# CoE-SQL: In-Context Learning for Multi-Turn Text-to-SQL with Chain-of-Editions

**Hanchong Zhang**[1], **Ruisheng Cao**[1], **Hongshen Xu**[1], **Lu Chen**[1,2*] **and Kai Yu**[1,2*]
[1]X-LANCE Lab, Department of Computer Science and Engineering
MoE Key Lab of Artificial Intelligence, SJTU AI Institute
Shanghai Jiao Tong University, Shanghai, China
[2]Suzhou Laboratory, Suzhou, China
{zhanghanchong,chenlusz,kai.yu}@sjtu.edu.cn

## Abstract

Recently, Large Language Models (LLMs) have been demonstrated to possess impressive capabilities in a variety of domains and tasks. We investigate the issue of prompt design in the multi-turn text-to-SQL task and attempt to enhance the LLMs' reasoning capacity when generating SQL queries. In the conversational context, the current SQL query can be modified from the preceding SQL query with only a few operations due to the context dependency. We introduce our method called CoE-SQL[1] which can prompt LLMs to generate the SQL query based on the previously generated SQL query with an edition chain. We also conduct extensive ablation studies to determine the optimal configuration of our approach. Our approach outperforms different in-context learning baselines stably and achieves state-of-the-art performances on two benchmarks SParC and CoSQL using LLMs, which is also competitive to the SOTA fine-tuned models.

## 1 Introduction

Text-to-SQL (Zhong et al., 2017; Xu et al., 2017) is a semantic parsing task that translates the natural language question into the appropriate SQL query according to the given database schema. This technique is critical in building a natural language interface to relational databases (Androutsopoulos et al., 1995), which alleviates the burden on programmers to curate valid and correct annotations.

In this work, we focus on the contextual settings (Yu et al., 2019b,a) where users interact with the system in multi-turn scenarios. In each turn, the text-to-SQL parser understands and interprets the current user question into a SQL query based on the entire dialogue history. Considering the requirement of context modeling, EditSQL (Zhang et al., 2019) introduces copy mechanism during the generation of SQL queries to re-use the SQL spans in history. DELTA (Chen et al., 2021b) firstly paraphrases the long context into a single question and transforms the original problem into single-turn parsing. IGSQL (Cai and Wan, 2020) and HIESQL (Zheng et al., 2022) both utilize the concept of cross-turn schema interaction into the specialized design of graph encoders. STAR (Cai et al., 2022) and CQR-SQL (Xiao et al., 2022) inject the capability of context modeling through self-supervised pre-training tasks. Though remarkable progress has been achieved, these task-specific methods rely on complicated model design which is labor-intensive and training-inefficient.

The advent of LLMs (Ouyang et al., 2022) provides a promising solution in the text-to-SQL field, that is in-context learning with few-shot exemplars (Brown et al., 2020). In this branch, DIN-SQL (Pourreza and Rafiei, 2023) adopts the concept of Chain-of-Thought (CoT, Wei et al., 2023) and establishes a new milestone in single-turn settings. ACT-SQL (Zhang et al., 2023) further eliminates the necessity of mannually crafting CoT exemplars and automatically constructs useful demonstraions. However, adapting the ICL method into contextual settings is less investigated. ACT-SQL attempts to reduce the original problem into single-turn problems via sentence rewriting. But poor performances are reported on multi-turn benchmarks due to severe error propagation.

In this work, we propose Chain-of-Edition framework to tackle the multi-turn text-to-SQL task (CoE-SQL). Inspired by recent progress in code edition (Chen et al., 2023b), we attempt to model the changes in the output SQL program across different interaction turns explicitly. These changes can be serialized into the reasoning thought, as a specific implementation of CoT, to track the shift in the user focus and intention. The underlying mechanism is that, in a coherent and

[*] The corresponding authors are Lu Chen and Kai Yu.
[1]Our code is publicly available at https://github.com/X-LANCE/text2sql-multiturn-GPT

*Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6487–6508
June 16-21, 2024 ©2024 Association for Computational Linguistics

**Turn 1**

**Q:** *What are the phone names available in Alberta?*

**SQL:** SELECT T1.Name FROM phone AS T1 JOIN phone_market AS T2 JOIN market AS T3 ON T1.Phone_ID = T2.Phone_ID AND T2.Market_ID = T3.Market_ID WHERE T3.District = "Alberta"

**Edit:** no edition

**Turn 2**

**Q:** *Show the district as well.*

**SQL:** SELECT T1.Name, T3.District FROM phone AS T1 JOIN phone_market AS T2 JOIN market AS T3 ON T1.Phone_ID = T2.Phone_ID AND T2.Market_ID = T3.Market_ID WHERE T3.District = "Alberta"

**Edit:** add select column T3.District

**Turn 3**

**Q:** *Remove the restriction of those in Alberta only, show all instead.*

**SQL:** SELECT T1.Name, T3.District FROM phone AS T1 JOIN phone_market AS T2 JOIN market AS T3 ON T1.Phone_ID = T2.Phone_ID AND T2.Market_ID = T3.Market_ID ~~WHERE T3.District = "Alberta"~~

**Edit:** delete where clause T3.District = "Alberta"

Table 1: A multi-turn example from SParC (Yu et al., 2019b). Each edition is based on the previous turn.

consistent dialogue, the user's questions often depend on the previous focus, and the latest request or intent can be obtained by modifying the already generated semantic representations (SQL program) through a few simple rules. For example, in Table 1, after attaining the raw SQL query in turn one, the user is too "lazy" to declare the full intention and only convey the difference. This can be easily captured by a simple column insertion on the target SQL query. Similarly, in turn 3, the complete SQL can be obtained via a simple deletion of the WHERE clause based on turn 2, instead of generating the tedious long output. To achieve this, we thoroughly analyze the entire training set and summarize 14 unit edit rules (3.2). Next, we propose an abstract syntax tree (AST) comparison algorithm to automatically extract the chain of edition rules with the minimum length (3.3). After that, we serialize and prepend those editions in the prompt before the output of each turn. Different serialization styles are analyzed (3.4), including self-defined edit rules, python code and natural language description. And we find that the NL description performs the best on two benchmarks, SParc (Yu et al., 2019b) and CoSQL (Yu et al., 2019a).

Our contributions can be summarized:

1. We propose CoE-SQL to tackle the complex multi-turn text-to-SQL, which formalizes the SQL editions as a specific reasoning process. This method is more interpretable towards how LLM deals with context modeling to simulate human thinking.

2. We provide the checklist of unit edit rules, and the corresponding tree comparison algorithm to automatically extract the edition chain by comparing two abstract syntax trees (ASTs).

3. We conduct comprehensive ablation study to analyze different CoE configurations and achieve state-of-the-art results with LLMs on the validation sets of two benchmarks SParC and CoSQL. It is also competitive to SOTA fine-tuned models.

## 2 Related Work

**Multi-turn text-to-SQL models** Before LLMs are applied in the multi-turn text-to-SQL task, researches mainly focus on building and fine-tuning specialized deep neural networks. (Zhang et al., 2019) and (Wang et al., 2020) use the previously generated SQL queries to improve the parsing accuracy. IGSQL (Cai and Wan, 2020) utilizes the graph neural network to model database schema items in the conversational scenario. R$^2$SQL (Hui et al., 2021) and HIE-SQL (Zheng et al., 2022) present a dynamic schema-linking graph which incorporates the current utterance, the previous preceding utterances, the database schema, and the last most recent SQL query. RASAT (Qi et al., 2022) is a Transformer (Vaswani et al., 2023) architecture augmented with relation-aware self-attention that could leverage a variety of relational structures while effectively inheriting the pre-trained parameters from the T5 model (Raffel et al., 2023). RASAT employs the PICARD method (Scholak et al., 2021) which constrains the auto-regressive decoder by rejecting invalid tokens.

