CREDIT-SQL: Few-shot prompting for context-dependent text-to-SQL with regularized examples from diversity sampling

Anonymous ACL submission

Abstract

In this paper, we propose a few-shot prompting method called CREDIT-SQL for the context-dependent text-to-SQL problem. CREDIT-SQL converts each question in a multi-turn dialogue into a self-contained question with a fixed few-shot prompt. Once a self-contained question is obtained, CREDIT-SQL converts it into an SQL query using a prompt made of in-context examples selected by diversity sampling and subsequent example voting. After experimentations with multiple LLMs, CREDIT-SQL achieves 58.6% in terms of the exact set match without values on the dev set of CoSQL, which is the performance comparable to the state-of-the-art models for context-dependent text-to-SQL. We also argue that the example voting we introduced in CREDIT-SQL can serve as an efficient and effective way to mitigate the instability of in-context example selection in general.

1 Introduction

Information retrieval from structured knowledge sources is an NLP task widely applicable in many areas. Text-to-SQL is a promising approach to achieve this goal due to the popularity of SQL as an interface between the user and the database. Text-to-SQL systems have shown remarkable improvements (Wang et al., 2020a,b; Lin et al., 2020; Cao et al., 2021; Scholak et al., 2021; Cai and Wan, 2020) along with the rapid advancements of sequence-to-sequence models including the infamous transformer model (Vaswani et al., 2017). The advantages of these advanced sequence-to-sequence models have been often utilized by fine-tuning pre-trained decoder-encoder models. However, these advanced models become so large that they are called large language models (LLMs) which typically have parameter size ranges from a few tens of billion to a few hundreds of billion (Ye et al., 2023; OpenAI, 2023; Touvron et al., 2023a,b; Anil et al., 2023). Because of this large model size, it takes too much resource to fine-tune these large language models on custom datasets.

To utilize the advantages of advanced sequence-to-sequence models without investing full resources for fine-tuning, in-context learning with zero-shot prompts or few-shot prompts has become popular recently. In the case of few-shot in-context learning, a few in-context examples are listed in the prompt along with a brief instruction, and the LLM outputs the desired sequence as the response to the input prompt. Although the limited context size of available LLMs only allows a handful of in-context examples to be included in each prompt, it has been shown that strategic designs of prompts can perform as well as fine-tuned models in the tasks of text-to-SQL (Pourreza and Rafiei, 2023; Nan et al., 2023; Dong et al., 2023; Gao et al., 2023).

Still, most in-context learning studies on text-to-SQL tasks focus on the context-independent setting where the system needs to answer a single SQL query on the input of a single question. This context-independent setting becomes particularly inconvenient when one needs to develop conversational information retrieval systems where previous questions or answers can implicitly appear in the user's later questions.

To address this problem, we propose a few-shot prompting method called CREDIT-SQL[1] in this paper (See Figure 1, 2, and 3). CREDIT-SQL does context-dependent question rephrasing to convert a multi-turn text-to-SQL task on each dialogue into a series of question-query pair text-to-SQL tasks. Once all the questions are rephrased, a diversity-sampled prompt is used to address the text-to-SQL tasks. This prompt is composed of examples obtained via multiple trials of diversity sampling and subsequent example voting. Each example in the prompt is represented as a pair of the rephrased question and the regularized SQL query along with

[1] Context-dependent Regularized Examples from DIversity sampling for Text-to-SQL.

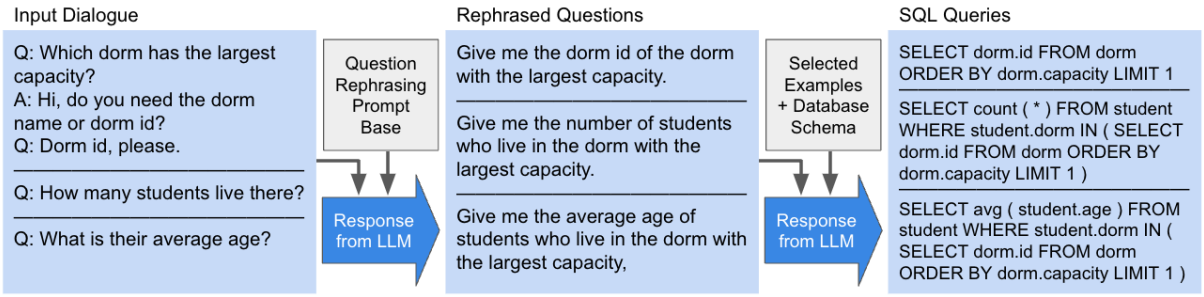

Figure 1: Illustration of overall question to SQL process of CREDIT-SQL. The details of the question rephrasing process are illustrated in Figure 2, and the details of the example selection process are illustrated in Figure 3.

the database schema.

We claim that our approach makes the following contributions. (1) To the best of our knowledge, we propose the first few-shot prompting approach to perform the dialogue state tracking task with systematically selected in-context examples out of the entire training data. (2) We experimented our approach with multiple open and closed-sourced LLMs, and we report the performance comparable to the state-of-the-art models in the dialogue state tracking task on the CoSQL dataset, which is ranked 1st in the execution accuracy without values and 4th in the exact set match without values on the CoSQL dev set among the models reported on the CoSQL leaderboard at the moment of writing. (3) We suggest a new method to mitigate the instability of in-context example selection in the few-shot prompting with LLM by introducing voting on collected examples.

2 Related works

2.1 Context-dependent text-to-SQL

Most studies on text-to-SQL tasks focused on context-independent settings where a single question is transcribed into a single SQL query. To cope with the complicated scenarios where multiple tables are involved, utilizing graph structures to capture the relations between entities has been the most popular and successful method in text-to-SQL tasks recently (Bogin et al., 2019; Wang et al., 2020a,b; Lin et al., 2020; Cao et al., 2021; Scholak et al., 2021; Cai et al., 2021; Hui et al., 2022).

Unlike its context-independent counterpart, context-dependent text-to-SQL tasks require encoding the context within the dialogue and exploiting this context in the SQL generation. Recently, numerous different approaches have been suggested to tackle this problem. Zhang et al. 2019 used turn attention to edit the SQL query of the previous turn to accommodate the question at the current turn. Cai and Wan 2020 extends the graph structure for the database schema to establish connections between neighboring turns in the dialogue. Wang et al. 2021 and Hui et al. 2021 suggested using a graph structure state tracker to capture the context of the dialogue at each turn, while Zheng et al. 2022 used BERT to encode the history of the dialogue. Pan et al. 2019, Chen et al. 2021, and Chai et al. 2023 rephrased the question at each turn reflecting the context of the dialogue. Xiao et al. 2022 applied question rephrasing recursively and introduced consistency training to build one of the state-of-the-art models at the time of writing. Other state-of-the-art models used utterance dependency tracking with weighted contrastive learning (Cai et al., 2022) or integrating relational structure through the attention layer into the pre-trained models (Qi et al., 2022).

2.2 Prompting with large language models for text-to-SQL

As in its non-prompting counterpart, most of the efforts to perform text-to-SQL tasks focused on context-independent text-to-SQL tasks. Pourreza and Rafiei 2023 used different in-context examples for each difficulty of the question. Dong et al. 2023 achieved one of the best performance among zero-shot prompting efforts. Beyond the arbitrary selection of in-context examples, there have been trials to choose in-context examples in a systematic manner. Liu et al. 2022 introduced question-similarity based example selection using k-NN algorithm. Nan et al. 2023 noticed that the diversity of the in-context examples in a prompt is indeed important, and suggested methods to balance the similarity and the diversity of in-context examples based on their ground-truth SQL queries. Gao et al. 2023 achieved the best performance on the leader-

board of Spider (Yu et al., 2018) at the time of writing, with systematic in-context example selection using both questions and SQL queries. Beyond the context-independent setting, Hu et al. 2022 addressed context-dependent text-to-SQL tasks with few-shot prompting and zero-shot prompting, but systematic in-context example selection was still lacking.

