.7% share. On the Stack Overflow website, there are 2.2 million questions tagged with 'Python',[2] compared to 0.67 million tagged with 'SQL',[3], fewer than one-third of the 'Python' questions. Such disproportional distribution of coding languages may grant LLMs a

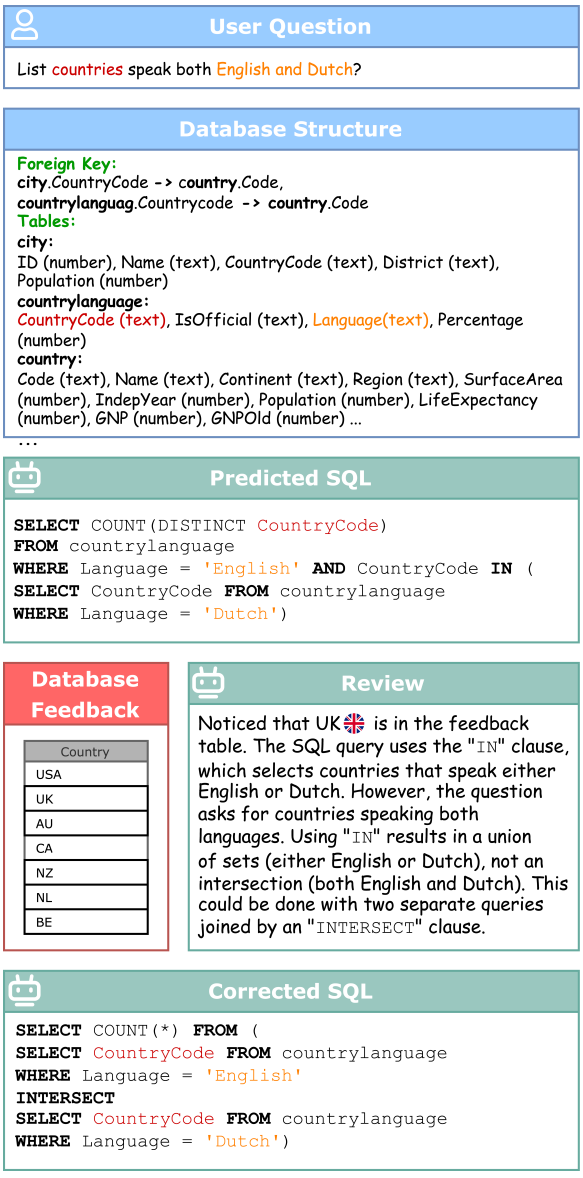

Figure 1: Database Feedback Based Correction. LLMs may identify errors in the SQL when the database feedback conflicts with common sense.

[1] madnight.github.io/githut/#/
[2] stackoverflow.com/questions/tagged/python
[3] stackoverflow.com/questions/tagged/sql

stronger reasoning ability in Python than SQL.

In this work, we aim to enhance LLMs' SQL generation ability. First, we propose Interactive Corrections (IC) to correct LLMs' responses based on the database feedback iteratively (Figure 1). This strategy is inspired by the tricks used by human experts in writing SQL, which involve using database feedback to correct potential mistakes. Second, we explore two ways to incorporate Python in LLMs' reasoning process for SQL generation, including generating Python code first and then translating it to SQL, as well as generating Python code together with the corresponding SQL query simultaneously (Chen et al., 2023a).

Finally, we conduct experiments on two recognized text-to-SQL benchmarks, Spider (Yu et al., 2018) and Bird (Li et al., 2023b). Our strategies enhance the ability of LLMs to generate SQL, surpassing the state-of-the-art methods on the Spider dataset.

2 SQL-CRAFT Framework

Our proposed SQL-CRAFT framework comprises two strategies, Python Enhanced Reasoning, and Interactive Correction.

Python Enhanced Reasoning. We try two methods to integrate Python during the reasoning process, PoT (Chen et al., 2023a) and Code Blocks (CBs).

- Program of Thoughts (PoT). We ask the model to generate both Python Code Block and the SQL queries simultaneously, therefore enforcing the model to incorporate Python code blocks in its "thought process". Formally,

$$\texttt{LM}(Q, D, P) = (C, S),$$

  where we prompt the LM with the user question Q, database description D and prompt P. The LM then sequentially generates Python Code Blocks C followed by SQL S.

- Executable Code Blocks (CBs). We divide the SQL generation process into two steps: first,

$$\texttt{LM}(Q, D, P_1) = C,$$

  where given the user question Q, database description D and prompt P_1, LM generates a series of Python Code Blocks C aligned with its chain-of-thought reasoning process, see Appendix A.4. Then,

$$\texttt{LM}(Q, D, P_2, C) = S,$$

  where the LM generates the target SQL query S given the user question Q, database description D, a different prompt P_2 and the Python Code Block C we get in the first step.