Despite the impressive results of specialized models, there are some unavoidable drawbacks. Creating and labeling a comprehensive text-to-SQL dataset requires a significant amount of resources and time. Additionally, training and refining the model is a laborious process that requires a lot of computing power.

**In-context learning for text-to-SQL** Recent studies have explored the potential of LLMs for the text-to-SQL task, with Rajkumar et al. (2022)

using the zero-shot and few-shot learning setting to empirically evaluate the capabilities of LLMs such as GPT-3 (Brown et al., 2020) and Codex (Chen et al., 2021a). Nan et al. (2023) focused on the strategy of exemplar selection, requiring an additional predictor to assess the difficulty of the SQL. DIN-SQL (Pourreza and Rafiei, 2023) provides a more complex approach, decomposing the problem into several simpler sub-problems.

The above works merely employ LLMs on the single-turn text-to-SQL task. ACT-SQL (Zhang et al., 2023) generates the chain-of-thoughts automatically and extends its approach onto the multi-turn text-to-SQL task. ACT-SQL converts the multi-turn dataset into the single-turn one by rewriting and completing questions with context dependencies. However, ACT-SQL performs poorly under the multi-turn setup due to the error propagation occurring in the process of question rewriting. In contrast, our proposed CoE-SQL is an edit-based method which can directly utilize the context dependency instead of rewriting the question.

# 3 Methodology

In the few-shot in-context learning setting, the multi-turn text-to-SQL task can be formulated as

$$R_n = \mathrm{LLM}(I, D, \mathcal{Q}_{\leq n}, \mathcal{R}_{<n}, \mathcal{E}).$$

$R_n$ represents the response to the current question created by LLMs. $I$ represents the instruction. $D$ represents the database schema. $\mathcal{Q} = [Q_1, Q_2, \cdots, Q_n]$ represents the entire context consisting of $n$ questions. $\mathcal{R} = [R_1, R_2, \cdots, R_{n-1}]$ represents LLMs' responses to the previous questions. $\mathcal{E} = [E_1, E_2, \cdots, E_{|\mathcal{E}|}]$ is the list of $|\mathcal{E}|$ exemplars used in few-shot learning.

## 3.1 Overview of CoE

In the real-world scenario, users are more likely to start the conversation with a relatively simple question because they are unfamiliar with the detailed structure of the system. With the increasing the number of conversation turns, the user question and the corresponding SQL query will become more complex. It is more difficult and redundant for LLMs to generate a complex SQL query from scratch, since the entire thinking and logical reasoning process is generally intricate. On the contrary, generating the current SQL query by updating the previous one through a few editions is a better option.

In Section 3.2, we provide our definition of unit edit rules that can help edit the SQL query. In Section 3.3, we explain how to extract the edition chain by comparing the two ASTs of two SQL queries. In Section 3.4, we introduce the different styles of edition chains used in our work. And finally in Section 3.5, we provide a simple method to help LLMs better analyze the edition process.

## 3.2 Definition of Unit Edit Rules

In order to edit a SQL query into another SQL query, we first define the set of unit edit rules. According to the different SQL components, we totally define 14 unit edit rules shown in Table 13. Taking the conversation instance in Table 1 as an example, we can apply the unit edit rule EditSelectItem(-, market.District) to edit SQL 1 into SQL 2. We can also apply the EditWhereCondition(market.District = “Alberta”, -) unit edit rule to edit SQL 2 into SQL 3.

## 3.3 Extraction of Edition Chains

We use the few-shot learning method to activate LLMs' ability of utilizing our pre-defined unit edit rules. Therefore, we need to select exemplars from the training dataset and then extract the edition chains in each conversation. Since this work does not focus on selecting better exemplars, we use a simple exemplars selection strategy. We first randomly choose $k_d$ database schemas and then randomly choose $k_e$ dataset examples for each database schema. Thus, total $k_d \times k_e$ exemplars are put in the prompting text for the few-shot learning. In the following process, we need to extract the edition chains from these exemplars.

Assume that the dataset example consists of $n$ questions $[Q_1, Q_2, \cdots, Q_n]$ and $n$ corresponding SQL queries $[S_1, S_2, \cdots, S_n]$. Consider the $i$-th one as the current turn. Notably, the edition chain is determined by the difference between the current SQL query $S_i$ and the previous SQL query $S_j (j < i)$. A reasonable approach to extracting the edition chain is to compare the two ASTs.

Figure 1 shows an example of a comparison between two ASTs. Notice that the FROM clause component is omitted in this figure. We compare each node pair in the two ASTs. Two nodes are considered equal iff they represent the same grammar rule and all of their child nodes are equal. The edition chain can be constructed according to the unequal part. By recognizing the grammar rules of the nodes in the unequal part, we can determine

**Algorithm 1:** Extraction of Edition Chains

```
Input  : Previous tree node $t_{\text{old}}$, current tree node $t_{\text{new}}$
Output : Edition chain $C$
$C \leftarrow \{\}$;
for $f$ in get_fields($t_{\text{old}}$) do
    $s_{\text{old}} \leftarrow t_{\text{old}}$.get_son($f$);
    $s_{\text{new}} \leftarrow t_{\text{new}}$.get_son($f$);
    if $s_{\text{old}}$ is tree and $s_{\text{new}}$ is tree then
        $C_{\text{sub}} \leftarrow$ ExtractCoE($s_{\text{old}}, s_{\text{new}}$);
        // recursively call the function
        $C$.update($C_{\text{sub}}$);
    end
    else if $s_{\text{old}}$ is None and $s_{\text{new}}$ is tree then
        $e \leftarrow$ get_add_edition($s_{\text{new}}, f$);
        $C$.add($e$);
    end
    else if $s_{\text{old}}$ is tree and $s_{\text{new}}$ is None then
        $e \leftarrow$ get_delete_edition($s_{\text{old}}, f$);
        $C$.add($e$);
    end
    else if $s_{\text{old}} \neq s_{\text{new}}$ then
        $e \leftarrow$
          get_change_edition($s_{\text{old}}, s_{\text{new}}, f$);

        $C$.add($e$);
    end
end
return $C$
```

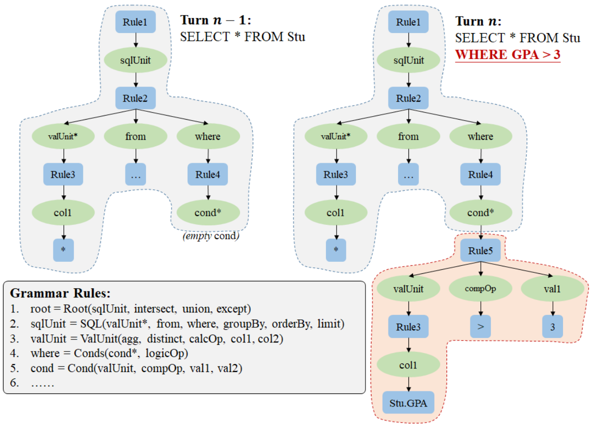
Turn $n-1$:
SELECT * FROM Stu
Turn $n$:
SELECT * FROM Stu
WHERE GPA > 3
Rule1
sqlUnit
Rule2
valUnit*
from
where
Rule3
…
Rule4
col1
cond*
*
(empty cond)
Rule5
valUnit
compOp
val1
Rule3
>
3
col1
Stu.GPA
Grammar Rules:
1. root = Root(sqlUnit, intersect, union, except)
2. sqlUnit = SQL(valUnit*, from, where, groupBy, orderBy, limit)
3. valUnit = ValUnit(agg, distinct, calcOp, col1, col2)
4. where = Conds(cond*, logicOp)
5. cond = Cond(valUnit, compOp, val1, val2)
6. ……

Figure 1: Comparison between two ASTs.

that the edition chain contains exactly one unit edit rule, i.e. EditWhereCondition(-, Stu.GPA > 3). The detailed procedure is outlined in Algorithm 1. With the method of comparing two ASTs, we can construct the edition chains for every exemplar automatically. LLMs can learn the chain-of-editions method during the few-shot learning process.

