DAC: Decomposed Automation Correction for Text-to-SQL

Anonymous ACL submission

Abstract

Text-to-SQL is an important task that helps access databases by generating SQL queries. Currently, correcting the generated SQL based on large language models (LLMs) automatically is an effective method to enhance the quality of the generated SQL. However, previous research shows that it is hard for LLMs to detect mistakes in SQL directly, leading to poor performance. Therefore, in this paper, we propose to employ the decomposed correction to enhance text-to-SQL performance. We first demonstrate that detecting and fixing mistakes based on the decomposed sub-tasks is easier than using SQL directly. Then, we introduce Decomposed Automation Correction (DAC), which first generates the entities and skeleton corresponding to the question, and then compares the differences between the initial SQL and the generated entities and skeleton as feedback for correction. Experimental results show that, compared with the previous automation correction method, DAC improves performance by 1.4% of Spider, Bird, and KaggleDBQA on average, demonstrating the effectiveness of our method[1].

1 Introduction

Text-to-SQL is an important task, significantly reducing the overhead of obtaining information from the database by generating corresponding SQL based on user questions (Deng et al., 2022). Specifically, about the text-to-SQL task, users provide the question and the related database, and then the model generates the corresponding SQL. Currently, the methods based on Large Language Models (LLMs) have become the mainstream for text-to-SQL because LLMs can achieve higher performance than fine-tuned models, without fine-tuning (Hong et al., 2024b; Li et al., 2024a; Shi et al., 2024; Kanburoğlu and Tek, 2024). Therefore, in this paper, we mainly focus on how to enhance the text-to-SQL performance based on LLMs.

[1]Our code and data will be released upon the acceptance

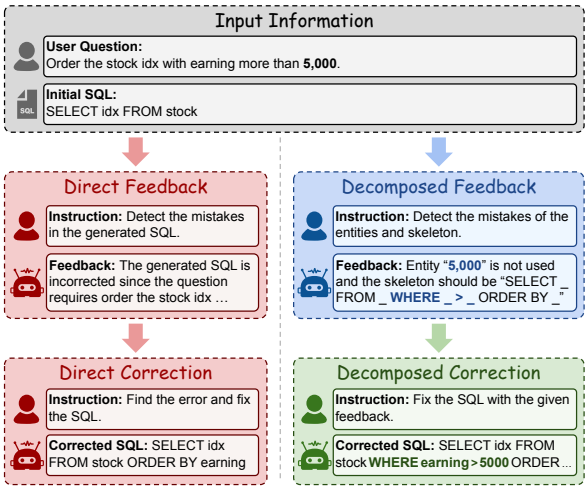

Figure 1: The comparison between direct correction (**left**) and decomposed correction (**right**). Direct correction shows poor performance since LLMs do not know how to detect mistakes. Decomposed correction brings better performance based on the decomposed tasks of entity linking and skeleton parsing.

Recent research shows that automated correction is an effective method to improve text-to-SQL performance (Chen et al., 2024; Askari et al., 2024; Xie et al., 2024; Wang et al., 2024a), where LLMs first generate an initial SQL and then detect mistakes of the SQL as correction feedback (Pan et al., 2024). For instance, SQL-CRAFT (Xia et al., 2024) assesses whether the SQL execution results on databases satisfy the user question and corrects the SQL accordingly. EPI-SQL (Liu and Tan, 2024) builds an example library of error cases and then retrieves similar cases from the library to guide each correction. However, previous studies indicate that it is hard for LLMs to directly detect and correct all mistakes in the generated SQL (Zhang et al., 2024a; Kamoi et al., 2024; Zhang et al., 2024c).

Previous works (Chen et al., 2023b; Dhuliawala et al., 2024; Vacareanu et al., 2024) show that correction with the decomposed reasoning process has better performance than directly correcting because detecting the mistakes in the reasoning pro-

cess is easier than in the final answer. Therefore, in this paper, we discuss that: *(i)* Experimentally, we present that ***correction with decomposed sub-tasks is more effective than direct correction***; *(ii)* Methodologically, we propose ***correcting the text-to-SQL results based on the decomposed tasks of entity linking and skeleton parsing*** to enhance the performance of the automation correction.

First, we discuss that correction with the decomposed sub-tasks can bring better correction performance than the direct correction, as shown in Figure 1. We present that decomposing into sub-tasks can lower the difficulty of detecting and correcting errors. Based on the above analysis, we propose **D**ecomposed **A**utomation **C**orrection (DAC), which corrects the generated SQL with the mistakes of entity linking and skeleton parsing as feedback, which are the two most crucial capabilities for addressing the text-to-SQL task (Li et al., 2023a; Deng et al., 2022; Hong et al., 2024b). Schema linking denotes detecting the table and column names related to the question, where we directly generate relevant entities based on the user question and the database, following Li et al. (2023c). Skeleton parsing denotes generating the SQL skeleton corresponding to the user question, where we generate the skeleton by parsing the user question without databases (Guo et al., 2024b,a). After generating the entities and skeleton, we take their inconsistencies with the initial SQL as feedback for the correction, as shown in Figure 1.

To validate DAC, we conduct experiments on three mainstream text-to-SQL datasets: Spider (Yu et al., 2018), Bird (Li et al., 2023b), and KaggleDBQA (Lee et al., 2021). The experiments show that DAC achieves an average improvement of 1.4% compared to the previous correction method, showing the effectiveness of DAC. Further analysis shows that DAC indeed improves the performance by improving the entity and the skeleton accuracy.

Our contributions are as follows:

- We experimentally discuss that decomposed correction has better performance than direct correction, shedding light on future research;
- We present DAC, which improves text-to-SQL automated correction performance by decomposing text-to-SQL into sub-tasks of entity linking and skeleton parsing;
- Experiments show that DAC improves performance by 1.4% on average compared to the previous automated correction methods, proving the effectiveness of our method.