3 Methods

3.1 Context-dependent question rephrasing

Inspired by the success of question rephrasing approaches (Pan et al., 2019; Chen et al., 2021; Chai et al., 2023; Xiao et al., 2022), we summarize the context of each dialogue into a single question with few-shot prompting at every turn. For the few-shot prompts for this question rephrasing step, we randomly sampled 20 examples out of train set of CoSQL dataset (Yu et al., 2019a), and the fixed prompt base we used for question rephrasing is in Appendix A. In each multi-turn interaction, rephrased questions from previous turns are appended in the prompt to rephrase questions in further turns. The entire process of question rephrasing is illustrated in Figure 2.

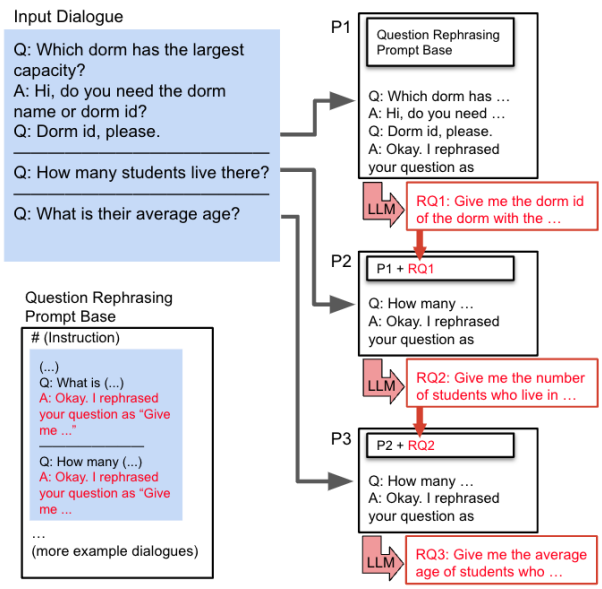

Figure 2: Context-dependent question rephrasing process. A prompt containing examples of multiple dialogues is used to rephrase each question into a rephrased question starting with "Give me ...". Each rephrased question is appended to the prompt for further question rephrasing of later turns.

This question rephrasing process converts the context-dependent text-to-SQL task into the simple text-to-SQL task which is better studied in the literature than the context-dependent counterpart. This process also regularizes the questions in a similar format ("Give me ...") which can help to create consistent in-context examples for the text-to-SQL tasks. This helps LLM to focus more on transcribing relevant natural language expressions into SQL expressions rather than on deciphering the meaning of the questions written in different styles.

3.2 SQL query regularization

To increase the consistency of the in-context examples for the text-to-SQL tasks, we regularize the SQL queries used in text-to-SQL prompts with rule-based methods. This regularization includes capitalization, spacing, unaliasing, table representation in each column reference, and so on. Examples of affected SQL queries through this regularization are shown in Table 1. For further details, we attach the pseudocode implementation of the SQL regularization in Appendix F.

Given SQL	Regularized SQL
select * from tb_1;	SELECT * FROM tb_1
SELECT T1.C1 FROM tb_1 as T1 JOIN tb_2 as T2 on T1.C3=T2.C4	SELECT tb_1.c1 FROM tb_1 JOIN tb_2 ON tb_1.c3 = tb_2.c4
select COUNT(*) from tb_1 where c2=="A"	SELECT count (*) FROM tb_1 WHERE tb_1.c2 == 'A'

Table 1: Examples of affected SQL query expressions through the rule-based SQL query regularization.

Similar to the question rephrasing, SQL query regularization helps LLM to focus more on the grammatical structure of SQL queries or links to the database schema rather than on different expression styles of SQL queries.

3.3 Example selection for text-to-SQL

In the few-shot prompting with LLM, the performance of the model is very sensitive to the choice of in-context examples. In particular, strategic sampling of examples out of the training data significantly outperforms the random choices of examples. A natural way to customize in-context examples for each question is to collect the closest examples to the given question, often based on the similarity in the embedding vector space (Liu et al., 2022). In the meanwhile, Nan et al. 2023 pointed

out that keeping the diversity of the example pool can be more important than a mere collection of similar examples. As introduced in Nan et al. 2023, we adopt diversity sampling through the k-means clustering based on the vectorized SQL queries. Specifically, we vectorized all SQL queries in the CoSQL train set as presented in the pseudocode in Appendix G. Then we perform the clustering with $k = N$ for the N-example prompt based on these vectors. In each cluster, we choose the example closest to the centroid in the SQL vector space. In case the number of resulting clusters N_c is smaller than N, we fill the rest of the examples based on the distance with previous examples. To be specific, among training data examples $\{e_1, e_2, \cdots, e_T\}$ with SQL vectors $\{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_T}\}$, nth selected example s_n is

$$
\begin{aligned}
s_n &= e_{i_n}, \\
i_n &= \underset{i}{\operatorname{argmin}} \left|\mathbf{x_i} - \mathbf{c_n}\right| \text{ for } n \leq N_c, \\
i_n &= \underset{i \notin \{i_1, \cdots, i_{n-1}\}}{\operatorname{argmax}} \left(\min_{j<n} \left|\mathbf{x_i} - \mathbf{x_{i_j}}\right| \right) \\
&\quad \text{for } N_c < n \leq N,
\end{aligned}
$$

where $\{\mathbf{c_1}, \mathbf{c_2}, \cdots \mathbf{c_{N_c}}\}$ are the SQL vectors of cluster centroids.

Since the k-means clustering is a non-deterministic algorithm that depends on the random initial positions of centroids, the selection result of examples varies depending on the choice of random seed. To mitigate this inconsistency of example selection, we adopt example voting. First, we collect N-example prompt through the k-means clustering. We repeat this example collection M times, with a different random seed each time. Then we rank each example by its occurrence among these M different sets of examples. The entire example selection process is illustrated in Figure 3. With some parameter search (see Figure 4 and Figure 5 for the search space), we obtained the best result with $N = 18$ and $M = 20$. For the rest of the paper, CREDIT-SQL used in-context examples selected by the voting process with $N = 18$ and $M = 20$ otherwise noted. We show these selected in-context examples in Appendix B.

3.4 Example demonstration

In our few-shot prompt for text-to-SQL, we demonstrate selected in-context examples along with the database ID and the database schema including table names, column names, and foreign keys. A sample text-to-SQL prompt for the CREDIT-SQL approach including the example demonstration is in Appendix C.

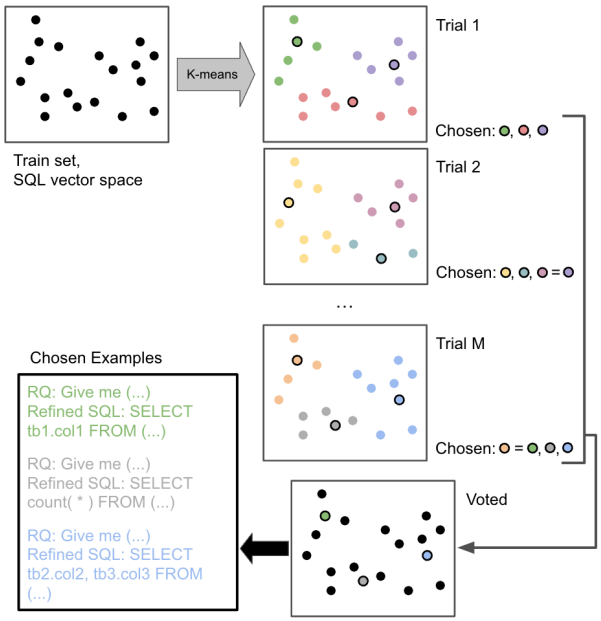

Figure 3: Example selection process using diversity sampling and subsequent example voting. Centroids obtained through the k-means clustering ($k = N$) are used to pick N examples. After repeating this selection M times, aggregated in-context examples are voted by their counts to finally choose N examples.