In quite a few conversations, the current question $Q_i$ may not inherit from the exactly previous question $Q_{i-1}$ but the more previous one $Q_j (j < i-1)$. Also, the current question may start a new topic irrelevant to the original one. To tackle the above two problems, we enumerate all the previous $i-1$ SQL queries and construct an edition chain $\text{CoE}_{i,j}$ for each SQL pair $(S_i, S_j)$ where $j = 1, 2, \cdots, i-1$. We eliminate those edition chains with lengths greater than $l_c$ (edition chains consisting of more than $l_c$ unit edit rules) where $l_c$ is a threshold. If the edition chain contains too many unit edit rules, we believe that the corresponding two questions are irrelevant. In that case, writing the current SQL query directly is more efficient than editing from the previous one. As for the left edition chains, we select the edition chain with the least number of tokens as the final edition chain.

### 3.4 Style of Edition Chains

We attempt to use three different styles to put the edition chain in the prompting text. Table 2 shows the detailed formats where the example chain-of-edition consists of two unit edit rules, namely EditSelectItem(*, COUNT(*)) and EditWhereCondition(-, Stu.GPA > 3).

With the edit rule style, we directly write our unit edit rules into the prompting text. With the Python code style, we regard the SQL query as a Python dict and convert unit edit rules into corresponding Python codes to update the Python dict. With the natural language style, unit edit rules are converted into plain texts which are closer to the corpus used in the LLMs' pretraining step.

### 3.5 Analysis of Differences Between Pre- and Post-Questions

When preprocessing the exemplar, the edition chain can be extracted by comparing the different ASTs. However, when handling the test case, LLMs have to predict the edition chain by comparing the current question and the previous question. Thus we complete the question analysis for the exemplars. Given the current question and the previous question, we instruct LLMs to generate the difference

| Style | Prompting Text |
|---|---|
| Edit Rule | FROM clause:<br>- no change is needed<br>SELECT clause:<br>- EditSelectItem(*, COUNT(*))<br>WHERE clause:<br>- EditWhereCondition(-, Stu.GPA > 3)<br>GROUP BY clause:<br>- no change is needed<br>ORDER BY clause:<br>- no change is needed<br>LIMIT clause:<br>- no change is needed<br>INTERSECT/UNION/EXCEPT:<br>- no change is needed |
| Python Code | sql[’select’].remove(’*’)<br>sql[’select’].append(’COUNT(*)’)<br>sql[’where’][’conditions’].append(’Stu.GPA > 3’) |
| Natural Language | FROM clause:<br>- no change is needed<br>SELECT clause:<br>- change * to COUNT(*)<br>WHERE clause:<br>- add WHERE condition Stu.GPA > 3<br>GROUP BY clause:<br>- no change is needed<br>ORDER BY clause:<br>- no change is needed<br>LIMIT clause:<br>- no change is needed<br>INTERSECT/UNION/EXCEPT:<br>- no change is needed |

Table 2: Three styles of serialization for CoE.

between them. The analysis texts of each exemplar are also added into the prompting text, which can motivate LLMs to analyze the difference between pre- and post-questions. Notice that, this thought-before-edition trick is an adaptation of the thought-before-action idea in ReAct (Yao et al., 2022) framework.

## 4 Experiments

### 4.1 Experiment Setup

**Models** We use the GPT-3.5-turbo-16k model to evaluate our proposed CoE-SQL. The CoE-SQL approach is based on self-defined edit operations, indicating that this method would not perform well if insufficient exemplars are provided. Only sufficient exemplars can cover most unit edit rules. Therefore, we expect LLMs to have a long context window. That's the reason we choose GPT-3.5-turbo-16k with a 16,385 tokens context window instead of GPT-3.5-turbo with a 4,096 tokens context window.

**Hyperparameters** The temperature in the API of LLMs is set to 0, meaning that the greedy decoding strategy is being used. Text-to-SQL tasks require the model to produce SQL queries that adhere to strict grammar regulations. If the temperature is too high, the LLMs are likely to generate SQL queries that are invalid or not pertinent to the posed questions. Regarding the exemplars used for the few-shot learning, we set the number of database schemas $k_d$ to 4 and set the number of examples from each database schema $k_e$ to 4.

**Datasets** We assess our proposed approach on SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a). SParC is composed of 4,298 coherent question sequences, including more than 12k individual questions and the related SQL queries. CoSQL has 10k+ annotated SQL queries. Each dialogue in CoSQL is designed to mimic a real-world situation, where a regular user is exploring the database and an expert is retrieving answers with SQL. They also provide an evaluation script that divides SQL queries into four difficulty levels (easy, medium, hard, and extra).

**Evaluation metrics** We assess the performance of our approach using three commonly used evaluation metrics for the text-to-SQL task: exact match accuracy (EM), execution accuracy (EX), and test-suite accuracy (TS). EM requires that each component of the predicted SQL is the same as the corresponding component of the gold SQL, disregarding the values in the query. EX evaluates the correctness of the execution result of the predicted SQL, which is usually more precise than EM. TS also evaluates the execution result, but requires the result to be correct across multiple database instances per database schema[2].

Since we are evaluating LLMs' performances in the multi-turn text-to-SQL task, question match accuracy (QM) and interaction match accuracy (IM) need to be considered respectively. QM is 1 if the predicted SQL query for the single question is correct, and IM is 1 if all the predicted SQL queries in the context are correct.

### 4.2 Main Results

In our main experiments, we choose the natural language style for the edition chain. We set the maximum length of the edition chain $l_c$ to 4 when testing on SParC and 3 when testing on CoSQL. Table 3 and Table 4 show the performance of our proposed CoE-SQL and other previous works on the dev sets of SParC and CoSQL respectively.