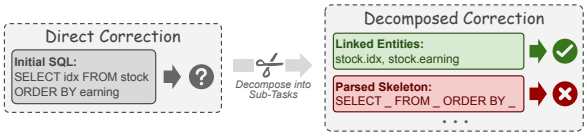

Figure 2: The illustration of our discussion, taking the question "*Order the stock idx with earnings more than 5,000*" as an example. About the direct correction (**left**), it is challenging for LLMs to pinpoint specific mistakes. After decomposing the SQL into sub-tasks (**right**), it is easier for LLMs to identify that the linked entities are correct, while the parsed skeleton is incorrect.

2 Preliminaries

In this section, we discuss that **decomposing text-to-SQL into sub-tasks can effectively enhance SQL correction performance**. First, we demonstrate that correction by decomposing into sub-tasks is more effective than direct correction since sub-tasks are easier to satisfy than the original task, making it less difficult to detect and correct errors. Then, we adapt the above conclusion to the text-to-SQL task by decomposing the task into two sub-tasks: entity linking and skeleton parsing, thereby enhancing the SQL correction performance.

2.1 Performance of Sub-Task Correction

We consider the correction performance of task decomposition in two parts: correction results satisfy the sub-tasks, and correction results that satisfy the sub-tasks also satisfy the original task. We propose that decomposing the task into sub-tasks can make the correction results better satisfy the sub-tasks, while the correction results satisfying the sub-tasks could not satisfy the original task. The illustration of the above proposition is shown in Figure 2.

Intuitively, the original task can be viewed as a combination of multiple sub-tasks (Khot et al., 2023). Since the sub-tasks used are included in the original task, according to Wang et al. (2024b), generating correct answers to sub-tasks is easier than the original task, thereby it is easier to detect the mistakes based on these sub-tasks than the original task. However, because the used sub-tasks do not entirely encompass the original task, even if the correction results satisfy all the used sub-tasks, it cannot guarantee that the results fully meet the original task. Therefore, when decomposing sub-tasks, it is necessary to ensure that the sub-tasks can maximally cover the original task so that the correction results satisfying the sub-tasks are more likely to satisfy the original task.

2.2 Adaption to Text-to-SQL

We apply the discussion above to the text-to-SQL task, where we use entity linking and skeleton parsing as sub-tasks for automated correction. Entity linking refers to identifying relevant table and column names from the given database schema based on the user question (Liu et al., 2021). Skeleton parsing denotes creating the corresponding SQL skeleton of the user question, which removes the table names, column names, and values in SQL (Guo et al., 2024b). We use these two sub-tasks because previous work regards them as the most crucial capabilities for text-to-SQL (Li et al., 2023a; Deng et al., 2022; Hong et al., 2024b).

While based on the discussion above, correction of the sub-tasks could not ensure the result correctness of the text-to-SQL task. Therefore, we calculate the proportion of correct SQL among all SQL with correct entities and skeletons, as shown in Table 4. The result in the table shows that the generated SQL in line with our proposed sub-tasks is also in line with the text-to-SQL task, demonstrating the effectiveness of our sub-task decomposition method for the text-to-SQL automated correction.

3 Methodology

Following the discussion above, in this section, we present DAC, which generates automated correction feedback by decomposing the text-to-SQL task into the sub-tasks of entity linking and skeleton parsing. The illustration of DAC is shown in Figure 3, and the prompts we used for DAC are present in Appendix A.

3.1 SQL Generation

Before the correction, we first generate an initial SQL to be corrected using LLMs. Following previous work (Xia et al., 2024), we use few-shot learning to generate SQL. We use BM25 to select demonstrations similar to the user question from the given demonstration pool, together with the database and the user question as the input. With the input, LLMs directly generate a single SQL as the initial result to be corrected.

3.2 Entity Linking

After obtaining the initial SQL, we aim to identify mistakes in the linked entities. To achieve this, it is essential to know the entities related to the user question, specifically the table and column names in the question-related database. Following previous work (Andrew et al., 2024; Goel et al., 2023), we use LLMs to select the question-related tables and columns. We employ the tokenized user question and the database as inputs and output the entity linking information in the format of `List[Dict[str, str]]` in Python, where the length of the list matches the number of tokens in the tokenized question. We use this format because the performance of entity linking generated in programmatic format by LLMs is superior to only generating entities (Li et al., 2023c).

Following Liu et al. (2021), each `Dict` includes three keys: *(i) token*: the corresponding token in the original question; *(ii) schema*: the corresponding table name or column name in the database; *(iii) type*: including "*tab*", "*col*", and "*val*", representing the table name, column name, or a condition value that the token corresponds to respectively.

3.3 Skeleton Parsing

After identifying the linked entities, we seek to parse the skeleton corresponding to the user question to identify mistakes in the initial SQL generation on the skeleton. Previous research indicates that LLMs can parse the SQL skeleton based solely on the user question since the skeleton corresponds to the syntactic logic of the question without requiring database information (Gu et al., 2023; Pourreza et al., 2024; Kothyari et al., 2023). Therefore, we only use the user question as the input without the database, asking LLMs to parse an SQL skeleton corresponding to the question solely. Subsequently, we remove entities from the parsed SQL, since we do not provide the database in the input, where these entities do not correspond to the column names or table names in the original database, thus obtaining the SQL skeleton.