4 Experiments

4.1 Dataset

The most popular benchmark for the text-to-SQL task is Spider dataset (Yu et al., 2018), which is a large-scale, complex, and cross-domain dataset with 10k+ questions with annotated SQL queries and covers 200 different databases across 138 domains. SParC (Yu et al., 2019b) is a multi-turn version of the Spider dataset which covers the same sets of databases as Spider. CoSQL (Yu et al., 2019a) is a dialogue version of Spider and SParC, which includes about 3k dialogues with 10k+ annotated SQL queries over the same sets of databases. CoSQL is different from SParC for it contains turns that does not require immediate SQL query response, such as clarification of the question. This makes CoSQL a more suitable dataset for the development of conversational systems for information retrieval from structured knowledge sources. Since CoSQL is the latest and most complex context-dependent text-to-SQL dataset available at the moment of writing, we benchmark our

approach with this dataset.

4.2 Models

Throughout the paper, we experimented with LLMs of Open AI serviced through Microsoft Azure[2]. Beyond the Open AI models, we experimented with `gemini-pro` (Team et al., 2023) serviced through Google AI Python SDK[3]. To test other open-sourced models as well, we also experimented with `SQLCoder-7B-2`[4] and `CodeLlama-13B` (Roziere et al., 2023) using vLLM library (Kwon et al., 2023)[5].

In any experiment, we restricted the output size to 600 tokens to accommodate with the context size limitations. We also set the temperature to 0 for the consistency of the result.

4.3 Evaluation metric

To evaluate the performance of our approach, we use two following metrics as suggested for the SQL-grounded dialogue state tracking task in CoSQL challenge (Yu et al., 2019a):

- **Exact set match without values (EM)**: Measures if the predicted SQL query and the ground truth SQL query are equivalent to each other, by comparing the equivalence of each component of the queries. When multiple parallel items are compared, set equivalence is measured so that it does not prefer a particular ordering. It also masks literal/numeral values when comparing each component.

- **Execution accuracy with values (EX)**: Measures if the both outputs of the predicted SQL query and the ground truth SQL query are equal to each other. To generate actual outcomes based on the database, it uses the values in each query as they are.

The two accuracy metrics are evaluated at the question level (question match, QM) and at the interaction (dialogue) level (interaction match, IM).

[2]https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models. We used OpenAI Python API library (https://github.com/openai/openai-python) for the experiments. This library is under Apache-2.0 license and we complied to the license.

[3]https://github.com/google/generative-ai-python. This library is under Apache-2.0 license and we complied to the license.

[4]https://huggingface.co/defog/sqlcoder-7b-2

[5]https://github.com/vllm-project/vllm. This library is under Apache-2.0 license and we complied to the license.

4.4 Baseline approach: randomly sampled dialogues

For comparison purposes, we establish a baseline few-shot prompting method. It randomly samples multiple dialogues and presents them along with their database schema. The template of this baseline prompt is in Appendix D.

4.5 Experiment results

We report the performances of our baseline approach and CREDIT-SQL on the CoSQL dataset in Table 2, along with the performances of state-of-the-art models. In the dev set, our CREDIT-SQL method outperforms the baseline approach of few-shot prompting using randomly sampled dialogues by 8.7%p in EM for the question match while it outperforms the baseline approach by 2.4%p in EX for the question match. Our approach also shows comparable performance to the state-of-the-art models, by the margin of 0.2%p $\sim$ 1.1%p in EM for the question match on the dev set, and shows the best EX for the question match on the dev set. At the time of writing, our approach ranks 4th in EM-QM and 1st in EX-QM on the CoSQL dev set among the models reported on the CoSQL leaderboard.

5 Discussion

5.1 Effectiveness of SQL regularization and diversity sampling

We conducted an ablation study to find out the effectiveness of each module of our CREDIT-SQL approach. The study result is reported in Table 3. This study indicates that subtracting SQL regularization from the prompt drops the EM (EX) for the question match by 0.6%p (0.2%p). When the prompt with examples collected by diversity sampling is replaced with a prompt with randomly sampled examples, the EM (EX) for the question match drops by 5.5%p (2.7%p).

5.2 Effectiveness of example voting

To study the effectiveness of example voting in CREDIT-SQL, we investigated the performance of different methods to aggregate multiple sets of diversity-sampled examples with distinct random seeds. In particular, we plot EM and EX for the question match on the CoSQL dev set of those different methods versus the number of aggregated sets of diversity-sampled examples in Fig 4. These aggregation methods include: (1) the average of each set's performance, (2) the maximum of each

Model	EM (%)				EX (%)			
	QM		IM		QM		IM	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test
STAR (Cai et al., 2022)	59.7	57.8	30.0	28.2	-	-	-	-
CQR-SQL (Xiao et al., 2022)	58.5	58.3	31.1	27.4	-	-	-	-
RASAT+PICARD (Qi et al., 2022)	58.8	55.7	27.0	26.5	67.0	66.3	39.6	37.4
Few-shot Prompting								
Baseline, randomly sampled dialogues	49.9	-	21.6	-	65.0	-	33.2	-
CREDIT-SQL	58.6	-	25.8	-	68.6	-	36.5	-

Table 2: Results on the CoSQL dataset. Exact set match without values (EM) and execution accuracy with values (EX) are presented for both the question match (QM) and the interaction match (IM). For the few-shot prompting methods, we present the results of the baseline approach (randomly sampled dialogues) as well as the results of CREDIT-SQL, both with the average performance of 5 repeated experiments on `gpt-3.5-turbo-0301`. Results for other models are as reported in the literature for comparison.

Model	EM (%)		EX (%)	
	QM	IM	QM	IM
CREDIT-SQL	58.6	25.8	68.6	36.5
w/o SQL Reg.	58.0	24.9	68.4	36.2
w/o Div. Prompt	53.1	20.1	65.9	33.1

Table 3: Ablation studies for CREDIT-SQL on the CoSQL Dev set. We used `gpt-3.5-turbo-0301` and used 18 examples out of 20 votes for the text-to-SQL. The results without SQL regularization and the results without diversity prompt are presented. Each experiment is repeated 5 times and the average performance is reported.

Clustering Method	EM (%)		EX (%)	
	QM	IM	QM	IM
k-means	58.6	25.8	68.6	36.5
Agglomerative	58.1	26.3	68.2	36.5
Spectral	56.3	23.5	67.7	34.5

Table 4: Comparison of different clustering methods. We used `gpt-3.5-turbo-0301` and used 18 examples out of 20 votes for the text-to-SQL. Agglomerative clustering method here used Ward linkage, and spectral method used k-means to cluster the spectrum.

set's performance, (3) the performance of EM-based consistency voting from each set's SQL results, and (4) the performance of the prompt made of voted examples among the all examples of the given sets. The last method is adopted for our CREDIT-SQL. As illustrated in Fig 4, the voted-example prompt outperforms either the average or the maximum of the individual results of distinct diversity-sampled prompts for the number of sets around 12 or more. Moreover, our proposed voted-example prompt performs similar to or better than the popular method of voting on SQL results, for the number of sets around 12 or more. Furthermore, the voted-example prompt is more efficient in the sense that it only uses a single inference of text-to-SQL per each SQL query regardless of the aggregation number M, while the consistency voting requires M inferences to aggregate M sets of examples. This may suggest a new possibility for efficiently mitigating the instability of example selection in the few-shot prompting with LLM in general.

5.3 Effects of number of examples in text-to-SQL prompt

We investigated the effects of number of examples in text-to-SQL prompt in the performance of the model in Fig 5. Due to the limited context size, we could not experiment number of examples beyond 21. Within the number of examples we investigated, there was no clear correlation between the performance of the model and the number of examples used. We opted best number of examples within our hyperparameter search.