Notably, when comparing in-context learning approaches with fine-tuned models, the EM evalua-

[2] https://github.com/taoyds/test-suite-sql-eval

| Fine-tuned Model | QM | | | IM | | |
|---|---|---|---|---|---|---|
| | EM ↑ | EX ↑ | TS ↑ | EM ↑ | EX ↑ | TS ↑ |
| GAZP+BERT (Zhong et al., 2020) | 48.9 | 47.8 | - | - | - | - |
| HIE-SQL+GraPPa (Zheng et al., 2022) | 64.7 | - | - | 45.0 | - | - |
| RASAT+PICARD (Qi et al., 2022) | **67.7** | **73.3** | - | **49.1** | **54.0** | - |
| **In-Context Learning Approach** | | | | | | |
| ACT-SQL (Zhang et al., 2023) | 51.0 | 63.8 | 56.9 | 24.4 | 38.9 | 29.6 |
| Baseline (Ours) | 50.0 | 67.0 | 59.5 | 30.8 | 46.7 | 37.9 |
| **CoE-SQL (Ours)** | **56.0** | **70.3** | **63.3** | **36.5** | **50.5** | **41.9** |

Table 3: Performances of CoE-SQL and other previous works on SParC dev set.

| Fine-tuned Model | QM | | | IM | | |
|---|---|---|---|---|---|---|
| | EM ↑ | EX ↑ | TS ↑ | EM ↑ | EX ↑ | TS ↑ |
| GAZP+BERT (Zhong et al., 2020) | 42.0 | 38.8 | - | - | - | - |
| HIE-SQL+GraPPa (Zheng et al., 2022) | 56.4 | - | - | **28.7** | - | - |
| RASAT+PICARD (Qi et al., 2022) | **58.8** | **67.0** | - | 27.0 | **39.6** | - |
| **In-Context Learning Approach** | | | | | | |
| ACT-SQL (Zhang et al., 2023) | 46.0 | 63.7 | 55.2 | 13.3 | 30.7 | 21.5 |
| Baseline (Ours) | 47.8 | 69.4 | 58.5 | 20.1 | 38.9 | 27.6 |
| **CoE-SQL (Ours)** | **52.4** | **69.6** | **60.6** | **23.9** | **39.6** | **30.4** |

Table 4: Performances of CoE-SQL and other previous works on CoSQL dev set.

tion metric is not that worthy to be paid attention to. Fine-tuned models can learn the dataset feature from the training set. These models are more likely to generate the SQL query with the same structure as the gold SQL query and thus can achieve higher EM scores. On the contrary, LLMs tend to write the SQL query based on their original knowledge learning in the pretraining phase. Only a few exemplars from the training dataset cannot provide sufficient information about the dataset feature. Therefore, LLMs are more likely to generate the SQL query with the accurate semantic and logic and the correct execution result. In general, we would like to mainly focus on the EX and TS evaluation metrics in the following discussion. Most fine-tuned models only provide their EM scores. We compare our method with the GAZP and the RASAT methods because these two models provide their EX scores, where RASAT is the SOTA one.

Compared with fine-tuned models, our proposed CoE-SQL approach achieves a 70.3% EX(QM) score and a 50.5% EX(IM) score on SParC dev set, which has surpassed the GAZP + BERT model (Zhong et al., 2020) a lot and has been comparable to the RASAT + PICARD model (Qi et al., 2022). CoE-SQL even achieves the highest EX(QM) score on CoSQL dev set. The experiment result proves that LLMs have possessed the strong ability for handling the complex multi-turn text-to-SQL task. Using the GPT-3.5-turbo-16k LLM, the CoE-SQL approach can perform almost as well as the previous best fine-tuned model (with EX score). We believe that our CoE-SQL can achieve a better performance if larger LLMs (e.g. GPT-4) are applied.

Furthermore, the CoE-SQL approach achieves the highest EM, EX, and TS scores among the existing in-context learning methods. The ACT-SQL method converts the multi-turn dataset into the single-turn one by rewriting and completing the questions with context dependencies. Comparing our simple baseline method and the ACT-SQL method, we can conclude that paraphrasing the multi-turn dataset with LLMs is not a good choice. It performs even worse than the baseline method. Based on the edit operations, CoE-SQL performs much better than the baseline method which merely takes the original database schema and questions as the LLMs' input. This indicates that editing SQL queries is an effective method in the multi-turn text-to-SQL task.

We also complete the experiments on SParC and CoSQL test sets. The results of the experiment are shown in Table 5. Under the EX metric, our CoE-

| Fine-tuned Model | SParC |  | CoSQL |  |
|---|---|---|---|---|
|  | EX (QM) ↑ | EX (IM) ↑ | EX (QM) ↑ | EX (IM) ↑ |
| GAZP+BERT (Zhong et al., 2020) | 44.6 | 19.7 | 35.9 | 8.4 |
| RASAT+PICARD (Qi et al., 2022) | 74.0 | **52.6** | 66.3 | 37.4 |
| **In-Context Learning Approach** |  |  |  |  |
| **CoE-SQL (Ours)** | **74.1** | 51.9 | **71.1** | **42.9** |

Table 5: Performances of CoE-SQL and other previous works on SParC and CoSQL test sets.

SQL approach keeps the good performance and even surpasses the SOTA fine-tuned model on the test sets. This also reveals that in-context learning approaches would not suffer the performance gap between the dev set and the test set.

**Case study** Table 6 shows an instance from the SParC dev set. The condition "employees under age 30" is mentioned in the first question. The baseline method successfully keeps this condition in the second query but forgets it in the final query. However, due to the chain-of-edition mechanism, the CoE-SQL method retains this condition from the beginning to the end. More detailed case studies can be found in Appendix B.

**Turn 1**
**Question:** *Find all employees who are under age 30.*
**Baseline:** SELECT * FROM employee WHERE age < 30 ✓
**CoE-SQL:** SELECT * FROM employee WHERE age < 30 ✓

**Turn 2**
**Question:** *Which cities did they come from?*
**Baseline:** SELECT DISTINCT city FROM employee WHERE age < 30 ✓
**CoE-SQL:** SELECT City FROM employee WHERE age < 30 ✓

**Turn 3**
**Question:** *Show the cities from which more than one employee originated.*
**Baseline:** SELECT city FROM employee GROUP BY city HAVING count(*) > 1 ✖
**CoE-SQL:** SELECT City FROM employee WHERE age < 30 GROUP BY City HAVING COUNT(*) > 1 ✓

Table 6: Case study from the SParC dev set.

### 4.3 Ablation Study

In this section, we show the experiment results of the ablation study to find out the best setting of CoE-SQL. We only report the TS score since it is most suitable for the in-context learning method.

**Simple CoT baseline** Besides the most trivial baseline method, we attempt an advanced approach based on the simple CoT method. We manually label the few-shot exemplars using the question explanation mentioned in (Chen et al., 2023a). Table 7 shows the result, which proves that the simple CoT method can be beneficial for LLMs compared with the most trivial baseline. It is also significant that our CoE-based approach is more suitable for the multi-turn text-to-SQL task.

| Approach | TS(QM) ↑ | TS(IM) ↑ |
|---|---|---|
| Baseline | 59.5 | 37.9 |
| Simple CoT | 61.8 | 39.8 |
| CoE-SQL | **63.3** | **41.9** |

Table 7: Performances of CoE-SQL and baseline methods on SParC dev set.

**Style of edition chains** Table 8 shows the performance of CoE-SQL on SParC dev set influenced by three styles of prompting text for chain-of-editions mentioned in Section 3.4. The experiment result proves that the natural language style is the most suitable one. This is because LLMs like GPT models are mostly trained with natural language corpuses. The chain-of-editions style with the edit rule performs relatively poor, since our unit edit rules are self-designed and very unlikely to appear in the pretraining corpus. Although LLMs must have seen many Python codes during pretraining, the Python-code style still receives bad scores. We believe that this is because the Python codes generated by LLMs are used to update the Python dict that represents the SQL query. The structure of this Python dict is complex and unfamiliar for LLMs, though we have provided the structure in the instruction and exemplars.

When using the Python-code style, the prompting text mainly consists of three parts, i.e. the

| Style | TS(QM) ↑ | TS(IM) ↑ |
|---|---|---|
| Edit Rule | 61.2 | 40.5 |
| Python Code | 58.6 | 37.9 |
| Natural Language | **63.3** | **41.9** |

Table 8: CoE-SQL performance on SParC dev set influenced by three styles of prompting text for chain-of-editions.