3.4 Comparison

After obtaining the linked entities and parsed skeleton, we directly compare and identify the inconsistency parts between them and the initial SQL as the correction feedback, without employing LLMs. We first extract the entities and skeletons in the initial SQL according to the SQL syntax. For *linked entities*, we determine which linked entities are mentioned in the question but not used in the initial SQL. We do not consider entities that are present in the SQL but not mentioned in the question as mistakes because the user question could not explicitly mention some entities used in the SQL. For example, some tables are used to bridge the other two

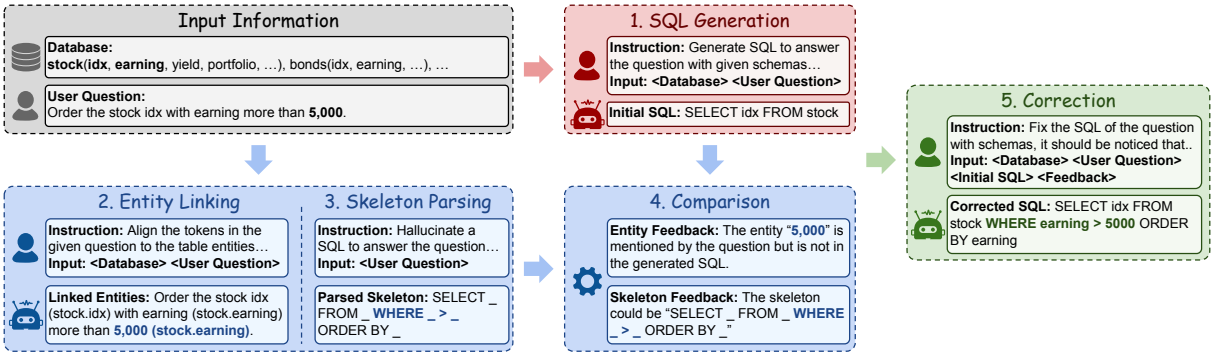

Figure 3: The pipeline of DAC, which consists of five steps: ***(i)*** **SQL Generation**: Generate the initial SQL; ***(ii)*** **Entity Linking**: Detect the question-related entities; ***(iii)*** **Skeleton Parsing**: Generate the SQL skeleton of the question; ***(iv)*** **Comparison**: Determine the inconsistencies between the initial SQL and the linked entities and parsed skeletons as feedback; ***(v)*** **Correction**: Correct the SQL with the comparison feedback.

tables by the foreign keys without being mentioned by the user question. About the *parsed skeleton*, we correct the initial SQLs whose skeletons are inconsistent with the parsed skeletons. We input the completed parsed skeleton as the feedback, where do not indicate mistaken keywords because different keywords could appear multiple times in the SQL, while too specific information could confuse the model about which keywords to be modified.

3.5 Correction

Upon constructing correction feedback, we use them to correct the initial SQL. If an error message occurs during SQL execution, we use the error message as feedback, following Chen et al. (2024). We take the database, user question, initial SQL, and feedback as input, and ask LLMs to output the corrected SQL. During the correction, we first correct entity mistakes, followed by skeleton mistakes. We do not correct both simultaneously because overly complex feedback could confuse the LLM. We first correct the linked entities since the entity information can help LLMs correct the skeleton. For example, if the user question asks for the "*student with the top-ranked score*", and the database directly provides a "*rank*" column, the SQL can directly select data with "*rank=1*" without changing the skeleton with "*ORDER BY score*".

4 Experiments

4.1 Settings

4.1.1 Datasets

We use Spider (Yu et al., 2018), Bird (Li et al., 2023b), and KaggleDBQA (Lee et al., 2021) to evaluate DAC, where we adapt DAC on the dev sets of these datasets. Spider is one of the most mainstream text-to-SQL datasets, which covers multi-domain questions. Compared with Spider, Bird is more difficult, and databases are more complex and more near to the practical application. The difficulty of KaggleDBQA is close to that of Bird, but the scale of the database is larger[2].

4.1.2 Metric

In this paper, following the previous works (Askari et al., 2024; Cen et al., 2024), we use execution accuracy (EX) to evaluate our method. EX refers to the proportion of predicted SQL execution results that are the same as correct SQL execution results.

4.1.3 Models

We verify our methods on the instruction fine-tuning version of Llama3 (Meta, 2024), Deepseek-Coder (Deep.C.) (Guo et al., 2024c), `gpt-3.5-turbo-1106` (GPT-3.5) (Ye et al., 2023) and `gpt-4o` (GPT-4o) (OpenAI et al., 2024). Llama3 and Deepseek-Coder are the most mainstream general and code LLMs in the current open-source LLMs, respectively. While GPT-3.5 and GPT-4o are the most popular closed-source LLMs. Therefore, our experimental LLMs can cover different application scenarios well.

4.2 Baselines

We compare DAC with the following two baselines: *(i) Few-Shot*: generate SQL with few-shot inference (Brown et al., 2020) without correction; *(ii) Self-Debug*: LLMs correct the generated SQL according to the errors during execution or found by themselves (Chen et al., 2024).

[2]We call KaggleDBQA as **Kaggle** for simplicity.

4.2.1 Implementation Details

We employ few-shot (Brown et al., 2020) inference for each step in DAC. Following the previous work (Chang and Fosler-Lussier, 2023), we use 5-shot for the inference. We use the Spider training set as the candidate demonstration pool and use BM25 to select the most similar demonstrations for each user question. For the entity linking, we employ the labeled linking information of Spider collected by Liu et al. (2021), which is used to guide the generation format (Madaan et al., 2023) and can be adapted to different databases as shown in Table 1. The prompts we used can be seen in Appendix A.

4.3 Main Experiment

The main experiment results of our method are shown in Table 1. From the table, it can be found that DAC brings performance improvement on all datasets and all models, with an average improvement of 1.4% compared with the Self-Debug method, proving the effectiveness and generalization of our method. To better study the improvement brought by DAC, we also compare DAC with other SQL correction methods in Appendix B and adapt case studies in Appendix C. Besides, from Table 1, we can also see that:

Dataset DAC improves performance across different datasets. Compared to Spider, our method achieves more performance gains on Bird and KaggleDBQA in most settings. This is because, with more complex questions and databases, LLMs are more likely to make entity-linking mistakes or generate incorrect skeletons. Consequently, our method enhances model performance more effectively in Bird and KaggleDBQA by correcting skeleton or entity mistakes.

Model Our method also brings significant performance improvements across different models. Compared to code-specific LLMs, the average improvement by our method is more pronounced in general LLMs compared to the Self-Debug method. This is because the general LLMs have better entity linking and skeleton parsing ability than the code-specific LLMs, leading to higher correction performance, as the discussion in §4.5.2 and §4.5.3.