Algorithm 1: IC-Loop Control

Given C_0;
$C = \texttt{LLM}(P, C_0, E)$; $E \leftarrow \texttt{DB}(C_0)$;
while $C \neq C_0$ **do**
  $C_0 \leftarrow C$; $C = \texttt{LLM}(P, C_0, E)$; $E \leftarrow \texttt{DB}(C_0)$
end

IC-Loop. We introduce Interactive Correction (IC) shown in Figure 1, which incorporates database feedback into the prompt to guide LLMs to refine their initial response. As different questions may require varying numbers of corrections, we embed IC within a loop (IC-Loop). To avoid over-correction or insufficient correction, we design an automated control process to decide when to terminate the loop. Given the prompt P (including Instruction, Schema, Question) and code C_0, we execute it on the database DB to get the result E. The LLM continues reviewing the execution result and rewrites C_0 until LLM produces the same code as the previous run ($C = C_0$). Algorithm 1 depicts this process.

In simpler terms, we first set an upper limit on the number of iterations for a loop. After generating the initial SQL, we enter the IC-Loop. At this stage, the execution results are converted into CSV format and inserted into the prompt. The model is then instructed to write a new SQL based on the question, the original SQL, and its execution results. If the model believes the current SQL is correct, it will repeat the same SQL. The loop terminates when the program detects that the model has generated the same SQL again, indicating that the model's knowledge and the question-answer pair have reached a consensus. Correction cases can be found in Appendix A.3.3

3 Experiments and Results

We conduct experiments on two cross-domain text-to-SQL benchmarks detailed in Table 1. We employ test-suite execution evaluation[4] (Zhong et al., 2020), the standard evaluation protocol for Spider, and the official SQL execution accuracy (EA) evaluation for Bird[5].

[4] github.com/taoyds/test-suite-sql-eval
[5] bird-bench.github.io/

	Spider (Yu et al., 2018)	Bird (Li et al., 2023b)
Dev	1,034	1,534
#Domain	138	37
#DB	200	95
DB Size	-	33.4 GB

Table 1: Statistics of two text-to-SQL benchmarks we use in our experiments. "#Domain" and "#DB" refer to the number of domains and databases, respectively.

	GPT-3.5-Turbo		GPT-4	
	Spider	Bird	Spider	Bird
SQL-Only	78.2	29.99	79.7	53.30
PoT	78.5	30.70	80.0	54.61
CBs	78.6	–	80.6	–
IC-Lp	78.3	30.38	82.3	54.89
IC-Lp+CBs	79.6	–	83.3	–
DAIL$_{\text{Gao et al. (2023)}}$	–	–	83.1	54.76
IC-Lp+PoT$_{\text{ours}}$	79.3	33.25	85.4	55.20

Table 2: Performance comparison between our methods and current SOTA DAIL(Gao et al., 2023) and ablation studies of different components for our method. IC-Lp refers to the IC-Loop in Section 2. For a fair comparison, we adopt a 5-shot setting for experiments in this table. Hypothesis testing (see Appendix A.2) supports that the enhancement in the SQL generation capabilities of LLMs by the DB-Copilot framework is significant.

For a fair comparison with the latest state-of-the-art work DAIL (Gao et al., 2023) on Spider-Dev[6], we also apply the 5-shot setting to GPT-4 and GPT-3.5-Turbo. Specifically, we construct the embeddings for the user questions and training set questions using the Ada 2 text embedding model from OpenAI[7], and then retrieve 5 question-SQL pairs from the training set with the highest cosine similarity scores as our few-shot examples.

Overall Results. Table 2 reports the overall results. Both IC-Loop and PoT/CBs improve the execution accuracy across the models and datasets. Compared to naively using GPT-4 for SQL generation, our framework achieves a maximum of 5.7% and 1.9% improvement on the dev set for Spider and Bird respectively. Our best results yield an 85.4% and 55.2% performance on the dev set for Spider and Bird respectively, surpassing the state-of-the-art, DAIL (83.1% and 54.76% on Spider and Bird, respectively).