Python code that represents the edit rule, the Python dict that represents the SQL components, and the current SQL query. Thus we complete more ablation studies about these LLMs-generated parts. Table 9 shows the result. First, we change the order of the Python dict and the SQL query in the exemplar. Second, we complete the post-processing according to different parts which are marked with "*" symbols in the table. When post-processing with the code, we run the LLM-generated code to update the dict and get the SQL by parsing the updated dict. When post-processing with the dict, we directly get the SQL by parsing the dict generated by LLMs. The experiment shows that appending the dict after the SQL is generally a better choice. The Python codes generated by LLMs are not that reliable. It is better to directly use the Python dict or the SQL itself.

| Python-Code Style | TS(QM) ↑ | TS(IM) ↑ |
|---|---|---|
| code*+dict+SQL | 51.8 | 28.7 |
| code+dict*+SQL | 57.2 | 34.8 |
| code+dict+SQL* | 58.3 | 36.3 |
| code*+SQL+dict | 52.5 | 29.1 |
| code+SQL+dict* | **58.6** | **37.9** |
| code+SQL*+dict | 56.9 | 36.0 |

Table 9: CoE-SQL performance on SParC dev set influenced by different Python-code styles of prompting text for chain-of-editions. Different python-code styles indicate that the three components code, dict, and SQL are put in the prompting text in different orders. The "*" mark means that we use this specific component to complete the post-processing procedure.

When using the natural language style, we add "no change is needed" in the prompt if the clause is not edited as shown in Table 2. The experiment result in Table 10 proves that this prompting text is necessary. Through adding this special sentence, the CoE prompting text in the exemplars can be regular and normalized, since all the components and clauses can be mentioned in the context. LLMs are generally better at receiving and handling regular and normalized contexts.

| Natural Language Style | TS(QM) ↑ | TS(IM) ↑ |
|---|---|---|
| Natural Language | **63.3** | **41.9** |
| w/o "no change is needed" | 62.8 | 41.0 |

Table 10: CoE-SQL performance on SParC dev set influenced by the "no change is needed" prompting text.

**Analysis of differences between pre- and post-Questions** Table 11 proves that the question analysis mentioned in Section 3.5 is effective in our approach. The analysis of differences between the current question and the previous one is beneficial for LLMs to think more about the possible edit rules. Without the analysis, the TS(QM) score and the TS(IM) score both drop about 2%.

| Method | TS(QM) ↑ | TS(IM) ↑ |
|---|---|---|
| CoE-SQL | **63.3** | **41.9** |
| w/o analysis | 61.3 | 39.8 |

Table 11: CoE-SQL performance on SParC dev set influenced by the question analysis.

**Coverage of edition chain** As mentioned in Section 3.3, if an edition chain is too long, LLMs would generate the SQL query directly instead of using the edit-based method. We can control the coverage of the edition chain on the training dataset by changing the maximum length of the edition chain $l_c$. Figure 2 shows the performances on SParC dev set influenced by $l_c$. According to the experiment result, we set $l_c$ to 4 in our main experiment. If $l_c$ is too small, LLMs would be more likely to directly generate the SQL query without using the edit-based method. If $l_c$ is too large, LLMs would always edit the SQL query although the CoE-SQL approach may not be suitable for the current testing case.

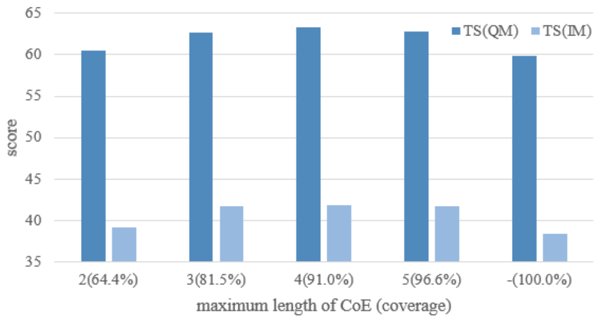

Figure 2: CoE-SQL performance on SParC dev set influenced by the maximum length of the edition chain.

We also try another way to control the chain-of-edition coverage in the training dataset. We remove the relatively complex unit edit rules and only retain the commonly used ones. Therefore, LLMs would not be forced to edit the SQL query if the SQL query contains complex clauses. Table 12 shows the experiment results with the different definitions of unit edit rules. In the first row, we only keep the unit edit rules involving the SELECT clause and the WHERE clause. In the second row, we add the unit edit rules that are relative to the FROM clause. In the third row, we do not set any limitations. The experiment result indicates that this method is not that effective. With the change of the unit edit rules, the LLMs' performance does not change a lot. This explains why we use the maximum length of the edition chain to limit the edit-based approach.

| Unit Edit Rules (Coverage) | TS(QM) ↑ | TS(IM) ↑ |
|---|---|---|
| S + W (48.1%) | 60.0 | 38.9 |
| F + S + W (65.9%) | 60.3 | 38.2 |
| - (100.0%) | 59.9 | 38.4 |

Table 12: CoE-SQL performance on SParC dev set influenced by the definition of unit edit rules. F represents the unit edit rules about the FROM clause. S represents the unit edit rules about the SELECT clause. W represents the unit edit rules about the WHERE clause.

### 4.4 Discussion

We discuss the effectiveness of our CoE-SQL method under the SParC dev set. There are totally 422 interactions and 1,203 SQL queries in the SParC dev set. Among all 1,203 SQL results written by LLMs, 484 SQL queries are generated directly without using edit rules. Moreover, the first SQL in each interaction is always generated directly. Ignoring the first SQL in each interaction, the probability of generating SQL directly is merely $\frac{484-422}{1203-422} \times 100\% = 7.9\%$. This indicates that LLMs effectively utilize our CoE-based approach to handle most complex testing cases. The CoE-based approach takes a significant role in the prediction process.

## 5 Conclusion

We propose our CoE-SQL approach for the multi-turn text-to-SQL task based on editing the previous SQL query to the current SQL query. We explore the definition and the style of our unit edit rules. We also provide the method to extract the edition chain by comparing two ASTs of two different SQL queries. Furthermore, our proposed CoE framework follows the human thinking process. The experiment results demonstrate that our approach achieves the best performances on the SParC and CoSQL dev set among existing in-context learning approaches and is also comparable to the SOTA fine-tuned model. We complete some ablation studies and prove the effectiveness of various components in CoE-SQL.

## Limitations

There are some limitations in our work. First, we mainly concentrate on investigating the effectiveness of the edition chain. We do not explore anything about the exemplar selection strategy which can influence LLMs' performances a lot. Second, we cannot ensure that we have thoroughly optimized the CoE-SQL approach. There still may exist some methods for the optimization. Third, our approach does not surpass the previous SOTA fine-tuned model on the SParC and CoSQL dev sets under some evaluation metrics. These are all difficult tasks that need to be addressed in the future work.

## Acknowledgements

This work is funded by the China NSFC Projects (62106142, U23B2057 and 62120106006) and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

## References

Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases-an introduction. *arXiv preprint cmp-lg/9503016*.

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.

Yitao Cai and Xiaojun Wan. 2020. IGSQL: Database schema interaction graph based neural model for context-dependent text-to-SQL generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6903–6912, Online. Association for Computational Linguistics.

Zefeng Cai, Xiangyu Li, Binyuan Hui, Min Yang, Bowen Li, Binhua Li, Zheng Cao, Weijie Li, Fei Huang, Luo Si, et al. 2022. Star: Sql guided pre-training for context-dependent text-to-sql parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1235–1247.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021a. Evaluating large language models trained on code.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023a. Teaching large language models to self-debug.