Scale DAC improves performance across different model scales, showing the effectiveness of DAC. The performance improvement is more significant for small-scale models compared to large-scale models. This is because small-scale models are more prone to mistakes in entities and skeletons due to their performance limitations. Our method, by correcting from sub-tasks, lowers the difficulty of detecting and correcting errors, thereby effectively guiding the model to generate correct results.

4.4 Ablation Study

To validate the effectiveness of each step that our method designed, we conduct ablation experiments on entity linking and skeleton parsing. The results are shown in Table 2, from which we can see that: *(i)* Removing either entity linking or skeleton parsing results in a performance drop, confirming the effectiveness of each step of DAC; *(ii)* The performance drop is more significant when skeleton parsing is removed compared to entity linking, indicating that the primary performance bottleneck of the current model lies in skeleton parsing; *(iii)* The performance degradation is more pronounced in models with smaller scales after ablation, suggesting that errors in the results generated by smaller models are more significant, and thus DAC can bring greater performance improvements.

4.5 Analysis

4.5.1 Is SQL with the correct entity and skeleton the correct SQL?

In §2.1, we propose that automated correction from the perspective of sub-tasks could not yield correct SQL results. To demonstrate the effectiveness of DAC, we calculate the proportion of correct SQL among those with correct entities and skeletons, as shown in Table 4. From the table, we can observe that: *(i)* The results under most settings are close to 100%, proving a strong consistency between automated correction from the perspectives of entities and skeletons and the correct SQL, thereby validating the effectiveness of our method; *(ii)* Compared to Spider, the accuracy of results on Bird and KaggleDBQA is lower, indicating that these two datasets are more challenging and require abilities beyond entity linking and skeleton parsing, like how to fill the entities into the skeleton correctly.

4.5.2 Can LLMs generate entity linking information effectively?

To validate the performance of LLMs in entity linking, we conduct experiments on the Spider dev set in Table 3, where we employ the evaluation metric following Liu et al. (2021). From the table, we can see that: *(i)* In most metrics, the performance of entity linking based on LLMs is superior to that based

Dataset	Method	Llama3 8b	Llama3 70b	Deepseek-Coder 6.7b	Deepseek-Coder 33b	GPT-3.5 -	GPT-4o† -
Spider	Few-Shot	70.7	80.7	74.1	77.0	76.9	82.8
	+ Self-Debug	72.1	80.8	76.1	79.4	79.2	84.4
	+ DAC	**75.0**	**81.3**	**77.1**	**80.1**	**80.6**	**85.2**
Bird	Few-Shot	30.4	47.0	42.4	48.3	40.0	49.2
	+ Self-Debug	34.6	47.6	43.6	51.6	43.5	50.8
	+ DAC	**37.3**	**48.4**	**44.7**	**52.7**	**44.9**	**52.3**
KaggleDBQA	Few-Shot	24.6	31.4	21.7	27.0	24.3	37.5
	+ Self-Debug	28.7	33.6	25.0	27.8	29.8	40.6
	+ DAC	**29.4**	**35.1**	**25.4**	**29.2**	**30.9**	**45.3**

Table 1: The main experiment results of DAC on the dev set of each dataset. The baseline Few-Shot method uses the initial SQL as the result. † denotes the results on 128 examples randomly sampled from the dev set of each dataset due to the limited computational resources. The best performance of each setting is marked in **bold**.

Model	Scale	Method	Spider	Bird	Kaggle
Llama3	8b	DAC	75.0	37.3	29.4
		- Entity	74.9 (−0.1)	36.4 (−0.9)	29.2 (−0.2)
		- Skeleton	73.3 (−1.7)	35.3 (−2.0)	29.0 (−0.4)
	70b	DAC	81.3	48.4	35.1
		- Entity	81.2 (−0.1)	47.1 (−1.3)	34.9 (−0.2)
		- Skeleton	80.9 (−0.4)	47.1 (−1.3)	33.8 (−1.3)
Deepseek-Coder	6.7b	DAC	77.1	44.7	25.4
		- Entity	77.0 (−0.1)	44.1 (−0.6)	25.4 (−0.0)
		- Skeleton	76.5 (−0.6)	44.6 (−0.1)	25.2 (−0.2)
	33b	DAC	80.1	52.7	29.2
		- Entity	79.9 (−0.2)	52.1 (−0.6)	28.3 (−0.9)
		- Skeleton	79.4 (−0.7)	51.6 (−1.1)	28.9 (−0.3)

Table 2: The ablation experiments of DAC on: *(i) Entity*: remove the linked entities during the correction; *(ii) Skeleton*: remove the parsed skeleton during the correction.

Method	Table P	Table R	Table F	Column P	Column R	Column F
N-Gram	78.2	69.6	73.6	61.4	69.1	65.1
EtA+BERT	81.1	85.3	83.1	86.1	79.3	**82.5**
Llama3-8b	80.9	86.4	81.6	76.6	78.3	75.5
Llama3-70b	**88.9**	**89.1**	**87.4**	78.9	**83.1**	79.0
Deep.C.-6.7b	69.4	74.6	70.0	72.8	72.6	70.3
Deep.C.-33b	79.1	82.5	79.1	**86.2**	78.6	79.9

Table 3: The entity linking performance on Spider. P, R, and F represent the macro average of accuracy, recognition rate, and F1 respectively. The results of N-Gram and EtA+BERT are reported by Liu et al. (2021). The best performances are marked in **bold**.

on grams or fine-tuning methods (Liu et al., 2021), demonstrating the effectiveness of LLMs in entity linking for the text-to-SQL task; *(ii)* Compared to code-specific LLMs, general LLMs exhibit better entity-linking performance in most metrics, indicating that general-purpose LLMs are more suitable for the entity-linking task. *(iii)* However, the F value of LLMs is lower than the past method, indicating that LLMs could omit or add irrelevant tokens during the linking, lowering performance.

Model	Scale	Spider	Bird	Kaggle
Llama3	8b	99.7	93.8	94.0
	70b	99.7	93.3	96.8
Deepseek-Coder	6.7b	99.7	96.1	98.9
	33b	99.7	94.2	96.0

Table 4: The proportion of the correct SQL in the results that the linked entities and the skeleton are correct.