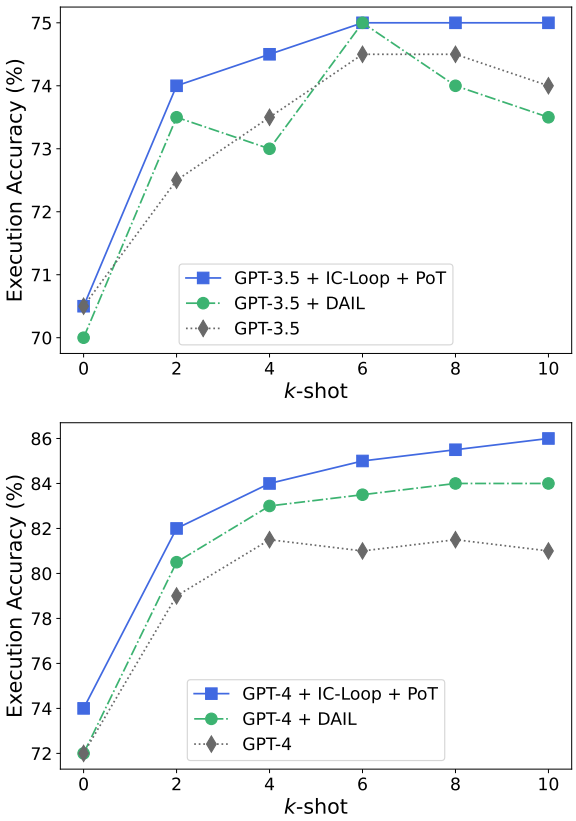

Figure 2: k-shots Sensitivity Analysis.

Effects of k in k-shot. We test various k on 200 examples from Spider-Dev (easy 26%, medium 36.5%, hard 16%, extra-hard 21.5%). As shown in Figure 2, when integrated with our methods of IC-Loop and PoT, both GPT-3.5 and GPT-4 outperform direct prompting and the DAIL approach consistently across almost all k values. Moreover, for the less capable model GPT-3.5, both direct prompting and DAIL approaches yield a worse performance at $k = 8, 10$, potentially due to the model's diminished attention over longer input. In contrast, the performance decay for our method is minimal to none, suggesting our method can effectively enhance LLM's reasoning capability even for less capable LLMs.

Error Analysis In total, there are 151 examples of Spider dev on which our framework fails to generate the gold SQL. Table 3 shows the error case distribution for our framework on the dev set of Spider (more cases in Appendix A.3).

Some errors do not stem from our framework's inability to accurately predict SQL, but rather from

[6]yale-lily.github.io/spider

[7]platform.openai.com/docs/guides/embeddings/use-cases

Error Type	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:** `SELECT Name FROM country WHERE Continent = "Asia" AND population > (SELECT min(population) FROM country WHERE Continent = "Africa")` **Pred:** `SELECT Name FROM country WHERE Continent = "Asia" AND population > (SELECT max(population) FROM country WHERE Continent = "Africa")`	Only countries with a population larger than the largest in Africa can be called "have larger population than any country in Africa". Therefore, employing the term `... > (SELECT max(population) ...` is correct, whereas the Gold is erroneous.
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 wrongly assumes that all owners have had dogs by attempting to subtract the number of current dog owners from the total owners.
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;`	Previous studies have shown that multiple correct answers can coexist across various NLP tasks (Plank, 2022; Deng et al., 2023). In this case, both "FullName" and "Maker" columns can represent the "name of makers" in the question, therefore both gold and model-generated SQL are correct.
Imprecise (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 predicted SQL selects the "Name" column, which 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%";`	After checking the database values in IC-Loop, the model notices that there is an space for the value "Alton" in the database. Since the Gold SQL query employs an exact match (=`"Alton"`), it cannot retrieve the corresponding data. In contrast, our framework takes such nuanced differences into consideration and uses the `LIKE` clause for a fuzzy match on "Alton".
Others (4.6%): other errors that we cannot categorize into the above types.		

Table 3: Error Cases

the incorrect gold answer, question ambiguity, or the database setup. Among the 151 examples, 30.5% are due to annotation errors (4.5% of all the examples in Spider dev). Appendix A.3.1 shows an example where the gold SQL fails to answer the question. We catalog the instances with incorrect gold SQL, correct the errors, and share the details.[8] Our findings indicate that the existing evaluation protocols for text-to-SQL generation may not authentically capture the capabilities of these sophisticated models. Therefore, we advocate for a reassessment and enhancement of text-to-SQL evaluation methods.

Apart from the aforementioned errors, other errors can be attributed to the *limitations of the model in accurately interpreting or processing the given information*. Such an error reveals that even sophisticated LLMs, such as GPT-4, still struggle with precise interpretation and alignment to the specific requirements of the question. We provide error analysis on Bird in Appendix A.3.2.