Zhi Chen, Lu Chen, Hanqi Li, Ruisheng Cao, Da Ma, Mengyue Wu, and Kai Yu. 2021b. Decoupled dialogue modeling and semantic parsing for multi-turn text-to-sql. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3063–3074.

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 61th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Toronto, Canada. Association for Computational Linguistics.

Binyuan Hui, Ruiying Geng, Qiyu Ren, Binhua Li, Yongbin Li, Jian Sun, Fei Huang, Luo Si, Pengfei Zhu, and Xiaodan Zhu. 2021. Dynamic hybrid relation network for cross-domain context-dependent semantic parsing.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, 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.

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

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Run-Ze Wang, Zhen-Hua Ling, Jing-Bo Zhou, and Yu Hu. 2020. Tracking interaction states for multi-turn text-to-sql semantic parsing.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Dongling Xiao, LinZheng Chai, Qian-Wen Zhang, Zhao Yan, Zhoujun Li, and Yunbo Cao. 2022. Cqr-sql: Conversational question reformulation enhanced context-dependent text-to-sql parsers. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2055–2068.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. 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 1962–1979, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. 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 5338–5349, Hong Kong, China. Association for Computational Linguistics.

Yanzhao Zheng, Haibin Wang, Baohua Dong, Xingjun Wang, and Changshan Li. 2022. HIE-SQL: History information enhanced network for context-dependent text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2997–3007, Dublin, Ireland. Association for Computational Linguistics.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

# A Unit Edit Rules

| Unit Edit Rule | Explanation |
|---|---|
| EditSelectItem(oldItem, newItem) | Replace oldItem with newItem in the SELECT clause.<br>Add newItem into the SELECT clause if oldItem is “-”.<br>Delete oldItem from the SELECT clause if newItem is “-”. |
| EditFromTable(oldTable, newTable) | Replace oldTable with newTable in the FROM clause.<br>Add newTable into the FROM clause if oldTable is “-”.<br>Delete oldTable from the FROM clause if newTable is “-”. |
| EditNestedFromClause(SQL) | Edit the nested FROM clause with SQL.<br>Delete the nested FROM clause if SQL is “-”. |
| EditJoinCondition(oldCondition, newCondition) | Replace oldCondition with newCondition in the ON clause.<br>Add newCondition into the ON clause if oldCondition is “-”.<br>Delete oldCondition from the ON clause if newCondition is “-”. |
| EditJoinLogicalOperator(and/or) | Edit the logical operator in the ON clause. |
| EditWhereCondition(oldCondition, newCondition) | Replace oldCondition with newCondition in the WHERE clause.<br>Add newCondition into the WHERE clause if oldCondition is “-”.<br>Delete oldCondition from the WHERE clause if newCondition is “-”. |
| EditWhereLogicalOperator(and/or) | Edit the logical operator in the WHERE clause. |
| EditGroupByColumn(oldColumn, newColumn) | Replace oldColumn with newColumn in the GROUP BY clause.<br>Add newColumn into the GROUP BY clause if oldColumn is “-”.<br>Delete oldColumn from the GROUP BY clause if newColumn is “-”. |
| EditHavingCondition(oldCondition, newCondition) | Replace oldCondition with newCondition in the HAVING clause.<br>Add newCondition into the HAVING clause if oldCondition is “-”.<br>Delete oldCondition from the HAVING clause if newCondition is “-”. |
| EditHavingLogicalOperator(and/or) | Edit the logical operator in the HAVING clause. |
| EditOrderByItem(oldItem, newItem) | Replace oldItem with newItem in the ORDER BY clause.<br>Add newItem into the ORDER BY clause if oldItem is “-”.<br>Delete oldItem from the ORDER BY clause if newItem is “-”. |
| EditOrder(asc/desc) | Edit the order in the ORDER BY clause. |
| EditLimit(oldLimit, newLimit) | Replace oldLimit with newLimit in the LIMIT clause.<br>Add newLimit into the LIMIT clause if oldLimit is “-”.<br>Delete oldLimit from the LIMIT clause if newLimit is “-”. |
| EditIUE(intersect/union/except, left/right, SQL) | Append SQL to the left/right side of the previous SQL with intersect/union/except keyword.<br>Delete the left/right side of the previous SQL with intersect/union/except keyword if SQL is “-”. |

Table 13: All 14 defined unit edit rules.

# B Detailed Experiment Results

Table 14 and Table 15 show the detailed performances of ACT-SQL (Zhang et al., 2023), our baseline, and our CoE-SQL on the SParC and CoSQL dev sets respectively. All the results in these two tables are TS scores. The results prove that the CoE-SQL approach performs better than the previous methods with different difficulty levels and numbers of conversational turns.

| Approach | TS(QM) | TS(IM) | Easy | Medium | Hard | Extra | Turn 1 | Turn 2 | Turn 3 | Turn 4 | Turn > 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ACT-SQL (Zhang et al., 2023) | 56.9 | 29.6 | 77.2 | 51.9 | 40.0 | 17.9 | 72.5 | 55.7 | 41.5 | 34.1 | 100.0 |
| Baseline (Ours) | 59.5 | 37.9 | 77.6 | 58.5 | 42.8 | 15.7 | 69.2 | 59.5 | 49.6 | 43.2 | 100.0 |
| **CoE-SQL (Ours)** | 63.3 | 41.9 | 81.8 | 59.9 | 46.2 | 26.9 | 76.1 | 60.0 | 53.3 | 48.9 | 100.0 |

Table 14: Detailed performances on the SParC dev set.

| Approach | TS(QM) | TS(IM) | Easy | Medium | Hard | Extra | Turn 1 | Turn 2 | Turn 3 | Turn 4 | Turn > 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ACT-SQL (Zhang et al., 2023) | 55.2 | 21.5 | 76.5 | 46.6 | 42.3 | 17.8 | 65.2 | 54.7 | 50.8 | 48.2 | 42.3 |
| Baseline (Ours) | 58.5 | 27.6 | 79.9 | 48.7 | 46.6 | 22.4 | 66.6 | 58.2 | 55.3 | 56.1 | 40.8 |
| **CoE-SQL (Ours)** | 60.6 | 30.4 | 79.1 | 54.7 | 50.3 | 21.5 | 67.2 | 61.8 | 57.0 | 56.1 | 47.9 |

Table 15: Detailed performances on the CoSQL dev set.

Table 16 shows two testing cases from the SParC dev set. The complete LLMs' responses are shown in the table. In the first case, CoE-SQL successfully retains the WHERE condition while the trivial baseline method forgets this key information during the conversation. The second case demonstrates that the CoE-SQL approach can also assist LLMs to handle complex logical reasoning processes such as the JOIN operation for multi-tables and the GROUP BY operation in this case.

**Turn 1**

**Question:** *Find all employees who are under age 30.*

**Baseline:** SELECT * FROM employee WHERE age < 30 ✓

**CoE-SQL:**

Let's think step by step.
SQL 17-1 can be written directly instead of being edited from previous SQL.
So SQL 17-1 is:
SELECT * FROM employee WHERE age < 30 ✓

**Turn 2**

**Question:** *Which cities did they come from?*

**Baseline:** SELECT DISTINCT city FROM employee WHERE age < 30 ✓

**CoE-SQL:**

Let's think step by step.
SQL 17-2 can be edited from SQL 17-1.
The previous question asked for all employees who are under the age of 30, while the current question asks for the cities they come from.
Therefore, following edit operations are used:
FROM clause:
- no change is needed
SELECT clause:
- change * to employee.City
WHERE clause:
- no change is needed
GROUP BY clause:
- no change is needed
ORDER BY clause:
- no change is needed
LIMIT clause:
- no change is needed
INTERSECT/UNION/EXCEPT:
- no change is needed
So SQL 17-2 is:
SELECT City FROM employee WHERE age < 30 ✓

**Turn 3**

**Question:** *Show the cities from which more than one employee originated.*

**Baseline:** SELECT city FROM employee GROUP BY city HAVING count(*) > 1 ✖

**CoE-SQL:**

Let's think step by step.
SQL 17-3 can be edited from SQL 17-2.
The previous question asked for the cities from which the employees originated, while the current question asks for the cities from which more than one employee originated.
Therefore, following edit operations are used:
FROM clause:
- no change is needed
SELECT clause:
- no change is needed
WHERE clause:
- no change is needed
GROUP BY clause:
- add HAVING condition COUNT(*) > 1

ORDER BY clause:
- no change is needed

LIMIT clause:
- no change is needed

INTERSECT/UNION/EXCEPT:
- no change is needed

So SQL 17-3 is:
SELECT City FROM employee WHERE age < 30 GROUP BY City HAVING COUNT(*) > 1 ✓

---

**Turn 1**
**Question:** *What are all the airlines?*
**Baseline:** SELECT Airline FROM airlines ✓
**CoE-SQL:**
Let's think step by step.
SQL 17-1 can be written directly instead of being edited from previous SQL.
So SQL 17-1 is:
SELECT Airline FROM airlines ✓

- - -

**Turn 2**
**Question:** *Which ones have more than 10 flights?*
**Baseline:** SELECT Airline FROM flights GROUP BY Airline HAVING count(*) > 10 ✖
**CoE-SQL**:
Let's think step by step.
SQL 17-2 can be edited from SQL 17-1.
The previous question asked for all the airlines, while the current question asks for the airlines that have more than 10 flights.
Therefore, following edit operations are used:

FROM clause:
- add table flights
- add JOIN condition airlines.uid = flights.Airline

SELECT clause:
- no change is needed

WHERE clause:
- add WHERE condition COUNT(*) > 10

GROUP BY clause:
- add column airlines.Airline

ORDER BY clause:
- no change is needed

LIMIT clause:
- no change is needed

INTERSECT/UNION/EXCEPT:
- no change is needed

So SQL 17-2 is:
SELECT airlines.Airline FROM airlines JOIN flights ON airlines.uid = flights.Airline GROUP BY airlines.Airline HAVING COUNT(*) > 10 ✓

Table 16: Detailed case studies from the SParC dev set.

## C Experiment Results with Open-Source LLM

We attempt to complete the experiment using the open-source LLM on the SParC dev set. Table 17 shows the result. We use the Vicuna LLM in our experiment. We find that open-source LLM's few-shot learning abilities are far inferior than GPT-series LLMs.

| Approach | QM | | | IM | | |
|---|---|---|---|---|---|---|
| | **EM ↑** | **EX ↑** | **TS ↑** | **EM ↑** | **EX ↑** | **TS ↑** |
| CoE-SQL + lmsys/vicuna-7b-v1.5-16k | 23.7 | 36.0 | 29.9 | 11.4 | 19.4 | 14.5 |
| CoE-SQL + lmsys/vicuna-13b-v1.5-16k | 21.1 | 35.5 | 30.4 | 6.4 | 14.7 | 11.1 |

Table 17: Performances of CoE-SQL with open-source LLM on SParC dev set.

## D Prompts

In this section, we show our prompt engineering with an example from the SParC train set.

### D.1 Baseline

```
role: system
content:
Given the database schema, you need to translate the question into the SQL query.

role: user
content:
Database schema:
create table party (
        Party_ID number,
        Party_Theme text,
        Location text,
        First_year text,
        Last_year text,
        Number_of_hosts number,
        primary key (Party_ID)
)
/*
3 example rows from table party:
Party_ID        Party_Theme        Location        First_year        Last_year        Number_of_hosts
1        Spring        Hemkade 48 Zaandam        2000        2001        5
2        Club Q-BASE        Hemkade 48 Zaandam        2002        2002        23
3        Hardcore Resurrection        Various locations        2000        2003        4
**/
create table host (
        Host_ID number,
        Name text,
        Nationality text,
        Age text,
        primary key (Host_ID)
)
/*
3 example rows from table host:
Host_ID        Name        Nationality        Age
1        Austin Daye        United States        43
2        Lloyd Daniels        United States        37
3        Kornel David        Hungary        31
**/
create table party_host (
        Party_ID number,
        Host_ID number,
        Is_Main_in_Charge others,
        primary key (Party_ID),
        foreign key (Party_ID) references party(Party_ID),
        foreign key (Host_ID) references host(Host_ID)
)
/*
3 example rows from table party_host:
Party_ID        Host_ID        Is_Main_in_Charge
1        1        T
8        7        T
6        10        F
**/
```

Question: What are all the parties?

role: assistant
content:
SELECT * FROM party

role: user
content:
Question: Order them by the number of hosts.

role: assistant
content:
SELECT * FROM party ORDER BY Number_of_hosts ASC

### D.2 Self-Defined Edit Rule Style

role: system
content:
Given the database schema, you need to translate the question into the SQL query.
You can use following operations to edit SQL:
1. EditIUE(intersect/union/except, left/right, SQL): Append SQL to the left/right side of the previous SQL with intersect/union/except keyword. Delete the left/right side of the previous SQL with intersect/union/except keyword if SQL is "-".
2. EditFromTable(oldTable, newTable): Replace oldTable with newTable in the FROM clause. Add newTable into the FROM clause if oldTable is "-". Delete oldTable from the FROM clause if newTable is "-".
3. EditJoinCondition(oldCondition, newCondition): Replace oldCondition with newCondition in the ON clause. Add newCondition into the ON clause if oldCondition is "-". Delete oldCondition from the ON clause if newCondition is "-".
4. EditJoinLogicalOperator(and/or): Edit the logical operator in the ON clause.
5. EditNestedFromClause(SQL): Edit the nested FROM clause with SQL. Delete the nested FROM clause if SQL is "-".
6. EditSelectItem(oldItem, newItem): Replace oldItem with newItem in the SELECT clause. Add newItem into the SELECT clause if oldItem is "-". Delete oldItem from the SELECT clause if newItem is "-".
7. EditWhereCondition(oldCondition, newCondition): Replace oldCondition with newCondition in the WHERE clause. Add newCondition into the WHERE clause if oldCondition is "-". Delete oldCondition from the WHERE clause if newCondition is "-".
8. EditWhereLogicalOperator(and/or): Edit the logical operator in the WHERE clause.
9. EditGroupByColumn(oldColumn, newColumn): Replace oldColumn with newColumn in the GROUP BY clause. Add newColumn into the GROUP BY clause if oldColumn is "-". Delete oldColumn from the

GROUP BY clause if newColumn is "-".
10. EditHavingCondition(oldCondition, newCondition): Replace oldCondition with newCondition in the HAVING clause. Add newCondition into the HAVING clause if oldCondition is "-". Delete oldCondition from the HAVING clause if newCondition is "-".
11. EditHavingLogicalOperator(and/or): Edit the logical operator in the HAVING clause.
12. EditOrderByItem(oldItem, newItem): Replace oldItem with newItem in the ORDER BY clause. Add newItem into the ORDER BY clause if oldItem is "-". Delete oldItem from the ORDER BY clause if newItem is "-".
13. EditOrder(asc/desc): Edit the order in the ORDER BY clause.
14. EditLimit(oldLimit, newLimit): Replace oldLimit with newLimit in the LIMIT clause. Add newLimit into the LIMIT clause if oldLimit is "-". Delete oldLimit from the LIMIT clause if newLimit is "-".

role: user
content:
Database schema:

```
create table party (
    Party_ID number,
    Party_Theme text,
    Location text,
    First_year text,
    Last_year text,
    Number_of_hosts number,
    primary key (Party_ID)
)
/*
3 example rows from table party:
Party_ID    Party_Theme    Location    First_year    Last_year    Number_of_hosts
1    Spring    Hemkade 48 Zaandam    2000    2001    5
2    Club Q-BASE    Hemkade 48 Zaandam    2002    2002    23
3    Hardcore Resurrection    Various locations    2000    2003    4
**/
create table host (
    Host_ID number,
    Name text,
    Nationality text,
    Age text,
    primary key (Host_ID)
)
/*
3 example rows from table host:
Host_ID    Name    Nationality    Age
1    Austin Daye    United States    43
2    Lloyd Daniels    United States    37
3    Kornel David    Hungary    31
**/
create table party_host (
    Party_ID number,
    Host_ID number,
    Is_Main_in_Charge others,
    primary key (Party_ID),
    foreign key (Party_ID) references party(Party_ID),
    foreign key (Host_ID) references host(Host_ID)
```

)
/*
3 example rows from table party_host:

| Party_ID | Host_ID | Is_Main_in_Charge |
|---|---|---|
| 1 | 1 | T |
| 8 | 7 | T |
| 6 | 10 | F |

**/
Question 1-1: What are all the parties?

role: assistant:
content:
Let’s think step by step.
SQL 1-1 can be written directly instead of being edited from previous SQL.
So SQL 1-1 is:
SELECT * FROM party

role: user:
content:
Question 1-2: Order them by the number of hosts.

role: assistant
content:
Let’s think step by step.
SQL 1-2 can be edited from SQL 1-1.
The previous question asked for a list of all the parties, while the current question asks for the parties to be ordered by the number of hosts they have.
Therefore, following edit operations are used:
FROM clause:
- no change is needed
SELECT clause:
- no change is needed
WHERE clause:
- no change is needed
GROUP BY clause:
- no change is needed
ORDER BY clause:
- EditOrderByItem(-, party.Number_of_hosts)
- EditOrder(asc)
LIMIT clause:
- no change is needed
INTERSECT/UNION/EXCEPT:
- no change is needed
So SQL 1-2 is:
SELECT * FROM party ORDER BY Number_of_hosts ASC

### D.3 Python Code Style

role: system
content:
Given the database schema, you need to translate the question into the SQL query.

role: user

content:

```
Database schema:
create table party (
    Party_ID number,
    Party_Theme text,
    Location text,
    First_year text,
    Last_year text,
    Number_of_hosts number,
    primary key (Party_ID)
)
/*
3 example rows from table party:
Party_ID    Party_Theme    Location    First_year    Last_year    Number_of_hosts
1    Spring    Hemkade 48 Zaandam    2000    2001    5
2    Club Q-BASE    Hemkade 48 Zaandam    2002    2002    23
3    Hardcore Resurrection    Various locations    2000    2003    4
**/
create table host (
    Host_ID number,
    Name text,
    Nationality text,
    Age text,
    primary key (Host_ID)
)
/*
3 example rows from table host:
Host_ID    Name    Nationality    Age
1    Austin Daye    United States    43
2    Lloyd Daniels    United States    37
3    Kornel David    Hungary    31
**/
create table party_host (
    Party_ID number,
    Host_ID number,
    Is_Main_in_Charge others,
    primary key (Party_ID),
    foreign key (Party_ID) references party(Party_ID),
    foreign key (Host_ID) references host(Host_ID)
)
/*
3 example rows from table party_host:
Party_ID    Host_ID    Is_Main_in_Charge
1    1    T
8    7    T
6    10    F
**/
```

Question 1-1: What are all the parties?

role: assistant:
content:
Let's think step by step.

SQL 1-1 can be written directly instead of being edited from previous SQL.
So SQL 1-1 is:
SELECT * FROM party
So SQL dict 1-1 is:

```
{
    "from": {
        "tables": [
            "party"
        ]
    },
    "select": [
        "*"
    ]
}
```

role: user:
content:
Question 1-2: Order them by the number of hosts.

role: assistant
content:
Let’s think step by step.
SQL 1-2 can be edited from SQL 1-1.
The previous question asked for a list of all the parties, while the current question asks for the parties to be ordered by the number of hosts they have.
Therefore, following edit operations are used:
sql['order_by']['columns'].append('party.Number_of_hosts')
sql['order_by']['order'] = 'ASC'
So SQL 1-2 is:
SELECT * FROM party ORDER BY Number_of_hosts ASC
So SQL dict 1-2 is:

```
{
    "from": {
        "tables": [
            "party"
        ]
    },
    "select": [
        "*"
    ],
    "order_by": {
        "columns": [
            "party.Number_of_hosts"
        ],
        "order": "ASC"
    }
}
```

### D.4 Natural Language Style

role: system
content:
Given the database schema, you need to translate the question into the SQL query.

role: user
content:
Database schema:

```
create table party (
        Party_ID number,
        Party_Theme text,
        Location text,
        First_year text,
        Last_year text,
        Number_of_hosts number,
        primary key (Party_ID)
)
/*
3 example rows from table party:
Party_ID        Party_Theme        Location        First_year        Last_year        Number_of_hosts
1        Spring        Hemkade 48 Zaandam        2000        2001        5
2        Club Q-BASE        Hemkade 48 Zaandam        2002        2002        23
3        Hardcore Resurrection        Various locations        2000        2003        4
**/
create table host (
        Host_ID number,
        Name text,
        Nationality text,
        Age text,
        primary key (Host_ID)
)
/*
3 example rows from table host:
Host_ID        Name        Nationality        Age
1        Austin Daye        United States        43
2        Lloyd Daniels        United States        37
3        Kornel David        Hungary        31
**/
create table party_host (
        Party_ID number,
        Host_ID number,
        Is_Main_in_Charge others,
        primary key (Party_ID),
        foreign key (Party_ID) references party(Party_ID),
        foreign key (Host_ID) references host(Host_ID)
)
/*
3 example rows from table party_host:
Party_ID        Host_ID        Is_Main_in_Charge
1        1        T
8        7        T
6        10        F
**/
```

Question 1-1: What are all the parties?

role: assistant:

content:
Let’s think step by step.
SQL 1-1 can be written directly instead of being edited from previous SQL.
So SQL 1-1 is:
SELECT * FROM party

role: user:
content:
Question 1-2: Order them by the number of hosts.

role: assistant
content:
Let’s think step by step.
SQL 1-2 can be edited from SQL 1-1.
The previous question asked for a list of all the parties, while the current question asks for the parties to be ordered by the number of hosts they have.
Therefore, following edit operations are used:
FROM clause:
- no change is needed
SELECT clause:
- no change is needed
WHERE clause:
- no change is needed
GROUP BY clause:
- no change is needed
ORDER BY clause:
- add party.Number_of_hosts
- change order to ASC
LIMIT clause:
- no change is needed
INTERSECT/UNION/EXCEPT:
- no change is needed
So SQL 1-2 is:
SELECT * FROM party ORDER BY Number_of_hosts ASC