4.5.3 Is the skeleton not based on databases better than based on databases?

To verify whether the skeletons generated by LLMs are better than those generated directly, we evaluate the accuracy of the skeletons of generated SQL. The experimental results are shown in Table 5, from which we can observe: *(i)* Across all datasets and models, the accuracy of the parsed skeletons is higher than that of directly generated SQL, demonstrating the effectiveness of using parsed skeletons for correction; *(ii)* The improvement in skeleton accuracy is more significant in LLMs with smaller scales since small-scale models are less robust and more affected by disturbances in database information, leading to worse initial SQL.

Model	Scale	Method	Spider	Bird	Kaggle
Llama3	8b	Initial	32.0	8.1	18.2
		Parsed	**35.1**	**10.1**	**19.5**
	70b	Initial	36.1	11.7	24.3
		Parsed	**37.4**	**12.0**	**25.2**
Deep.C.	6.7b	Initial	36.0	13.2	15.4
		Parsed	**37.7**	**18.3**	**19.9**
	33b	Initial	35.0	11.5	17.5
		Parsed	**36.5**	**16.2**	**19.3**

Table 5: The skeleton accuracy of initial SQL and parsed skeletons by DAC. The best performance of each model and each dataset is marked in **bold**.

4.5.4 What is the performance of DAC using oracle entities and skeletons?

To explore the bottlenecks in improving the performance of DAC, we conduct experiments using oracle entities and skeletons. The experimental results are shown in Table 6, from which we can observe: *(i)* Introducing oracle entities or skeletons further improves the performance of DAC, demonstrating that enhancing entity linking or skeleton parsing can further enhance the performance; *(ii)* The performance improvement using oracle skeletons is greater than using oracle entities, indicating that the performance limitation of skeleton parsing is more severe than that of entity linking; *(iii)* The performance improvement using the oracle entities and skeletons is not significant, suggesting that direct correction by LLMs is not effective, where future work should explore more effective methods to align the SQL to the entities and skeletons.

4.5.5 What is the performance of DAC on SQLs with different hardness?

To evaluate the performance of DAC on questions of varying difficulty, we analyze the results of Spider, as shown in Table 7. From the table, we can observe: *(i)* Our method significantly improves performance across questions of different hardness, demonstrating its effectiveness on both easy and hard questions; *(ii)* The performance improvement is more pronounced on harder questions, showing that LLMs are likelier to make mistakes in the entity or skeleton of harder questions.

4.5.6 What is the main error of DAC?

To analyze the limitations of DAC and inspire future research, we analyze the error cases with and without DAC. The error cases are shown in Figure 4, from which we can observe: *(i)* Our method significantly reduces errors across all categories,

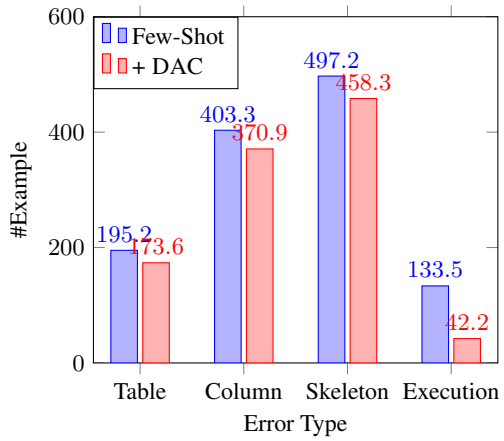

Figure 4: The error analysis with and without DAC. #Example denotes the average error examples across both models (Llama3, Deepseek-Coder) and all three datasets (Spider, Bird and KaggleDBQA). The table and column errors denote that the generated SQL contains incorrect tables or columns. The skeleton error denotes the incorrect skeleton. The execution error denotes that the generated SQL can not be executed.

demonstrating its effectiveness; *(ii)* The primary error remains in skeleton generation, showing that compared with entity linking, improving the skeleton accuracy demands more attention.

5 Related Works

5.1 Text-to-SQL

Research on the text-to-SQL task investigates how to generate corresponding SQL results based on given user questions and databases (Deng et al., 2022). The past research is mainly based on small-scale models (Guo et al., 2019; Bogin et al., 2019; Wang et al., 2020; Scholak et al., 2021; Dou et al., 2022). Currently, the methods based on LLMs have become the mainstream of the text-to-SQL task since their brilliant performance without fine-tuning (Hong et al., 2024b). One type of approach is to fine-tune LLMs to help them better adapt to the text-to-SQL task by directly using the text-to-SQL data (Zhang et al., 2024b; Li et al., 2024b; Pourreza and Rafiei, 2024), or merging the data of the tasks related to the text-to-SQL task (e.g., entity linking, knowledge generation) (Sun et al., 2024; Hong et al., 2024a). Another type of approach uses LLMs by few-shot inference, achieving significant performance improvement without the overhead of fine-tuning, like in-context learning (Pourreza and Rafiei, 2023; Gao et al., 2024; Chang and Fosler-Lussier, 2023), self-consistency (Ni et al., 2023; Zhong et al., 2023; Lee et al., 2024), and multi-agent (Wang et al., 2024a; Xie et al., 2024).

Model	Scale	Method	Spider	Bird	Kaggle
Llama3	8b	DAC	75.0	37.3	29.4
		+ Oracle Entity	75.9 (+0.9)	37.7 (+0.4)	29.8 (+0.4)
		+ Oracle Skeleton	75.6 (+0.6)	37.9 (+0.6)	30.5 (+1.1)
		+ Oracle Both	76.4 (+1.4)	38.0 (+0.7)	30.7 (+1.3)
	70b	DAC	81.3	48.4	35.1
		+ Oracle Entity	81.8 (+0.5)	48.6 (+0.2)	37.3 (+2.2)
		+ Oracle Skeleton	83.6 (+2.3)	48.9 (+0.5)	36.9 (+1.8)
		+ Oracle Both	83.8 (+2.5)	49.5 (+1.1)	38.6 (+3.5)
Deepssek-Coder	6.7b	DAC	77.1	44.7	25.4
		+ Oracle Entity	77.9 (+0.8)	44.7 (+0.0)	26.3 (+0.9)
		+ Oracle Skeleton	78.9 (+1.8)	44.8 (+0.1)	26.5 (+1.1)
		+ Oracle Both	78.9 (+1.8)	45.4 (+0.7)	27.2 (+1.8)
	33b	DAC	80.1	52.7	29.2
		+ Oracle Entity	80.7 (+0.6)	52.7 (+0.0)	31.1 (+1.9)
		+ Oracle Skeleton	81.0 (+0.9)	52.8 (+0.1)	30.0 (+0.8)
		+ Oracle Both	81.6 (+1.5)	53.1 (+0.4)	31.1 (+1.9)

Table 6: The performance of DAC with oracle entities and skeletons. Entity and Skeleton denote using the oracle entities and the oracle skeleton respectively, and Both denote using the oracle of them both.

Method	Easy	Medium	Hard	Extra
Llama3-8b	87.1	78.0	59.8	38.0
+ DAC	**89.9**	**81.8**	**61.5**	**48.8**
Llama3-70b	92.3	87.0	68.4	59.0
+ DAC	**94.0**	**86.5**	**69.0**	**61.4**
Deep.C.-6.7b	86.7	83.0	61.5	44.6
+ DAC	**90.3**	**84.8**	**66.7**	**47.6**
Deep.C.-33b	92.3	85.7	59.8	48.8
+ DAC	**95.2**	**86.8**	**62.1**	**58.4**

Table 7: Performance on questions with different hardness of the Spider dev set with and without DAC. Each model name denotes the few-shot performance. The best performance of different settings is marked in **bold**.

However, it is hard for the existing methods to effectively detect and correct errors of generated SQL (Zhang et al., 2024a; Kamoi et al., 2024), limiting their practical application. Therefore, we propose our method, which automatically assesses the correctness based on the decomposed sub-tasks of entity linking and skeleton parsing, thereby correcting the mistakes in the generated SQL.

5.2 Automated Correction

The automated correction approach first determines the correctness of the generated result after producing an answer, where if the answer is incorrect, it makes the necessary corrections (Pan et al., 2024). The most mainstream automated correction methods primarily focus on the code generation task, such as using feedback during training to correct errors (Ouyang et al., 2022; Feng et al., 2024; McAleese et al., 2024) or having the model judge and correct errors after inference (Chen et al., 2023a, 2024; Wu et al., 2024; Zhang et al., 2024c). For the text-to-SQL task, automated correction methods pay more attention to the characteristics of the task itself, such as correcting based on execution results from the database (Li and Xie, 2024; Zhong et al., 2023; Xia et al., 2024) or generating SQL-related error messages (Chen et al., 2023b; Liu and Tan, 2024; Askari et al., 2024).

However, previous work shows that LLMs struggle to directly identify SQL mistakes (Zhang et al., 2024a; Kamoi et al., 2024). Therefore, we propose DAC to provide decomposed feedback from the sub-tasks of entity linking and skeleton parsing, thereby helping correct mistakes.

6 Conclusion

In this paper, we try to alleviate the problem that LLMs perform poorly when directly correcting generated SQL for the text-to-SQL task. We first propose that correcting by decomposing the original task into sub-tasks is more effective than directly correcting the generated results. Based on this discussion, we propose DAC that decomposes the text-to-SQL task into two sub-tasks: entity linking and skeleton parsing. We ask LLMs to generate entity and skeleton information, identify inconsistencies with the initial SQL as feedback, and use this feedback for correction. Experimental results show that our method brings a 1.4% performance improvement on three mainstream text-to-SQL datasets compared with the previous correction method, showing the effectiveness of DAC.

Limitations

(i) We do not conduct complete experiments on more advanced models, such as `gpt-4o` (OpenAI et al., 2024), where we will conduct experiments with more advanced models on the complete datasets when the computing resources are sufficient in the future; *(ii)* We have limited types of sub-tasks for the correction with only experimented on two sub-tasks, entity link and skeleton parsing, where in the future, we will study error correction from the perspective of more types of sub-tasks.

Ethics Statement

All datasets and models used in this paper are publicly available, and our usage follows their licenses and terms.

References

Judith Jeyafreeda Andrew, Marc Vincent, Anita Burgun, and Nicolas Garcelon. 2024. Evaluating LLMs for temporal entity extraction from pediatric clinical text in rare diseases context. In *Proceedings of the First Workshop on Patient-Oriented Language Processing (CL4Health) @ LREC-COLING 2024*, pages 145–152, Torino, Italia. ELRA and ICCL.

Arian Askari, Christian Poelitz, and Xinye Tang. 2024. Magic: Generating self-correction guideline for in-context text-to-sql. *Preprint*, arXiv:2406.12692.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Global reasoning over database structures for text-to-SQL parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3659–3664, Hong Kong, China. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. 2024. Sqlfixagent: Towards semantic-accurate sql generation via multi-agent collaboration. *Preprint*, arXiv:2406.13408.

Shuaichen Chang and Eric Fosler-Lussier. 2023. Selective demonstrations for cross-domain text-to-SQL. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14174–14189, Singapore. Association for Computational Linguistics.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2023a. Codet: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Ziru Chen, Shijie Chen, Michael White, Raymond Mooney, Ali Payani, Jayanth Srinivasa, Yu Su, and Huan Sun. 2023b. Text-to-SQL error correction with language models of code. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1359–1372, Toronto, Canada. Association for Computational Linguistics.

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.

Shehzaad Zuzar Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason E Weston. 2024. Chain-of-verification reduces hallucination in large language models.

Longxu Dou, Yan Gao, Mingyang Pan, Dingzirui Wang, Jian-Guang Lou, Wanxiang Che, and Dechen Zhan. 2022. Unisar: A unified structure-aware autoregressive language model for text-to-sql. *arXiv preprint arXiv:2203.07781*.

Yunlong Feng, Yang Xu, Libo Qin, Yasheng Wang, and Wanxiang Che. 2024. Improving language model reasoning with self-motivated learning. *Preprint*, arXiv:2404.07017.

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

Akshay Goel, Almog Gueta, Omry Gilon, Chang Liu, Sofia Erell, Lan Huong Nguyen, Xiaohong Hao, Bolous Jaber, Shashir Reddy, Rupesh Kartha, Jean Steiner, Itay Laish, and Amir Feder. 2023. Llms accelerate annotation for medical information extraction. In *Proceedings of the 3rd Machine Learning for Health Symposium*, volume 225 of *Proceedings of Machine Learning Research*, pages 82–100. PMLR.

Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. 2023. Interleaving pre-trained language models and large language models for zero-shot nl2sql generation. *Preprint*, arXiv:2306.08891.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang. 2024a. Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain. In *Neural Information Processing*, pages 341–356, Singapore. Springer Nature Singapore.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. 2024b. Prompting gpt-3.5 for text-to-sql with desemanticization and skeleton retrieval. In *PRICAI 2023: Trends in Artificial Intelligence*, pages 262–274, Singapore. Springer Nature Singapore.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024c. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. 2024a. Knowledge-to-sql: Enhancing sql generation with data expert llm. *Preprint*, arXiv:2402.11517.

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

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Preprint*, arXiv:2406.01297.

Ali Buğra Kanburoğlu and F. Boray Tek. 2024. Text-to-sql: A methodical review of challenges and models. *Turkish Journal of Electrical Engineering and Computer Sciences*, 32(3):403–419. Publisher Copyright: © TÜBİTAK.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. 2023. CRUSH4SQL: Collective retrieval using schema hallucination for Text2SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14054–14066, Singapore. Association for Computational Linguistics.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online. Association for Computational Linguistics.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *Preprint*, arXiv:2405.07467.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to sql: Are we fully ready? *Preprint*, arXiv:2406.01265.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024b. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3).

Jinyang Li, Binyuan Hui, GE QU, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Peng Li, Tianxiang Sun, Qiong Tang, Hang Yan, Yuanbin Wu, Xuanjing Huang, and Xipeng Qiu. 2023c. CodeIE: Large code generation models are better few-shot information extractors. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15339–15353, Toronto, Canada. Association for Computational Linguistics.

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

Qian Liu, Dejian Yang, Jiahui Zhang, Jiaqi Guo, Bin Zhou, and Jian-Guang Lou. 2021. Awakening latent grounding from pretrained language models for semantic parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1174–1189, Online. Association for Computational Linguistics.

Xiping Liu and Zhao Tan. 2024. Epi-sql: Enhancing text-to-sql translation with error-prevention instructions. *Preprint*, arXiv:2404.14453.

Aman Madaan, Katherine Hermann, and Amir Yazdanbakhsh. 2023. What makes chain-of-thought prompting effective? a counterfactual study. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1448–1535, Singapore. Association for Computational Linguistics.

Nat McAleese, Rai (Michael Pokorny), Juan Felipe Cerón Uribe, Evgenia Nitishinskaya, and Maja Trebacz. 2024. Llm critics help catch llm bugs. Technical report, OpenAI.

Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date. Technical report, Meta.

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I. Wang, and Xi Victoria Lin. 2023. Lever: learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder,

Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2024. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. *Preprint*, arXiv:2402.01117.

Mohammadreza Pourreza, Davood Rafiei, Yuxi Feng, Raymond Li, Zhenan Fan, and Weiwei Zhang. 2024. Sql-encoder: Improving nl2sql in-context learning through a context-aware encoder. *Preprint*, arXiv:2403.16204.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Liang Shi, Zhengju Tang, and Zhi Yang. 2024. A survey on employing large language models for text-to-sql tasks. *Preprint*, arXiv:2407.15186.

Yinggang Sun, Ziming Guo, Haining Yu, Chuanyi Liu, Xiang Li, Bingxuan Wang, Xiangzhan Yu, and Tiancheng Zhao. 2024. Qda-sql: Questions enhanced dialogue augmentation for multi-turn text-to-sql. *Preprint*, arXiv:2406.10593.

Robert Vacareanu, Anurag Pratik, Evangelia Spiliopoulou, Zheng Qi, Giovanni Paolini, Neha Anna John, Jie Ma, Yassine Benajiba, and Miguel Ballesteros. 2024. General purpose verification for chain of thought prompting. *Preprint*, arXiv:2405.00204.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024a. Mac-sql: A multi-agent collaborative framework for text-to-sql. *Preprint*, arXiv:2312.11242.

Dingzirui Wang, Longxu Dou, Wenbin Zhang, Junyu Zeng, and Wanxiang Che. 2024b. Exploring equation as a better intermediate meaning representation for numerical reasoning of large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19116–19125.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. 2024. Large language models can self-correct with minimal effort. In *AI for Math Workshop @ ICML 2024*.

Hanchen Xia, Feng Jiang, Naihao Deng, Cunxiang Wang, Guojiang Zhao, Rada Mihalcea, and Yue Zhang. 2024. Sql-craft: Text-to-sql through interactive refinement and enhanced reasoning. *Preprint*, arXiv:2402.14851.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, MingXiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, ChengXiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm. *Preprint*, arXiv:2402.10671.

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. *Preprint*, arXiv:2303.10420.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024a. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *Preprint*, arXiv:2403.02951.

Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024b. Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. In *Companion of the 2024 International Conference on Management of Data*, SIGMOD/PODS '24, page 93–105, New York, NY, USA. Association for Computing Machinery.

Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. 2024c. Self-contrast: Better reflection through inconsistent solving perspectives. *Preprint*, arXiv:2401.02009.

Ruiqi Zhong, Charlie Snell, Dan Klein, and Jason Eisner. 2023. Non-programmers can label programs indirectly via active examples: A case study with

text-to-SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5126–5152, Singapore. Association for Computational Linguistics.