5.4 Effects of different clustering methods

We investigated the effects of different clustering methods in text-to-SQL prompt in the performance of the model in Table 4. While we opted k-means for the CREDIT-SQL, agglomerative clustering with Ward linkage (Ward Jr, 1963) also showed similar performances on the CoSQL dev set. The spectral clustering (Shi, 2003) was behind the other two clustering methods in the performances on the same dev set.

LLM	Number of examples	Completion method	EM (%)		EX (%)	
			QM	IM	QM	IM
gpt-3.5-turbo-0301	18	Text	58.6	25.8	68.6	36.5
gpt-3.5-turbo-0301	12	Text	57.3	24.6	67.4	35.2
gpt-3.5-turbo-instruct	12	Text	54.9	23.9	65.7	32.4
gpt-3.5-turbo-16k	18	Chat	51.7	19.8	63.3	29.7
gpt-3.5-turbo-16k	50	Chat	51.5	18.8	64.1	30.4
gpt-4-turbo	18	Chat	54.0	21.5	65.2	32.1
gemini-pro	18	Text	51.3	16.4	63.9	29.0
SQLCoder-7B-2	18	Text	33.1	6.8	46.1	15.4
CodeLlama-13B	18	Text	28.6	5.8	34.3	8.9

Table 5: Performances of CREDIT-SQL on the CoSQL dev set, with different LLMs. We present the number of text-to-SQL examples as well as the completion method used for each experiment. For the chat completion method, we input the entire few-shot prompt as a system message.

5.5 Performance analysis by question difficulty

CoSQL provides the difficulty of each question based on the components of the golden SQL query for that question. Here we analyze the performance of the best-performing CREDIT-SQL on the CoSQL dev set by the question difficulty (Figure 6). As anticipated, both the execution accuracy and the exact set match decreases as the question difficulty increases.

5.6 Performances with different LLMs

To determine the LLM model to be used for the CREDIT-SQL, we evaluated the CoSQL dev set with different LLMs (Table 5). Within the OpenAI models (`gpt-...`), `gpt-3.5-turbo-0301` performed the best while we observed that the text completion method outperforms the chat completion method significantly. Other models beyond OpenAI models were not performing as well as other OpenAI models while `gemini-pro` showed the best performance among them. `SQLCoder-7B-2`, which is a fine-tuned CodeLlama model to the text-to-SQL task, performed better than the larger size CodeLlama model, `CodeLlama-13B`. Although we tested only 7B model for the resource limitation, the benchmark on the model card of `SQLCoder-7B-2`[6] indicates that the performance gain of using a larger model might be limited to a few percent points.

5.7 Error analysis

To understand the cases in which CREDIT-SQL does not perform well, we performed an error analysis on the results of the best-performing CREDIT-SQL. In particular, we present questions with the incorrect exact set match at the question match level in Figure 7. To categorize the errors, we used the keyword analysis provided by the official evaluation code[7] for the CoSQL dataset (Zhong et al., 2020). To understand how much the exact set match errors and the execution accuracy errors are correlated, we also present the pie chart that describes the correlation of those two metrics in Figure 7. Further case studies of errors can be found in Appendix E.

5.8 Limitations and future works

Since our work focused on the CoSQL dataset, the prompts we suggest in this paper might have difficulty in generalizing to the databases and SQL queries outside the CoSQL dataset. Indeed, testing the generalizability of CREDIT-SQL to other context-dependent SQL datasets would be an interesting subject for future research. While we tested several other LLMs beyond OpenAI models, one may test the performance of CRDEIT-SQL on a further variety of the latest LLMs for future research. Our ablation study on the number of text-to-SQL examples was rather inconclusive due to the limited context size, so an investigation of the effects of the number of examples on a LLM with far larger context size can provide a good insight into our approach.

6 Conclusion

In this paper, we propose a few-shot prompting method called CREDIT-SQL which is the first few-

[6] https://huggingface.co/defog/sqlcoder-7b-2

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

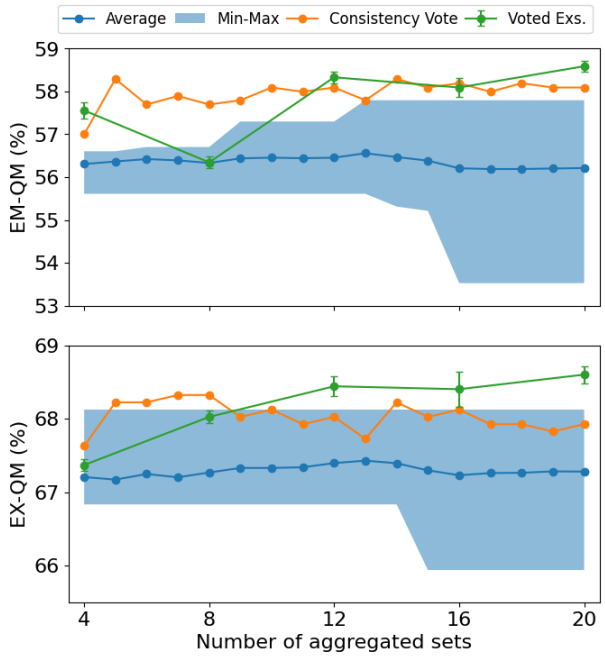

Figure 4: Comparison of different methods of aggregating multiple diversity-sampled sets of examples generated with distinct random seeds. We present EM-QM and EX-QM on the CoSQL dev set for comparison. For this study, We used `gpt-3.5-turbo-0301` and used 18 examples out of 20 votes for the text-to-SQL. **Blue solid line:** Average of the results generated by prompts with distinct random seeds, up to the given number of prompts. **Blue shades:** The minimum to the maximum range for the results generated by prompts with distinct random seeds, up to the given number of prompts. **Orange:** The EM-based consistency voting results on the SQL results of the given number of sets. **Green:** The result of the prompt made of voted examples out of all examples from the given number of prompts. Average performance over 5 repetitions is reported along with the error bar size of the standard deviation.

shot prompting approach to perform the dialogue state tracking task with systematically selected in-context examples out of the entire training data. CREDIT-SQL splits each dialogue state tracking task into multiple question-query pair text-to-SQL tasks by question rephrasing and utilizes the diversity sampling and subsequent in-context example voting to prepare the few-shot prompts for the text-to-SQL tasks. Experiments demonstrate that CREDIT-SQL achieves a performance comparable to the state-of-the-art models. Also, the technique of example voting used in CREDIT-SQL suggests a new way to mitigate the instability of in-context example selection in the generic few-shot prompting setting.

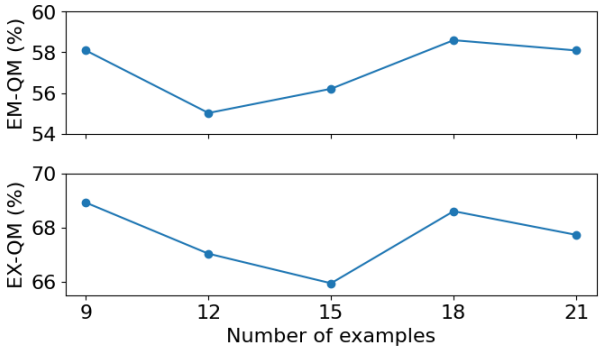

Figure 5: EM-QM and EX-QM on the CoSQL dev set for the different number of examples. We used `gpt-3.5-turbo-0301` and used 18 examples out of 20 votes for the text-to-SQL.

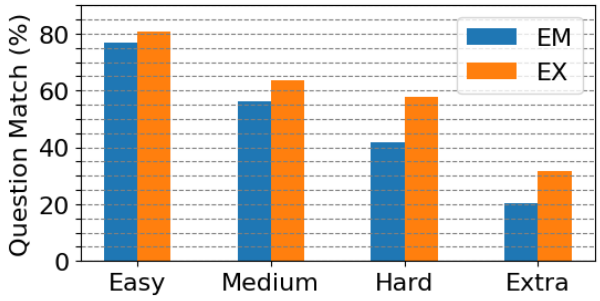

Figure 6: Performance of CREDIT-SQL on the CoSQL dev set by the question difficulty. We present the EM-QM and EX-QM along with 4 difficulty categories: easy, medium, hard, and extra.

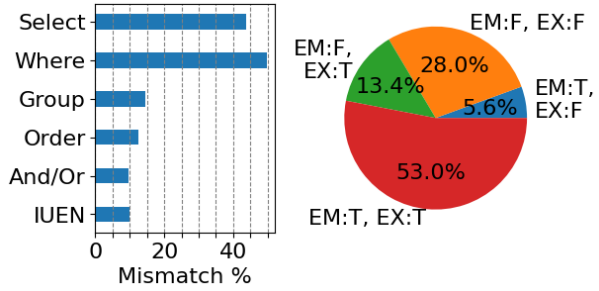

Figure 7: **Left:** Error analysis of CREDIT-SQL on the exact set match errors at the question match level. Here we present the mismatch percentage for each category of SQL query keywords. Here IUEN stands for IN, UNION, EXCEPT, or NOT IN. Since each incorrect SQL query may contain multiple mismatches, mismatch percentages for different categories are not disjoint. **Right:** Correlation of EM and EX, evaluated on the CoSQL dev set at the question match level.

References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng

Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. Palm 2 technical report.

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.

Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems*, 34:7664–7676.

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, and Yongbin Li. 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, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. 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 2541–2555, Online. Association for Computational Linguistics.

Linzheng Chai, Dongling Xiao, Jian Yang, Liqun Yang, Qian-Wen Zhang, Yunbo Cao, Zhoujun Li, and Zhao Yan. 2023. Qurg: Question rewriting guided context-dependent text-to-sql semantic parsing. *arXiv preprint arXiv:2305.06655*.

Zhi Chen, Lu Chen, Hanqi Li, Ruisheng Cao, Da Ma, Mengyue Wu, and Kai Yu. 2021. 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, Online. Association for Computational Linguistics.

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

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

Yushi Hu, Chia-Hsuan Lee, Tianbao Xie, Tao Yu, Noah A Smith, and Mari Ostendorf. 2022. In-context learning for few-shot dialogue state tracking. *arXiv preprint arXiv:2203.08568*.

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 exploration network for cross-domain context-dependent semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13116–13124.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. 2022. S^2SQL: Injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1254–1262, Dublin, Ireland. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model

serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.

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. *arXiv preprint arXiv:2305.12586*.

OpenAI. 2023. Gpt-4 technical report.

Zhufeng Pan, Kun Bai, Yan Wang, Lianqiang Zhou, and Xiaojiang Liu. 2019. Improving open-domain dialogue systems via multi-turn incomplete utterance restoration. 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 1824–1833, Hong Kong, China. Association for Computational Linguistics.

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

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. RASAT: Integrating relational structures into pretrained Seq2Seq model for text-to-SQL. 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.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

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.

Shi. 2003. Multiclass spectral clustering. In *Proceedings ninth IEEE international conference on computer vision*, pages 313–319. IEEE.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020a. 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.

Kai Wang, Weizhou Shen, Yunyi Yang, Xiaojun Quan, and Rui Wang. 2020b. Relational graph attention network for aspect-based sentiment analysis. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3229–

3238, Online. Association for Computational Linguistics.

Run-Ze Wang, Zhen-Hua Ling, Jingbo Zhou, and Yu Hu. 2021. Tracking interaction states for multi-turn text-to-sql semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13979–13987.

Joe H Ward Jr. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244.

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, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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.

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, 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.

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.

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.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suite. In *The 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

A Question Rephrasing Prompt Base

For the question rephrasing prompt, we randomly sampled 20 examples from the CoSQL training data. Each question in the sampled examples is manually rephrased so that the rephrased question starts with "Give me ..." and the question is self-contained without referring to any previous context. Following is the question rephrasing prompt base we used for CREDIT-SQL:

Make sure the rephrased questions do not need any implicit information of the previous dialogue.

Database: department_store
Q1: List the name of all suppliers in alphabetical order of their addresses.
A1: Okay. I rephrased your question as "Give me the names of all suppliers, ordered alphabetically by their addresses."
Q2: And their phone numbers?
A2: Okay. I rephrased your question as "Give me the phone numbers of all suppliers, ordered alphabetically by their addresses."
Q3: What states is Tesco located in?
A3: do you mean the address id or the address details?
Q4: The address details.
A4: Okay. I rephrased your question as "Give me the address details of Tesco."
Q5: How about Lidl?
A5: Okay. I rephrased your question as "Give me the address details of Lidl."

Database: theme_gallery
Q1: Show the average ticket prices for exhibitions for all years before 2009?
A1: Okay. I rephrased your question as "Give me the average ticket price of all exhibitions whose year before 2009."
Q2: Show minimum ticket prices for exhibitions for all years before 2009?
A2: Okay. I rephrased your question as "Give me the minimum ticket price of all exhibitions whose year before 2009."
Q3: Show maximum ticket prices for exhibitions for all years before 2009?
A3: Okay. I rephrased your question as "Give me the maximum ticket price of all exhibitions whose year before 2009."
Q4: Show average ticket price for exhibitions for year 2005?
A4: Okay. I rephrased your question as "Give me the average ticket price of all exhibitions whose year is 2005."
Q5: Show average ticket price for exhibitions for all years before 2006?
A5: Okay. I rephrased your question as "Give me the average ticket price of all exhibitions whose year before 2006."

Database: apartment_rentals
Q1: How many bathroom count those building 809 has?
A1: The total number of bathrooms building 809 has is NULL.
Q2: How about building 808?
A2: Okay. I rephrased your question as "Give me the total number of bathroom count of all building 809."
Q3: How about the total number of rooms it has?
A3: Okay. I rephrased your question as "Give me the total number of rooms of all building 809."

Database: wine_1
Q1: Which wine has the highest price?
A1: Okay. I rephrased your question as "Give me the wine with the highest price."
Q2: Which wine has the highest score?
A2: Okay. I rephrased your question as "Give me the wine with the highest score."
Q3: Which wine has the most cases available?
A3: Okay. I rephrased your question as "Give me the wine with the most cases available."
Q4: How about winery with the least cases?
A4: Okay. I rephrased your question as "Give me the winery with the least cases."

Database: dorm_1
Q1: How many students live in each dorm?
A1: Okay. I rephrased your question as "Give me the total number of students of each dorm."
Q2: What is the average age of the students living in each dorm?
A2: Okay. I rephrased your question as "Give me the average number of students of each dorm."

Database: roller_coaster
Q1: what is the highest length of coaster
A1: Okay. I rephrased your question as "Give me the highest length of all costers."
Q2: which roller coaster are top speed

A2: Okay. I rephrased your question as "Give me the roller coaster with highest speed."
Q3: what is the height of Boardwalk Bullet
A3: Okay. I rephrased your question as "Give me the height of Boardwalk Bullet."

Database: entrepreneur
Q1: What is the largest weight that appears in the people table?
A1: Okay. I rephrased your question as "Give me the maximum weight of all people."
Q2: Can you add the people id associated with that weight into the table?
A2: Okay. I rephrased your question as "Give me the maximum weight of all people and the people id of that person."
Q3: Can you also add the name associated with that people id into the table?
A3: Okay. I rephrased your question as "Give me the maximum weight of all people and the people id, name of that person."

Database: customers_campaigns_ecommerce
Q1: Can you tell me the highest number of mailshots for the customers?
A1: Okay. I rephrased your question as "Give me the highest number of mailshots for the customers."
Q2: What about the highest mailshot id number?
A2: Okay. I rephrased your question as "Give me the mailshot id number of mailshot with the highest number of mailshots."
Q3: What is the customer id for that mailshot id?
A3: Okay. I rephrased your question as "Give me the customer id number of mailshot with the highest number of mailshots."
Q4: How many mailshots does customer id 1 have?
A4: Okay. I rephrased your question as "Give me the total number of mailshots of customer id 1."
Q5: What is the name of customer id 1?
A5: Okay. I rephrased your question as "Give me the name of customer id 1."
Q6: What is the amount of mailshots that Wendell has?
A6: Okay. I rephrased your question as "Give me the total number of mailshots of Wendell."

Database: solvency_ii
Q1: What is the most common type code of all the products?
A1: Okay. I rephrased your question as "Give me the type code of all products with the maximum number of type code."
Q2: What are the product names of all products with the type code "Food"?
A2: Okay. I rephrased your question as "Give me the product names of all products with the type code "Food"."
Q3: What are the prices of those?
A3: Okay. I rephrased your question as "Give me the prices of all products with the type code "Food"."
Q4: What are the names and prices of all products with the type code books?
A4: Okay. I rephrased your question as "Give me the names and prices of all products with the type code books."
Q5: What are the name and type code of the product with the highest price?
A5: Okay. I rephrased your question as "Give me the name and type code of product with the highest price."
Q6: Can you list all product names and their corresponding type codes of the products with prices below the average price?
A6: Okay. I rephrased your question as "Give me the names and product codes of all products with prices below the average price of all products."

Database: student_assessment
Q1: what is the course id of English course?
A1: Okay. I rephrased your question as "Give me the course id of English course."
Q2: name the student ids who attend the statastic course?in order of attendance date
A2: Okay. I rephrased your question as "Give me the student ids of all students who attend the statastic course, ordered by the attendence date."
Q3: what is the date of attendance of the student is 121?
A3: the date of attendance of the student with id 121 is 2012-04-09 11:44:34.
Q4: which student id attend english and french both course?
A4: Okay. I rephrased your question as "Give me the student ids of all students who attend both courses of English and French."

Database: chinook_1
Q1: what are the durations of the longest tracts?
A1: Okay. I rephrased your question as "Give me the durations of the longest tracts."
Q2: what are the shortest tracks?

A2: Okay. I rephrased your question as "Give me the durations of the shortest tracks."
Q3: How many composer are there?
A3: Okay. I rephrased your question as "Give me the number of composers."

Database: activity_1
Q1: Hello, what are the names of the faculty members who don't participate in any activity?
A1: Okay. I rephrased your question as "Give me the names of all faculty members who do not participate in any activity."
Q2: Which faculty members participate in activities?
A2: Okay. I rephrased your question as "Give me the names of all faculty members.who participate in any activity."
Q3: What activities do Michael Goodrich and Gerald Masson participate in?
A3: Okay. I rephrased your question as "Give me the activities that both of Michael Goodrich and Gerald Masson participate in."
Q4: In total, how many faculty members are there?
A4: Okay. I rephrased your question as "Give me the total numbe of faculty members."

Database: product_catalog
Q1: what are the names of all catalogs
A1: Okay. I rephrased your question as "Give me the names of all catalogs."
Q2: list the publication dates for all catalogs
A2: Okay. I rephrased your question as "Give me the publication dates of all catalogs."
Q3: list the catalogs that have level number greater than 5
A3: Do you want catalog name or id?
Q4: id
A4: Okay. I rephrased your question as "Give me the catalog id of catalogs with level number greater than 5."

Database: hr_1
Q1: Show me all the employees in the same department as Clara.
A1: Here is all the information of all the employees in the same department as the employee with a first name Clara.
Q2: Can you show me their hire dates?
A2: Okay. I rephrased your question as "Give me the hire dates of all employees with department same as the department of Clara."
Q3: What is their average salary?
A3: Okay. I rephrased your question as "Give me the average salary of all employees with department same as the department of Clara."

Database: store_product
Q1: Please list the headquarters of the store that is called Blackville?
A1: Okay. I rephrased your question as "Give me the headquarters of all stores with name Blackville."
Q2: What is the population of that city?
A2: Okay. I rephrased your question as "Give me the population of all cities where headquarters of all stores with name Blackville are located."
Q3: What is the name of the store in the city with the greatest population?
A3: Okay. I rephrased your question as "Give me the name of the store in the city with the greatest population."

Database: school_bus
Q1: How many drivers are there?
A1: Okay. I rephrased your question as "Give me the total number of all drivers."
Q2: What is the average age of your drivers?
A2: Okay. I rephrased your question as "Give me the average age of all drivers."

Database: customers_and_addresses
Q1: Can you give me a list of all products?
A1: Okay. I rephrased your question as "Give me the all products."
Q2: Could you order them by order quantity?
A2: From small to large?
Q3: large to small
A3: Okay. I rephrased your question as "Give me the all products, ordered by order quantity in descending order."

Database: game_1
Q1: Please show me the ids of the students who play video games and play sports?
A1: Okay. I rephrased your question as "Give me the ids of all students who play both of video games and sports."

Database: protein_institute
Q1: show the institute type with lowest number of institutions
A1: Okay. I rephrased your question as "Give

me the institute type with the lowest number of institutions."
Q2: show the institute type with highest number of institutions
A2: Okay. I rephrased your question as "Give me the institute type with the highest number of institutions."
Q3: which university is founded first
A3: Okay. I rephrased your question as "Give me the university with the earliest foundation date."

Database: formula_1
Q1: list the forename of all races
A1: Okay. I rephrased your question as "Give me the forenames of all races."
Q2: what is the forename of the race that occurred most recently?
A2: Okay. I rephrased your question as "Give me the forename of the race with the latest occurance date."
Q3: what is its code?
A3: Do you mean circuit id?
Q4: yes
A4: Okay. I rephrased your question as "Give me the circuit id of the race with the latest occurance date."

B In-context Examples Selected via Voting for CREDIT-SQL Text-to-SQL Prompt

We present the 18 voted examples we used for the text-to-SQL prompt of our CREDIT-SQL approach in Table 6. We performed k-means clustering with $k = 18$ and voted examples collected from 20 trials of clustering with distinct random seeds.

C Sample Text-to-SQL Prompt for CREDIT-SQL

Following is a sample text-to-SQL prompt for CREDIT-SQL in the case of $N = 2$:

Generate the SQL query for each question about the given database schema.

Database: party_host
Table host, columns = [*, host_id, name, nationality, age]
Table party, columns = [*, party_id, party_theme, location, first_year, last_year, number_of_hosts]
Table party_host, columns = [*, party_id, host_id, is_main_in_charge] Foreign_keys = [party_host.party_id = party.party_id, party_host.host_id = host.host_id]
Q: Give me the names of all nations with hosts younger than 35.
SQL: SELECT DISTINCT host.nationality FROM host WHERE host.age < 35

Database: company_1
Table department, columns = [*, name, number, mgr_ssn, mgr_start_date]
Table dependent, columns = [*, essn, dependent_name, sex, bdate, relationship]
Table dept_locations, columns = [*, number, location]
Table employee, columns = [*, fname, minit, lname, ssn, bdate, address, sex, salary, super_ssn, dno]
Table project, columns = [*, pname, number, location, dnum]
Table works_on, columns = [*, essn, pno, hours]
Foreign_keys = []
Q: Give me the total number of employees with salary lower than 50000.
SQL: SELECT count (*) FROM employee WHERE employee.salary < 50000

D Sample Prompt for the Baseline Approach

Following is a sample prompt for the baseline approach with 2 dialogues:

Generate the SQL query for each question about the given database schema.

Database: inn_1
Table reservations, columns = [*, code, room, check-in, checkout, rate, lastname, firstname, adults, kids]
Table rooms, columns = [*, roomid, roomname, beds, bedtype, maxoccupancy, baseprice, decor]
Foreign_keys = [reservations.room = rooms.roomid]
Q: which rooms have a king bed?
A: SQL query you requested is [SELECT * FROM Rooms WHERE bedtype = 'King']
Q: what are the decor of those rooms?
A: SQL query you requested is [SELECT distinct decor FROM Rooms WHERE bedtype = 'King']
Q: show me the names and the decor of the rooms that have a king bed
A: SQL query you requested is [SELECT roomName, decor FROM Rooms WHERE bedtype =

'King']
Q: show me the price of the rooms also
A: SQL query you requested is [SELECT roomName, decor, basePrice FROM Rooms WHERE bedtype = 'King']
Q: now sort the list by price
A: SQL query you requested is [SELECT roomName, decor, basePrice FROM Rooms WHERE bedtype = 'King' order by basePrice]

Database: restaurant_1
Table restaurant, columns = [*, resid, resname, address, rating]
Table restaurant_type, columns = [*, restypeid, restypename, restypedescription]
Table student, columns = [*, stupid, lname, fname, age, sex, major, advisor, city_code]
Table type_of_restaurant, columns = [*, resid, restypeid]
Table visits_restaurant, columns = [*, stupid, resid, time, spent]
Foreign_keys = [type_of_restaurant.resid = restaurant.resid, type_of_restaurant.restypeid = restaurant_type.restypeid, visits_restaurant.stuid = student.stuid, visits_restaurant.resid = restaurant.resid]
Q: Which city code of Smith?
Q: Is Smith first name or last name?
Q: first name
A: SQL query you requested is [SELECT city_code from Student where fname = "Smith"]
Q: What age of last name Kim?
A: SQL query you requested is [SELECT age from Student where lname = "Kim"]
Q: What are the student id of city code WAS?
A: SQL query you requested is [SELECT stuid from Student where city_code = "WAS"]
Q: Which major has least number of students? List the major and the number of students.
A: SQL query you requested is [SELECT Major , count (*) FROM Student GROUP BY Major ORDER BY count (Major) ASC LIMIT 1]

E Error analysis: case studies

To understand types of errors CREDIT-SQL is generating, we perform case studies on the errors from the evaluation results on the CoSQL dev set. In particular, we focus on examples from hard difficulty and extra difficulty which the model is most struggling with (Figure 6). In the following examples, 'DB' indicates the database id, 'RQ' indicates the rephrased question by CREDIT-SQL, 'G' indicates the golden SQL query, 'P' indicates the predicted SQL query by CREDIT-SQL.

```
Example 1
DB: dog_kennel
RQ: Give me the first names of all
    professionals or owners.
G: SELECT first_name FROM Professionals UNION
    SELECT first_name FROM Owners
P: SELECT owners.first_name FROM owners UNION
    SELECT professionals.first_name FROM
    professionals
EM: False, EX: True
```

In Example 1, we can see a typical case where the exact match fails while the execution results are identical. In this particular case, this happens because the order of two tables (professionals and owners) is not preserved in the predicted SQL query, though there is no semantical difference.

```
Example 2
DB: battle_death
RQ: Give me the names of all battles with no
    ships lost in the English Channel.
G: SELECT name FROM battle EXCEPT SELECT T1.
    name FROM battle AS T1 JOIN ship AS T2 ON
    T1.id = T2.lost_in_battle WHERE T2.location
     = 'English Channel'
P: SELECT battle.name FROM battle JOIN ship ON
    battle.id = ship.lost_in_battle WHERE ship.
    id IS NULL AND ship.location = 'English
    Channel'
EM: False, EX: False
```

As demonstrated in Example 2, CREDIT-SQL predictions on questions involving the exclusion of certain groups of data often include `IS NULL`, but many such attempts incorrectly lead to the empty results.

```
Example 3
DB: dog_kennel
RQ: Give me the average age of all dogs that
    have gone through any treatment.
G: SELECT avg ( age ) FROM Dogs WHERE dog_id IN
     ( SELECT dog_id FROM Treatments )
P: SELECT avg ( dogs.age ) FROM dogs JOIN
    treatments ON dogs.dog_id = treatments.
    dog_id
EM: False, EX: False
```

The prediction in Example 3 could generate the correct execution result if the `dog_id` column was unique in both of `dogs` and `treatments` tables. However, this was not the case for the `treatments` table, and therefore the average age evaluated from the prediction in Example 3 counts dogs received multiple treatments excessively.

```
Example 4
DB: student_transcripts_tracking
RQ: Give me the name of the course with the
    least number of students of enrollments.
G: SELECT T1.course_name FROM Courses AS T1
    JOIN Student_Enrolment_Courses AS T2 ON T1.
    course_id = T2.course_id GROUP BY T1.
    course_name ORDER BY count ( * ) LIMIT 1
P: SELECT courses.course_name FROM courses JOIN
     student_enrolment_courses ON courses.
    course_id = student_enrolment_courses.
    course_id GROUP BY courses.course_id ORDER
    BY count ( * ) ASC LIMIT 1
EM: False, EX: False
```

The prediction in Example 4 indeed generates the execution result that the question asked. The problem in this case is that there are multiple courses with the least number of enrolled students, and only one of them is chosen by the column used for `GROUP BY`. Both `course_id` and `course_name` are unique in the `courses` table and therefore they are all acceptable columns, though they generate different execution results in this example.

```
Example 5
DB: car_1
RQ: Give me the names of all countries in
    Europe with at least 3 car manufacturers.
G: SELECT T1.CountryName FROM COUNTRIES AS T1
    JOIN CONTINENTS AS T2 ON T1.Continent = T2.
    ContId JOIN CAR_MAKERS AS T3 ON T1.
    CountryId = T3.Country WHERE T2.Continent =
     'europe' GROUP BY T1.CountryName HAVING
    count ( * ) > = 3
P: SELECT countries.countryname FROM countries
    JOIN car_makers ON countries.countryid =
    car_makers.country JOIN car_names ON
    car_makers.id = car_names.makeid GROUP BY
    countries.countryname HAVING count ( * ) >=
     3 AND countries.continent = 'Europe'
EM: False, EX: False
```

The prediction in Example 5 is semantically fine if not looking into the actual contents of the database. The problem in this case is that the `continent` column in the `countries` table is not presented as the actual continent name but as the continental code. If one replaces the text field `'Europe'` with numeral field 2, this prediction query generates identical execution results with the golden query.

F Details of SQL regularization

To regularize SQL queries, we use a `SQLObject` class. A Python pseudocode of this class is presented in Figure 8. To regularize a SQL query, we instantiate a `SQLObject` object initialized with the input SQL query, and then run `get_sql_string` method to get a regularized SQL query.

G Details of SQL vectorization

To vectorize SQL queries, we use a `SQLVectorizer` class which inherits `SQLObject` class in Figure 8. A Python pseudocode of this class is presented in Figure 9. To regularize a SQL query, we instantiate a `SQLVectorizer` object initialized with the input SQL query, and then run `vectorize` method to get a regularized SQL query.

```
class SQLObject:
    def __init__(self, schema_info, sql=None, words_and_texts=None):
        self.schema_info = schema_info
        self.components = {}
        if sql is not None:
            self.read_sql(sql)
        elif words_and_text is not None:
            words, texts = words_and_texts
            self.parse_components(words, texts)

    # parse SQL query into components
    def read_sql(self, sql):
        # replace text fields with quotation marks into [text1], [text2], etc., while storing original text fields
        new_sql, texts = self.replace_text_fields(sql)
        # parse words(column, table, operator, paratheses, texts) and locate the original text fields in the word sequence
        words, texts = self.parse_words(new_sql, texts)
        self.parse_componenets(words, texts)

    # parse components(joint, select, where, group by, order by, limit etc.) from given words and text fields
    def parse_components(self, words, texts):
        if any of ('intersect', 'union', 'except') in words:
            conj_op, conj_op_idx = (conjunction operator and its location)
            obj_left = SQLObject(self.schema_info, (words[:conj_op_idx], texts))
            obj_right = SQLObject(self.schema_info, (words[(conj_op_idx+1):], texts))
            self.components['joint'] = (conjunction_operator, obj_left, obj_right)
        else:
            self.parse_nonnested_components(words, texts)

    # parse the contents followed by each keyword in nonnested SQL query
    def parse_nonnested_components(self, words, texts):
        for keyword in ['from', 'select', 'where', 'group by', 'having', 'order by', 'limit']:
            followed_words = (words followed by the keyword in the given word sequence)

            if keyword == 'from':
                self.table_alias = (table alias dictionary built from field under 'from')
                field = (field under 'from' without alias; ex: from tb1 as T1 -> from tb1)
            else:
                contents = (replace any reference on table alias to the table name;
                            ex: T1.col1, col2 -> tb1.col1, tb1.col2)
            self.components[keyword] = contents

    # assemble components of SQL object into a SQL query
    def get_sql_string(self):
        text = ''
        for keyword in ['select', 'from', 'where', 'group by', 'having', 'order by', 'limit']:
            if keyword in self.components:
                expr = (lower case of self.components[keyword] except the text field.
                        all text field is surrounded by the quotation mark "'")
                text += keyword + ' ' + expr + ' '
        return text.strip()
```

Figure 8: Pseudocode for `SQLObject` class.

1	Give me the invoice dates of all customers. [chinook_1]
	SELECT customer.firstname , customer.lastname , invoice.invoicedate FROM customer JOIN invoice ON customer.customerid = invoice.customerid
2	Give me the event id of all events that have a participant with the detail Kenyatta Kuhn. [local_govt_in_alabama]
	SELECT events.event_id FROM events JOIN participants_in_events ON participants_in_events.event_id = events.event_id JOIN participants ON participants_in_events.participant_id = participants.participant_id WHERE participants.participant_details = 'Kenyatta Kuhn'
3	Give me the patient id of the appointment with the most recent start date. [hospital_1]
	SELECT appointment.patient FROM appointment ORDER BY appointment.start DESC LIMIT 1
4	Give me the minimum and maximum number of bathrooms and bedrooms of all the apartments. [apartment_rentals]
	SELECT min (apartments.bathroom_count) , max (apartments.bathroom_count) , min (apartments.bedroom_count) , max (apartments.bedroom_count) FROM apartments
5	Give me the total number of all professors. [college_1]
	SELECT count (*) FROM professor
6	Give me the names of 5 products that are not in any event. [solvency_ii]
	SELECT products.product_name FROM products WHERE products.product_id NOT IN (SELECT products_in_events.product_id FROM products_in_events)
7	Give me the total number of students who play football. [game_1]
	SELECT count (*) FROM sportsinfo JOIN student ON sportsinfo.stuid = student.stuid WHERE sportsinfo.sportname = 'Football'
8	Give me the names of all instructors who are advising more than one student. [college_2]
	SELECT instructor.name FROM instructor JOIN advisor ON instructor.id = advisor.i_id GROUP BY advisor.i_id HAVING count (*) > 1
9	Give me the account types of all customers whose credit score is above 100. [loan_1]
	SELECT customer.acc_type FROM customer
10	Give me the total number of students who have behavior incident reports with recommendations. [behavior_monitoring]
	SELECT count (*) FROM (SELECT * FROM behavior_incident JOIN students ON behavior_incident.student_id = students.student_id GROUP BY behavior_incident.student_id)
11	Give me the total number of residents of each property, and the property id. [local_govt_and_lot]
	SELECT properties.property_id , count (*) FROM properties JOIN residents ON properties.property_id = residents.property_id GROUP BY properties.property_id
12	Give me the claim id of the claim that incurred the most number of settlements. [insurance_policies]
	SELECT claims.claim_id FROM claims JOIN settlements ON claims.claim_id = settlements.claim_id GROUP BY claims.claim_id ORDER BY count (*) DESC LIMIT 1
13	Give me the date of ceremony of all music festivals with category 'best song' and 'awarded'. [music_4]
	SELECT music_festival.date_of_ceremony FROM music_festival WHERE music_festival.category = 'Best Song' AND music_festival.result = 'Awarded'
14	Give me the ids of all employees with role Role_Code. [cre_Doc_Tracking_DB]
	SELECT employees.employee_id , employees.role_code FROM employees
15	Give me the first names of all students who have a dorm id of 160. [dorm_1]
	SELECT student.fname FROM student JOIN lives_in ON student.stuid = lives_in.stuid WHERE lives_in.dormid = 160
16	Give me the college name of the employee with name Reggie Lewis. [company_employee]
	SELECT people.graduation_college FROM people WHERE people.name = 'Reggie Lewis'
17	Give me the names of all nations with hosts younger than 35. [party_host]
	SELECT DISTINCT host.nationality FROM host WHERE host.age < 35
18	Give me the total number of customers who pay by Credit card. [customers_campaigns_ecommerce]
	SELECT count (*) FROM customers WHERE customers.payment_method = 'Credit Card'

Table 6: Voted examples used for CREDIT-SQL. The first row of each example: rephrased question [database ID], the second row of each example: regulated SQL query. 19

```
class SQLVectorizer(SQLObject):
    def __init__(self, schema_info, sql):
        super().__init__(schema_info, sql)

    # Count the occurance of each component of SQL query
    def vectorize(self):
        keys = [
            'intersect', 'union', 'except', 'sel_col', 'sel_all', 'sel_count', 'sel_min', 'sel_max', 'sel_avg', 'sel_sum',
            'from_ext', 'from_tb', 'wh_and', 'wh_or', 'wh_in', 'wh_like', 'wh_is', 'wh_eq', 'wh_gl', 'wh_subq',
            'wh_between', 'wh_pm', 'groupby', 'hv_eq', 'hv_g', 'hv_l', 'hv_else', 'hv_count', 'hv_min', 'hv_max',
            'hv_avg', 'hv_sum', 'order_n', 'order_count', 'order_min', 'order_max', 'order_avg', 'order_sum', 'order_asc',
            'order_desc', 'limit'
        ]
        vec = {k: 0 for k in keys}

        if 'joint' in self.components:
            joint, obj1, obj2 = self.components['joint']
            return (get vectors for obj1 and obj2, then average them)

        for col in self.components['select']:
            vec['sel_col'] += 1 if (col is valid column) else 0
            vec['sel_all'] += 1 if (col is wildcard '*') else 0
            for op in ['count', 'min', 'max', 'avg', 'sum']:
                vec[f'sel_{op}'] += 1 if (col has op) else 0

        if 'where' in self.components:
            vec['wh_gl'] += (number of occurances of '<', '>', '<=', or '>=' in self.components['where'])
            vec['wh_eq'] += (number of occurances of '=' or '!=' in self.components['where'])
            vec['wh_pm'] += (number of occurances of '+' or '-' in self.components['where'])
            vec['wh_like'] += (number of occurances of 'ilike' or 'like' in self.components['where'])
            for op in ['and', 'or', 'in', 'is', 'between']:
                vec[f'wh_{op}'] += (number of occurances of op in self.components['where'])
            vec[f'wh_subq'] += (number of subqueries in self.components['where'])

        vec['groupby'] += (number of columns in self.components['where'])

        if 'having' in self.components:
            vec['hv_eq'] += (number of occurances of '=' in self.components['having'])
            vec['hv_g'] += (number of occurances of '>' or '>=' in self.components['where'])
            vec['hv_l'] += (number of occurances of '<' or '<=' in self.components['where'])
            vec['hv_else'] += (number of occurances of all other binary operators in self.components['where'])
            for op in ['count', 'min', 'max', 'avg', 'sum']:
                vec[f'hv_{op}'] += (number of occurances of op in self.components['where'])

        if 'orderby' in self.components:
            vec['order_n'] += (number of columns in self.components['orderby'])
            for op in ['count', 'min', 'max', 'avg', 'sum', 'asc', 'desc']:
                vec[f'order_{op}'] += (number of op in self.components['orderby'])

        vec['limit'] += (0 if 'limit' in self.components 1)

        return vec
```

Figure 9: Pseudocode for `SQLVectorizer` class.