[8]visible_after_review.com

4 Conclusion

We propose SQL-CRAFT, a framework involving interactive refinement and Python-enhanced reasoning that improves SQL generation accuracy for LLMs. SQL-CRAFT demonstrates an improvement of 5.7% on Spider and 1.9% on Bird compared to the naive prompting method. Moreover, SQL-CRAFT surpasses the state-of-the-art strategies on Spider. We conduct a comprehensive error analysis and observe that there are issues with the current text-to-SQL evaluation. Therefore, we call for attention from our community to develop a better text-to-SQL evaluation protocol that can capture nuances in SQL generation and authentically reflect the model performance.

5 Ethical Statements and Limitations

Strategies we propose are aim to improve the SQL generation capabilities of LLMs, with some of its concepts, possibly applicable to general language model tasks. There are some potential societal consequences of our work, none that we feel must be specifically highlighted here.

In industrial application scenarios, SQL genera-

tion is often more complex and demands higher accuracy. The research of Text-to-SQL still has a long way to go before it becomes practically valuable. During the process of conducting case studies, we encounter some unpredictable output results, which might be related to the hallucination of LLMs.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations. *arXiv preprint arXiv:2106.01093*.

Mingda Chen, Jingfei Du, Ramakanth Pasunuru, Todor Mihaylov, Srini Iyer, Veselin Stoyanov, and Zornitsa Kozareva. 2022. Improving in-context few-shot learning via self-supervised training. *arXiv preprint arXiv:2205.01703*.

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

Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023b. Seed: Domain-specific data curation with large language models. *arXiv e-prints*, pages arXiv–2310.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

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

Yixin Liu and Pengfei Liu. 2021. Simcls: A simple framework for contrastive learning of abstractive summarization. *arXiv preprint arXiv:2106.01890*.

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.

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*.

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. *arXiv preprint arXiv:2205.06983*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*.

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*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

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.

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 Additional Relevant Work

In addition to the related work highlighted in the introduction, there is other relevant work discussed below. In the past year, LLMs have made breakthroughs in many fields, recent advancements in NL2SQL conversion have been significantly influenced by LLMs such as GPT variants. Pourreza and Rafiei (2023) (2023) propose DIN-SQL, a method that employs GPT-4 in a few-shot learning paradigm, demonstrating notable performance in NL2SQL tasks. Dong et al. (2023) introduce C3, a ChatGPT-based zero-shot NL2SQL method, which emphasizes clear prompting, calibration with hints, and consistent output to improve execution accuracy. Gao et al. (2023) further explore the efficiency and effectiveness of LLMs in NL2SQL through their work DAIL-SQL, offering a comprehensive evaluation of various prompt engineering strategies. They demonstrate that DAIL-SQL, when equipped with GPT-4, achieves superior execution accuracy and token efficiency, making it a practical solution for real-world applications. Some recent works applied prompting method and the principle of consistency (Wang et al., 2022) to enhance the reasoning ability of LLMs through In-Context Learning, such as Chain-of-Thoughts(CoT) (Wei et al., 2022) and Tree-of-Thoughts(ToT) (Yao et al., 2023). Chen et al. (2023a) proposed the use of Python code to assist LLMs in reasoning called Program of Thoughts (PoT), achieving results that surpass CoT on math problems.

Recent studies indicate that Language Models can learn from a few examples given in the context, known as in-context learning (Brown et al., 2020; Chen et al., 2022; Liu and Liu, 2021). These works suggest that context learning can effectively enable LMs to perform a range of complex tasks. Therefore, considering the complexity of the NL2SQL task, context learning can be employed to guide LMs to better generate SQL queries.

In addition to leveraging the in-context learning capabilities of LLMs, numerous studies have applied Pre-trained Language Models (PLMs) to the NL2SQL task, as the extensive pre-training on large textual corpora enables PLMs to better model the semantic relationship between user questions and database schemas. Overall, the PLMs used in these works mainly fall into two categories: encoder-only PLMs (such as BERT (Devlin et al., 2018), ELECTRA (Clark et al., 2020)) and encoder-decoder PLMs (like BART (Lewis et al., 2019), T5 (Raffel et al., 2020)). For encoder-only PLMs, systems like RATSQL (Wang et al., 2019) and LGESQL (Cao et al., 2021) use BERT to encode user questions and database schemas, further employing graph neural networks to model foreign keys and schema linking. The encoded representations are then fed into a grammar-based syntactic neural decoder to generate SQL queries. For encoder-decoder PLMs, approaches like PICARD (Scholak et al., 2021), RASAT (Qi et al., 2022), and RESDSQL (Li et al., 2023a) frame the NL2SQL task as an end-to-end translation problem, using the T5 model to directly translate user questions into SQL queries. Additionally, task-specific strategies

like relation-aware self-attention (Qi et al., 2022), schema selection (Li et al., 2023a), and constrained decoding (Scholak et al., 2021) further enhance the accuracy of Encoder-Decoder PLMs in generating SQL queries.