A Prompts of DAC

The prompts of DAC is shown in Table 8, Table 9, Table 10, Table 11, and Table 12.

B Comparison with Other SQL Correction Methods

We compare our method with other text-to-SQL correction methods, as shown in Table 13. It can be observed that on GPT-3.5, our method achieves better performance improvement despite having a lower baseline performance, demonstrating that our method outperforms existing text-to-SQL correction methods on GPT-3.5.

C Case Study

Despite the effectiveness of DAC is demonstrated in Table 1, how our method specifically enhances text-to-SQL performance remains to be studied. Therefore, we conduct a case study, as shown in Figure 5. From the figure, we can see that, without our method, the model output misses using entities and generates an incorrect skeleton. With our method, the model generates the correct skeleton and utilizes all relevant entities, thereby enhancing text-to-SQL performance.

The prompt of SQL Generation.

Generate a SQL to answer the question with the given schema.
Quote your answer with:
“‘sql
<answer sql>
“‘

—

For example:

“‘sql
{schema of demonstrations}
“‘

Question: {question of demonstrations}
“‘sql
{sql of demonstrations}
“‘

—

...

—

Based on the instruction and the examples, answer the following question:

“‘sql
{schema}
“‘

Question: {question}

Table 8: The prompt of SQL Generation of DAC.

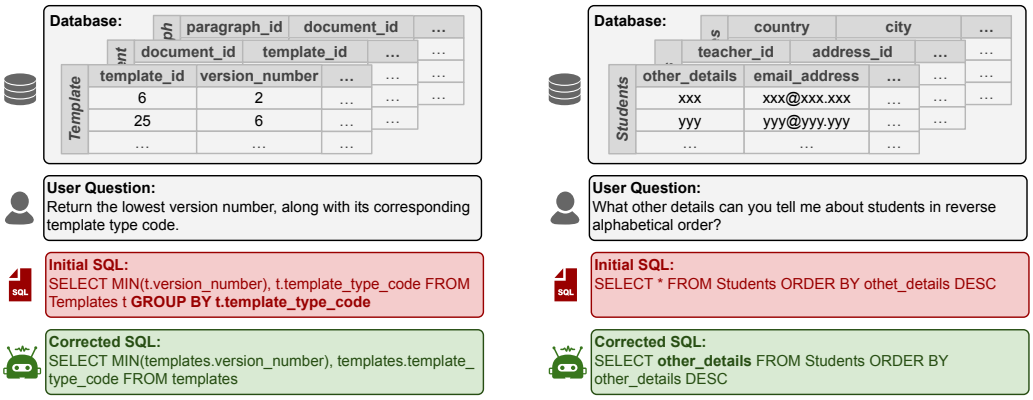

Figure 5: The case study with and without DAC of the Spider dev set using Llama3-70b. The mistake in the SQL is annotated with **bold**.

The prompt of Entity Linking.

Align the tokens in the given question to the table entities or the column entities of the schema above, considering the given SQL.
Present the aligned tokens in the python format List[Dict[str, str]], where each Dict[str, str] denoting each token in the question containing the following keys:
{
"token": the token in the question
"schema": the schema entity aligned to the token
"type": the type of the entity aligned to the token
}
The "type" can be one of the following:
* "tbl": the table name
* "col": the column name
* "val": the value
"schema" and "type" are either both null or not null at the same time.
 	
Here are some examples.
 	
—
 	
{schema of demonstrations}
 	
SQL: {sql of demonstrations}
Question: {question of demonstrations}
Alignments: {alignment of demonstrations}
 	
—
 	
...
 	
—
 	
Based on the instruction and the examples above, solve the following question:
 	
{schema}
 	
SQL: {sql}
Question: {question}
Alignments:

Table 9: The prompt of Entity Linking of DAC.

The prompt of Skeleton Parsing.

Hallucinate a SQL to answer the question.
Quote your answer with:
“‘sql
<answer sql>
“‘

—

For example:

Question: {question of demonstrations}
“‘sql
{sql of demonstrations}
“‘

—

...

—

Based on the instruction and the examples, answer the following question:

Question: {question}

Table 10: The prompt of Skeleton Parsing of DAC.

The prompt of Correction with the entity feedback.

“‘sql
{schema}
“‘

Fix the sql "{sql}" to answer the question "{question}" based on the above database and the alignment.
Present your sql in the format:
“‘sql
<your sql>
“‘
It should be noticed that {notification}. Your sql must contain the tables and columns mentioned by the question.

Table 11: The prompt of Correction of DAC with the entity feedback. The format of “{notification}” is like “<tables or columns> are mentioned by the question”.

The prompt of Correction with the skeleton feedback.

“‘sql
{schema}
“‘

Fix the sql "{sql}" to answer the question "{question}" with the above schema.
Present your sql in the format:
“‘sql
<your sql>
“‘
It should be noticed that the SQL skeleton could be like "{skeleton}", where each ’_’ can only be replaced with one single table, column or value.

Table 12: The prompt of Correction of DAC with the skeleton feedback.

Model	Method	Spider	Bird
gpt-3.5-turbo	Self-Debug (Chen et al., 2024)	71.1 → 72.2	–
	SQL-CRAFT (Xia et al., 2024)	78.2 → 79.3	–
	DAC	76.9 → **80.6**	40.0 → 44.9
gpt-4	Self-Debug (Chen et al., 2024)	73.2 → 73.6	–
	EPI-SQL (Liu and Tan, 2024)	79.7 → 85.1	–
	SQL-CRAFT (Xia et al., 2024)	79.7 → 85.4	53.3 → 55.2
	MAGIC (Askari et al., 2024)	78.6 → **85.7**	–

Table 13: The comparison of DAC with other text-to-SQL correction methods on GPT-3.5 and gpt-4. Since the performance is affected by the API version, we report the performance change after using each method. We do not report the performance on KaggleDBQA since other text-to-SQL correction methods do not report the performance in their paper on this dataset. We do not adapt DAC to gpt-4 on the complete evaluation datasets due to the high cost of the API.