A.2 Significance Test

We divided the SQL generated by several strategies in Table 2 into 10 equal parts and calculated the execution accuracy for each. Figure 3 shows the box plot drawn using this data.

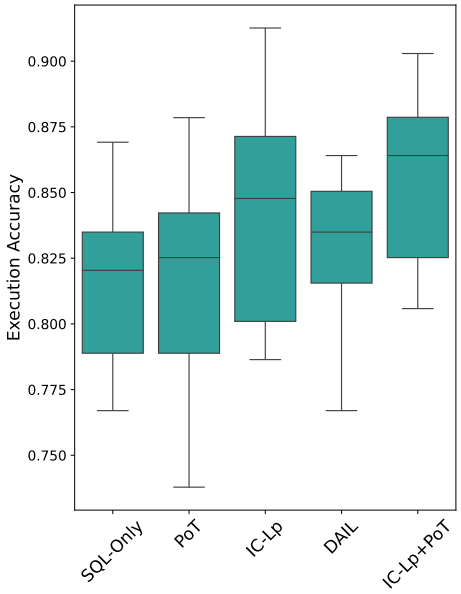

Figure 3: Execution Accuracy Box Plot

To test whether our strategy can indeed improve Execution Accuracy (EA), we conduct a significance test between the "SQL-Only" and "IC-Lp+PoT" strategies. The null hypothesis of the test is that the median EAs obtained by the two strategies are 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 0.0243, which is below the commonly accepted significance level of 0.05. Therefore, we have reason to reject the null hypothesis, indicating that the "IC-Lp+PoT" strategy leads to a significant performance improvement.

A.3 More Cases

A.3.1 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 4

A.3.2 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 5

A.3.3 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 6

A.4 Prompt Examples

A.4.1 PoT Prompt Example

Global The ‘write down CODE to answer the ‘### QUESTION‘, and translate CODE to SQL. ‘### DB_STRUCTURE‘ includes examples of each table, and the filtering criteria should be based on these data examples. In the CODE, ‘Table %s‘ is stored in ‘db_dict[’%s’]‘, ‘db_dict‘ is of type dict[pandas.DataFrame]

RELATED_SQLs: <...>

DB_STRUCTURE: Database name: world_1 Tables: city, sqlite_sequence, country, countrylanguage Primary Keys: city[’ID’], country[’Code’], countrylanguage[’CountryCode’], Foreign Key: pd.merge(city, country, left_on=’CountryCode’, right_on=’Code’) pd.merge(countrylanguage, country, left_on=’CountryCode’, right_on=’Code’) Columns:
Table city: ID (number), Name (text), CountryCode (text), District (text), Population (number)
Table sqlite_sequence: name (text), seq (text)
Table country: Code (text), Name (text), Continent (text), Region (text), SurfaceArea (number), IndepYear (number), Population (number), LifeExpectancy (number), GNP (number), GNPOld (number), LocalName (text), GovernmentForm (text), HeadOfState (text), Capital (number), Code2 (text)
Table countrylanguage: CountryCode (text), Language (text), IsOfficial (text), Percentage (number)

EXAMPLES: QUESTION: What is %s in the earliest year and what year was it? THOUGHT: The answer should consist of two columns, representing %s and the earliest year from the joined data, respectively. CODE: earliest_year = db_dict[%s][’Year’].min() year_filtered_data = step1_result[step1_result[’Year’] == earliest_year] result = year_filtered_data[[%s, ’Year’]] SQL: “‘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. THOUGHT: The answer to the question should only include the ’Name’ column of stadiums that do not have a concert in the year 2023, and it’s a text column. CODE: %s_2023 = db_dict[’%s’][db_dict[’%s’][’year’] == ’2023’] result = db_dict[%s][db_dict[%s][%s].isin(%ss_2023[%s])] SQL: “‘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? THOUGHT: The task is to find instances where a single column (%s) meets two conditions simultaneously: being ’A’ and ’B’. This scenario is akin to finding an intersection in two datasets. To achieve this, two separate filters are applied to the same dataset, each for one of the conditions. Then, an inner merge is used to find common entries that satisfy both conditions. CODE: 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: “‘sql SELECT T1.%s FROM %s AS T1 WHERE %s = ’A’ INTERSECT SELECT T1.%s FROM %s AS T1 WHERE %s = ’B’ “‘

QUESTION: How many countries speak both English and Dutch? THOUGHT: