SQLPrompt: In-Context Text-to-SQL with Minimal Labeled Data

Anonymous ACL submission

Abstract

Text-to-SQL aims to automate the process of generating SQL queries on a database from natural language text. In this paper, we propose *SQLPrompt* , a novel method to push the state-of-the-art of Text-to-SQL with in-context learning, leveraging the zero-shot and few-shot adaptation capability of large language models (LLMs). Our method comprises a novel prompt design approach to efficiently consider the database information; execution-based consistency decoding; and employing mixture of prompts and/or LLMs. We show that *SQLPrompt* outperforms previous state-of-the-art for in-context learning with zero labeled data by a large margin, closing the gap with finetuning state-of-the-art with thousands of labeled data.

1 Introduction

Text-to-SQL enables natural language interfaces for SQL query generation. It is crucial for enhancing database accessibility without requiring expertise in SQL, and enabling the development of conversational agents with advanced data analytics Notable recent works for Text-to-SQL, PICARD (Scholak et al., 2021), UnifiedSKG (Xie et al., 2022), and RESDSQL-3B + NatSQL(Li et al., 2023), achieve their state-of-the-art results by finetuning the LLMs with a large number of (text, SQL) pair data samples, followed by customized SQL-specific syntax improvements such as constrained decoding. Recently, massive LLMs such as GPT-3 (Brown et al., 2020), PaLM (Chowdhery et al., 2022), and ChatGPT[1](Stiennon et al., 2020) have demonstrated significant progress using few or zero-shot examples as prompts via in-context learning (Wei et al., 2022). In-context learning comes with numerous benefits, including alleviating expensive training, lowering adaptation data requirements, reducing out-of-distribution issues (e.g. unseen words), and lowering the risk of overfitting (e.g. not generalize). These benefits are also highly important for Text-to-SQL, especially given that collecting data in the form of (text, SQL) pairs can be costly, and there are many different SQL dialects and domain-dependent database types. While CodeX (Chen et al., 2021) and ChatGPT have shown promising results with in-context learning for Text-to-SQL, they have a gap between the fine-tuned counterparts, which are trained on significantly more data (thousands of samples). Our goal in this paper is to push the state-of-the-art in Text-to-SQL with minimal labeled data. We propose a novel method, SQLPrompt, which includes prompt design with database content, execution-based consistency decoding, and a mixure of prompts and LLMs. At zero-shot settings, SQLPrompt achieves the state-of-the-art results for in-context learning, closing the gap with fine-tuning state-of-the-art models that require thousands of samples.

[1] https://chat.openai.com/chat.

2 Methods

2.1 Problem setup for Text-to-SQL

Let q be natural language query and D_q be the database information. Text-to-SQL task is to convert query q into SQL. The database $D_q = \{S, K_p, K_f\}$ includes database schema S primary keys K_p, and foreign keys K_f. S usually contains multiple tables T_t: $S = \{T_1, T_2, ...T_t...\}$. Each table T_t has table name N_t, column names c_j and column data types t_j: $T_k = \{N_k, (c_{k1}, t_{k1}), (c_{k2}, t_{k2}), (c_{kj}, t_{kj})...\}$. Primary keys K_p uniquely identifying rows of each table, and foreign keys K_f join multiple tables.

2.2 Prompt design with database content and primary/foreign keys

We argue that prompts should include all necessary information for SQL generation as if expert humans generate answers for the queries. While

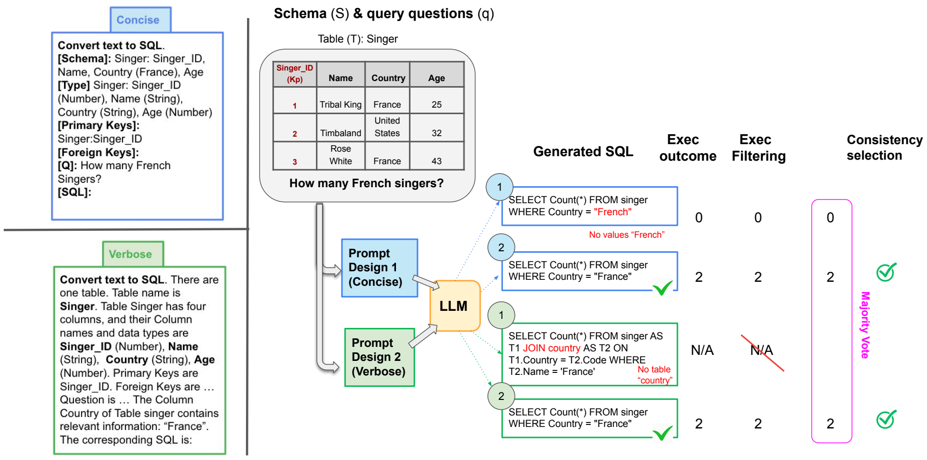

Figure 1: **SQLPrompt Overview**: **(Left) Prompt Design**: Concise prompt design (up) and Verbose prompt design (down). **(Right)** MixPrompt in *SQLPrompt* generates multiple prompts using database and query question, to query LLMs. For each query, LLMs are sampled twice, and two SQLs are generated and executed on the database with errors filtered out. The execution outcomes of both prompt designs are combined to select the most consistent SQL. Without MixPrompt, the true answer cannot be selected with only one prompt (blue) due to a tie situation.

most popular prompt design approaches only include database schema [2], we hypothesize that inclusion of primary and foreign keys, and the database content are crucial, because they help with understanding the schema, linking tables and selecting appropriate columns (Lin et al., 2020; Wang et al., 2020). Refer to Appendix A for more discussion.

Concise database description prompts To prompt LLMs, we linerize information in a table as "Table1 name: column name 1, column name 2 (relevant database content) | Table2 name: column1 ..." (Figure 1, *Concise*. Full example in Appendix B.1). This way describes table structure clearly, but can be less straightforward for LLMs to understand.

Verbose database description prompts We describe databases with human understandable words and emphasize on the information LLMs need to know: e.g. "Table CarNames contains three columns. The column names and their types are : MakeID (number), Model (string) ..."; "Foreign keys are .. Use foreign keys to join Tables". See Appendix B.2 for an example.

2.3 Refinement based on execution-based consistency with MixPrompt

We introduce *MixPrompt*, an in-context learning method based on querying LLMs multiple times and employing execution-based consistency with multiple prompt designs. Self-consistency (Wang et al., 2022), which samples LLMs multiple times to select the most consistent answer, has shown remarkable performance improvements, but its performance with the same prompt design may show saturation beyond some number of samples. Since diversity of outputs is critical for performance, we apply multiple prompt designs, with the assumption that varying prompt designs changes the interface of query and LLMs, leading to more diverse LLM's outputs (Zhou et al., 2022). Then, we select the most consistent answer across different prompt designs.

Suppose $F = \{f_1, f_2, ...\}$ is a collection of different design functions, e.g. f_1 is verbose, f_2 is concise. When we fix LLMs, we have ***MixPrompt*** prediction objectives:

$$p(sql|\text{LLM}, q) = \sum_f p(sql|\text{LLM}, f, q)p(f), \quad (1)$$

where $p(f)$ is mixing coefficient. Since we evenly mixing prompts,$p(f) = 1/nF$ where nF is number of design functions. $p(sql|\text{LLM}, f, q)$ is sampling probability of generating sql.
MixPromt is over-viewed in Fig 1. Specifically, for each design function f, we generate prompts using database D_q and the query q. The trained LLMs specify the distribution $\ell : q \rightarrow sql$, where we can draw sample from:

$$\text{Prompt}_q = f(q, D_q) \quad (2)$$

$$sql_{qf} \underset{i.i.d}{\sim} \text{LLM}(\text{Prompt}_q, r) \quad (3)$$

We sample B times from the LLM with the same

[2]https://platform.openai.com/examples/default-sql-translate.

Algorithm 1: Refinement based on execution and consistency with MixPrompt

Data: Require: Query questions Q_{test}; Database D_{test}; Prompt design function collections F
Result: SQL output of test set: SQL$_{test}$
while q *in* Q_{test} **do**
  $D_q \leftarrow D_{test}[q]$;
  while f *in* F **do**
    Prompt$_q \leftarrow f(q_i, D_q)$; eq (2)
    M = [];
    O = [];
    while b *in* B **do**
      $sql_q \underset{i.i.d}{\sim}$ LLM(Prompt$_q, r$) ; eq (3)
      $O_q = Exec(sql_q, D_q)$;
      if *"error" NOT in* O_q **then**
        $M \leftarrow sql_q$;
        $O \leftarrow O_q$;
      end
    end
  end
  $sql_{select} = \{sql_q : O_q = Majority(O), q \in M\}$; eq (9)
  SQL$_{test} \leftarrow sql_{select}$
end

prompt Prompt$_q$ to get SQL collections by Eq 3:

$$M_{qf} = \{sql_{qf}^1, ...sql_{qf}^b\}_B \quad (4)$$

We then execute the generated SQLs using an engine $Exec$ (i.e. *sqlite3*), which yields the outputs O as the execution result of SQL on the provided database.

$$O_{qf} = \{O_{qf}^b : O_{qf}^b = Exec(sql_{qf}^b, D_q), sql_q^b \in M_{qf}\} \quad (5)$$

We further exclude outputs O_{qf} that yield errors and only keep the valid output, therefore obtain final (SQL, outcome) pairs for prompt design f: $R_{qf} = (M_{qf}, O_{qf}) = \{(M_{qf}^b, O_{qf}^b) : O_{qf}^b \neq \text{errors}\}$. We repeat the above process for each prompt design function f and generate $R_q = \{R_{q1}, ...R_{qf}, ..\}_{nF}$, by concatenating all the results across multiple designs and obtain:

$$M_q = [M_{q1}, ..., M_{qf}..., M_{nF}] \quad (6)$$
$$Q_q = [O_{q1}, ..., O_{qf}..., O_{nF}] \quad (7)$$

Following self-consistency, we select the SQL that give the execution outcome consisted with the majority of the execution outcomes O_q generated by all M_q.

$$sql_{select} = \{sql_q^k : O_q^k = Majority(O_q) \quad (8)$$
$$O_q^k \in Q_q, sql_q^k \in M_q\}. \quad (9)$$

where k is the index across multiple prompt design and consistency repeats. The overall process is described in Algorithm 1.
With the goal of increasing diversity for better refinement, we further expand our method to not only use one LLM, but rather a mixture of LLMs:

$$p(sql|q) = \sum_{\text{LLM}} \sum_f p(sql|LLM, f, q)p(f)p(LLM) \quad (10)$$

Similar to the combination idea in *MixPrompt*, *"MixPrompt and Model"* combines outputs across multiple LLMs, in addition to across multiple prompt designs.

3 Experiments

Tasks and datasets: We consider the cross-domain large-scale Text-to-SQL benchmark, Spider (Yu et al., 2018) that contains 7000 training samples across 166 databases and 1034 evaluation samples ('Dev split') across 20 databases.

Models: **PaLM FLAN 540B** is a PaLM model variant (Chowdhery et al., 2022) with 540-billion parameters fine-tuned on a collection of tasks phrased as instructions. FLAN (Chung et al., 2022) is a reference to the fine-tuning in a way that respect instructions being given in the prompt. **PaLM-62B** is a PaLM variant with 62 billion parameters trained on 1.3T tokens following the (Hoffmann et al., 2022) **PaLM FLAN 62B** is FLAN fintuned variant. **Quantization** is applied to above models when with q. It reduces precision of a model's parameters and enable efficient inference.
Fine-tuning baselines: **PICARD** (Scholak et al., 2021) employs incremental parsing to constrain auto-regressive decoding. **RASAT** (Qi et al., 2022) is a transformer model that integrates relation-aware self-attention and constrained auto-regressive decoders. **RESDSQL** (Li et al., 2023) decouples schema linking and skeleton parsing using a ranking-enhanced encoding and skeleton-aware decoding framework.
In-context learning baselines: (Rajkumar et al., 2022) comprehensively evaluate the Text-to-SQL ability of CodeX and GPT3, while (Liu et al., 2023) conduct a comprehensive evaluation on ChatGPT.

Evaluation: We consider two commonly-used evaluation metrics: execution accuracy (EX) and test-suite accuracy (TS) (Zhong et al., 2020), where EX measures if SQL execution outcome matches

	Methods	SPIDER	
		EX	**TS**
Fine-tuning	T5-3B + PICARD	79.3	69.4
	RASAT + PICARD	80.5	70.3
	RESDSQL-3B + NatSQL (SOTA)	**84.1**	**73.5**
In-context learning	GPT-3 ada (0-shot)	2.3	0.3
	GPT-3 babbage (0-shot)	5.7	3.9
	GPT-3 curie (0-shot)	12.6	8.3
	GPT-3 davinci (0-shot)	26.3	21.7
	Codex cushman (0-shot)	63.7	53.0
	Codex davinci (0-shot)	67.0	55.1
	ChatGPT (0-shot)	70.1	60.1
	SQLPrompt (0-shot)	**76.6**	**68.0**
	SQLPrompt (4-shot)	**77.1**	**68.6**

Table 1: Performance on the Spider Dev set, measured in execution accuracy (EX) and test-suite accuracy (TS). GPT3 and CodeX results are from (Rajkumar et al., 2022) and ChatGPT results are from (Liu et al., 2023).

Table 2: Ablation study on prompt design approaches in 0-shot setting. MixPrompt improves concise or verbose prompt design approaches with different LLMs. We only mark TS Acc changes, not EX, because TS is more accurate evaluation.

Models	Concise		Verbose		MixPrompt	
	EX	TS	EX	TS	EX	TS
PaLM FLAN 62B q	67.7	61.3	70.8	62.9	70.5	63.2 ($\uparrow$ 0.3)
PaLM FLAN 540B q	72.3	64.1	71.6	61.3	74.0	65.5 ($\uparrow$ 1.4)

Table 3: Ablation Study: Few-shots

Models	Concise		Verbose		MixPrompt	
	EX	TS	EX	TS	EX	TS
PaLM FLAN 62B q	65.9	59.6	71.8	63.8	74.7	66.6 ($\uparrow$ 2.8)
PaLM FLAN 540B q	71.2	63.2	70.7	61.1	74.7	65.2 ($\uparrow$ 2.0)

ground truth. TS assesses each query by running **multiple tests** against randomly generated database with same schema (EX only evaluates on one test). So TS reduces false positives from EX and TS is more accurate evaluation. Here we focus on TS. Exact match evaluation is not performed, as multiple correct SQLs exist for one query.

4 Results

Table-1 presents the comparison between SQLPrompt and the state-of-the-art models for in-context learning and fine-tuning. For in-context learning, SQLPrompt outperforms context learning state-of-the-art (SOTA) ChatGPT (with their recommended prompts) by a large margin: $\uparrow$ 7% for execution accuracy (EX) and $\uparrow$ 8.1% for test suite accuracy (TS). Examples of SQL generated by SQLPrompt is Table 7 in Appendix.

Ablation study SQLPrompt consists of multiple components: prompt design, execution-based consistency decoding, Mix Prompt, and Mix LLMs. To shed light into the impact of these components, we present ablation studies. We first examine prompt designs and *MixPrompt* in zero-shot (Table 2) and few-shots setup (Table 3). We tested it via different LLMs. The results show that MixPrompt improves upon single prompt on both two LLMs tested. We

Table 4: Ablation Study of SQLPrompt (without Mix LLMs) :

	EX	TS
SQLPrompt (Prompt Design + Consistency + Execution Filtering +MixPrompt)	70.5	63.2
No MixPrompt	67.7	61.3 ($\downarrow$ 1.9)
Only Schema (No primary, No foreignkeys, no DB content)	66.4	57.3 ($\downarrow$ 5.9)
No Consistency	55.9	49.6 ($\downarrow$ 13.6)
No Execution Filtering	55.2	48.7 ($\downarrow$ 14.5)

Table 5: Ablation Study: SQLprompt with Mix LLMs

Num of Mixture	Zero-shots		Few-shots	
	2	4	6	16
EX	74	76.6	77.3	77.1
TS	65.5	68.0	68.3	68.6

do not observe improvement from few-shots over zero-shots for better model (i.e. 540B), we hypothesize when model gets larger, LLM's Text-to-SQL ability becomes better, leading to less room to improve. We also provide a different set of few-shots results in Table 6 in Appendix, which yield similar results with Table 3, indicating varying few-shots example with same prompt design may not improve much. Further, with single LLM, Table 4 shows ablation study on each component of *SQL-prompt*. Row 2 is SQLPrompt with PALM FLAN 62B q; Row 3-6 remove only one component. We can see each conpoment contribute positively, especially consistency and execution filtering. The effect of Mix LLMs of *SQLPrompt* shows in Table 5. When the number of mixture is less than 4, we use zero-shot results in Table 2. For example, with 4 mixtures, we combine all the four models in Table 2: PaLM FLAN 62B q: Concise or Verbose prompt design; PaLM FLAN 540B q: Concise or Verbose. When number of mixture is greater than 4, we include few-shots results. Note most of the components in SQLPrompt can be apply to other context learning methods.

Limitations

The limitation of this work is that query multiple prompt designs and/or multiple LLMs can be expensive and time consuming. Although combining multiple prompt designs and LLMs are promising for improving performance, future work can be work on effectively combine them to save cost.

References

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Decoupling the skeleton parsing and schema linking for text-to-sql. *arXiv preprint arXiv:2302.05965*.

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.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983*.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

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

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.

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

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

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*.

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

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

Chunting Zhou, Junxian He, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Prompt consistency for zero-shot task generalization. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2613–2626, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

A Text-to-SQL challenges and prompt design with primary/foreign keys and database content

Fig 2 shows a Text-to-SQL example from Spider Dataset. We use Fig 2 to demonstrate the necessarity of including primary and foreign keys, and content of database. The data schema contains multiple tables. Each table has multiple columns. **Primary keys** are the columns that uniquely identify a row in a table. Primary keys are important, because some columns might specifically be challenging and it might be beneficial to include them specifically as prompts, such as in Query 1 of Fig. 2 where "t2.makeid" may be mistakenly written as "t2.id" without proper emphasis. **Foreign key** is a column or combination of columns that is used to establish and enforce a link between the data in two tables. For example, in Fig 2 , Column Maker of Table Model list is equivalent to Column ID of Car Maker. By including foreign keys into prompt, LLMs can know how to join different tables. Otherwise, it can be ambiguous to link multiple tables, especially for complex data schema or schema with confusing column names. For example, Column Maker in Table Model list is not the same as Column Maker in Table Car Maker. Although they both called column "Maker", one is number and the other is string. Instead due to foreign keys, we known Column Maker of Table Model List is equivalent to Column ID in Table Car maker. Additionally, including relevant **database content value**, as seen in (Xie et al., 2022; Scholak et al., 2021), is necessary as they help identify which columns are relevant to key words in the query question, such as in Fig. 2, Query1's key information is "amc honrnet sportabout (sw)", however, without adding database content value, we do not know which columns contain the value of the key information. e.g. is it Column Maker of Table Model List? Is it Column Maker of Table Car Maker? or Is it Column Make of Table Car Names? Only by including database content values, LLM can know it should use The column of Make of Table Car Names. Note that the database content values are questions depended. Only content values that are related with questions is included into prompt. See Fig 3. Note not all the content values are included. So there is not problem if the number of database contents is very large. As for how to extract relevant database content values regarding the query questions, we follow (Xie et al., 2022; Scholak et al., 2021), where all the content values are compared against the query questions, and only top few ones that match the query question the best are included.

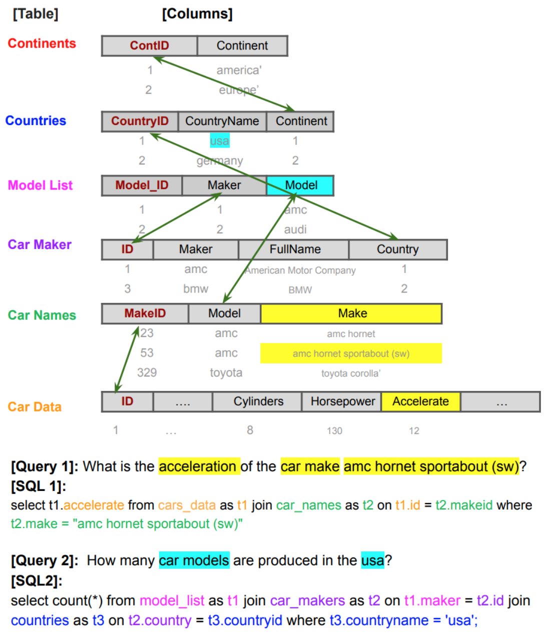

Figure 2: One database schema with two query questions and true SQL as demo. Dark red are primary keys. Dark green arrows are foreign keys joining different tables. Light gray is the context (values) in database (or table). Both primary key and foreign keys are given in the database schema. The highlighted (yellow or cyan) are the part of schema that are used to solve Query 1 and 2 respectively. Colors are simply for easy visualization. Same color, same table.

B Prompt design examples

We show the prompt design for a example in Spider dataset.

B.1 Concise prompt design

"This is a task converting text into SQL statement. We will first given the dataset schema and then ask a question in text. You are asked to generate SQL statement. Here is the test question to be anwered: Convert text to SQL: **[Schema (values)]**: | car_1 | continents : contid , continent | countries : countryid , countryname , continent | car_makers : id , maker (amc) , fullname , country | model_list : modelid , maker , model (amc) | car_names : makeid , model (amc) , make (amc hornet , amc hornet sportabout (sw)) | cars_data : id , mpg , cylinders , edispl , horsepower , weight , accelerate , year; **[Column names (type)]**: continents : contid (number) | continents : continent (text) | countries :

Query 1

continents : contid , continent | **countries** : countryid , countryname , continent | car_makers : id , maker (amc) , fullname , country | **model_list** : modelid , maker , model (amc) | **car_names** : makeid , model (amc) , make (amc hornet , amc hornet sportabout (sw)) | **cars_data** : id , mpg , cylinders , edispl , horsepower , weight , accelerate , year;

Query 2

continents : contid , continent | **countries** : countryid , countryname (usa) , continent | **car_makers** : id , maker , fullname , country | **model_list** : modelid , maker , model | car_names : makeid , model , make | **cars_data** : id , mpg , cylinders , edispl , horsepower , weight , accelerate , year;

Figure 3: **Example of database with content**: examples in Fig 2. Highlighted are database content for different queries. Following previous work (Xie et al., 2022; Scholak et al., 2021), only the relevant database content values are included. So different query questions have different database content value.

countryid (number) | countries : countryname (text) | countries : continent (number) | car_makers : id (number) | car_makers : maker (text) | car_makers : fullname (text) | car_makers : country (text) | model_list : modelid (number) | model_list : maker (number) | model_list : model (text) | car_names : makeid (number) | car_names : model (text) | car_names : make (text) | cars_data : id (number) | cars_data : mpg (text) | cars_data : cylinders (number) | cars_data : edispl (number) | cars_data : horsepower (text) | cars_data : weight (number) | cars_data : accelerate (number) | cars_data : year (number); **[Primary Keys]**: continents : contid | countries : countryid | car_makers : id | model_list : modelid | car_names : makeid | cars_data : id; **[Foreign Keys]**: countries : continent equals continents : contid | car_makers : country equals countries : countryid | model_list : maker equals car_makers : id | car_names : model equals model_list : model | cars_data : id equals car_names : makeid **[Q]**: What is the accelerate of the car make amc hornet sportabout (sw)?; **[SQL]**: "

B.2 Verbose prompt design

"This is a task converting text into SQL statement. We will first given the dataset schema and then ask a question in text. You are asked to generate SQL statement. Here is the test question to be anwered: Let us take a question and turn it into a SQL statement about database tables. There are 6 tables. Their titles are: continents, countries, car_makers, model_list, car_names, cars_data. Table 1 is continents, and its column names and types are: ContId (Type is number), Continent (Type is text). Table 2 is countries, and its column names and types are: CountryId (Type is number), CountryName (Type is text), Continent (Type is number). Table 3 is car_makers, and its column names and types are: Id (Type is number), Maker (Type is text), FullName (Type is text), Country (Type is text). Table 4 is model_list, and its column names and types are: ModelId (Type is number), Maker (Type is number), Model (Type is text). Table 5 is car_names, and its column names and types are: MakeId (Type is number), Model (Type is text), Make (Type is text). Table 6 is cars_data, and its column names and types are: Id (Type is number), MPG (Type is text), Cylinders (Type is number), Edispl (Type is number), Horsepower (Type is text), Weight (Type is number), Accelerate (Type is number), Year (Type is number). The primary keys are: contid from Table continents, countryid from Table countries, id from Table car_makers, modelid from Table model_list, makeid from Table car_names, id from Table cars_data. The foreign keys are: continent from Table countries is equivalent with contid from Table continents, country from Table car_makers is equivalent with countryid from Table countries, maker from Table model_list is equivalent with id from Table car_makers, model from Table car_names is equivalent with model from Table model_list, id from Table cars_data is equivalent with makeid from Table car_names. Use foreign keys to join Tables. Columns with relevant values: Table car_makers Column maker have values: amc; Table model_list Column model have values: amc; Table car_names Column model have values: amc; Table car_names Column make have values: amc hornet, amc hornet sportabout (sw); Only use columns with relevant values to generate SQL. Let us take a text question and turn it into a SQL statement about database tables. The question is: What is the accelerate of the car make amc hornet sportabout (sw)? The corresponding SQL is: "

C ChatGPT's default prompt design

"Complete sqlite SQL query only and with no explanation Sqlite SQL tables, with their properties: continents(ContId, Continent); countries(CountryId, CountryName, Continent); car_makers(Id,Maker, FullName, Country); mode_list(ModelId, Maker, Model); car_names(MakeId, Model, Make); car_data(Id, MPG, Cylinders, Edispl, Horsepower, Weight, Accelerate, Year). What is the accelerate of the car make amc hornet sportabout (sw)? SELECT"

Table 6: Another set of Few-shots

Methods/Datasets	Concise		Verbose		MixPrompt	
	EX	TS	EX	TS	EX	TS
PaLM FLAN 62B q	68.1	61.7	71.9	64.3	74.5	66.7

SQLPrompt output

We show a list of real output generated by $SQLPrompt$. See Table 7

D Selection of few-shot examples

We also provide results in few-shot learning setting with a different set of examples. Compared to Table 3 (eg. 66.6), the two results are very close, indicating changing the few shots examples doesn't improve performance when using the same prompt design.

E Experiment details

Query large language model does not require TPU or GPU. CPU is enough. Consistency repeats are 32. Sampling temperature is 0.5.

Question	What are the names of the singers whose birth years are either 1948 or 1949?
Ground Truth	SELECT Name FROM singer WHERE Birth_Year = 1948 OR Birth_Year = 1949
LLM Generation	SELECT name FROM singer WHERE birth_year = 1948 OR birth_year = 1949
Execution Accuracy	1
Question	What is the name of the singer with the largest net worth?
Ground Truth	SELECT Name FROM singer ORDER BY Net_Worth_Millions DESC LIMIT 1
LLM Generation	SELECT T1.name FROM singer AS T1 ORDER BY T1.net_worth_millions Desc LIMIT 1
Execution Accuracy	1
Question	What is the name of the singer who is worth the most?
Ground Truth	SELECT Name FROM singer ORDER BY Net_Worth_Millions DESC LIMIT 1
LLM Generation	SELECT T1.name FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id ORDER BY T1.net_worth_millions Desc LIMIT 1
Execution Accuracy	1
Question	Show different citizenship of singers and the number of singers of each citizenship.
Ground Truth	SELECT Citizenship , COUNT(*) FROM singer GROUP BY Citizenship
LLM Generation	SELECT T1.citizenship , count(*) FROM singer AS T1 GROUP BY T1.citizenship
Execution Accuracy	1
Question	For each citizenship, how many singers are from that country?
Ground Truth	SELECT Citizenship , COUNT(*) FROM singer GROUP BY Citizenship
LLM Generation	SELECT Citizenship , count(*) FROM singer GROUP BY Citizenship
Execution Accuracy	1
Question	Please show the most common citizenship of singers.
Ground Truth	SELECT Citizenship FROM singer GROUP BY Citizenship ORDER BY COUNT(*) DESC LIMIT 1
LLM Generation	SELECT citizenship FROM singer GROUP BY citizenship ORDER BY count(*) DESC LIMIT 1
Execution Accuracy	1
Question	What is the most common singer citizenship ?
Ground Truth	select citizenship from singer group by citizenship order by count(*) desc limit 1
LLM Generation	SELECT citizenship FROM singer GROUP BY citizenship ORDER BY count(*) DESC LIMIT 1
Execution Accuracy	1
Question	Show different citizenships and the maximum net worth of singers of each citizenship.
Ground Truth	SELECT Citizenship , max(Net_Worth_Millions) FROM singer GROUP BY Citizenship
LLM Generation	SELECT T1.citizenship , max(T1.net_worth_millions) FROM singer AS T1 GROUP BY T1.citizenship
Execution Accuracy	1
Question	For each citizenship, what is the maximum net worth?
Ground Truth	SELECT Citizenship , max(Net_Worth_Millions) FROM singer GROUP BY Citizenship
LLM Generation	SELECT T1.citizenship , max(T1.net_worth_millions) FROM singer AS T1 GROUP BY T1.citizenship
Execution Accuracy	1
Question	Show titles of songs and names of singers.
Ground Truth	SELECT T2.Title , T1.Name FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID
LLM Generation	SELECT T1.title , T2.name FROM song AS T1 JOIN singer AS T2 ON T1.singer_id = T2.singer_id
Execution Accuracy	1
Question	What are the song titles and singer names?
Ground Truth	SELECT T2.Title , T1.Name FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID
LLM Generation	SELECT T1.title , T2.name FROM song AS T1 JOIN singer AS T2 ON T1.singer_id = T2.singer_id
Execution Accuracy	1
Question	Show distinct names of singers that have songs with sales more than 300000.
Ground Truth	SELECT DISTINCT T1.Name FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID WHERE T2.Sales > 300000
LLM Generation	SELECT DISTINCT T1.name FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id WHERE T2.sales > 300000
Execution Accuracy	1
Question	what are the different names of the singers that have sales more than 300000?
Ground Truth	SELECT DISTINCT T1.Name FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID WHERE T2.Sales > 300000
LLM Generation	SELECT T1.name FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id WHERE T2.sales > 300000
Execution Accuracy	1
Question	Show the names of singers that have more than one song.
Ground Truth	SELECT T1.Name FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID GROUP BY T1.Name HAVING COUNT(*) > 1
LLM Generation	SELECT T1.name FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id GROUP BY T1.singer_id HAVING COUNT(*) > 1
Execution Accuracy	1
Question	What are the names of the singers that have more than one songs?
Ground Truth	SELECT T1.Name FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID GROUP BY T1.Name HAVING COUNT(*) > 1
LLM Generation	SELECT T1.name FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id GROUP BY T1.singer_id HAVING COUNT(*) > 1
Execution Accuracy	1
Question	Show the names of singers and the total sales of their songs.
Ground Truth	SELECT T1.Name , sum(T2.Sales) FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID GROUP BY T1.Name
LLM Generation	SELECT T1.name , sum(T2.sales) FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id GROUP BY T1.name
Execution Accuracy	1
Question	For each singer name, what is the total sales for their songs?
Ground Truth	SELECT T1.Name , sum(T2.Sales) FROM singer AS T1 JOIN song AS T2 ON T1.Singer_ID = T2.Singer_ID GROUP BY T1.Name
LLM Generation	SELECT T1.name , sum(T2.sales) FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id GROUP BY T1.name
Execution Accuracy	1
Question	List the name of singers that do not have any song.
Ground Truth	SELECT Name FROM singer WHERE Singer_ID NOT IN (SELECT Singer_ID FROM song)
LLM Generation	SELECT name FROM singer WHERE singer_id NOT IN (SELECT singer_id FROM song)
Execution Accuracy	1
Question	What is the sname of every sing that does not have any song?
Ground Truth	SELECT Name FROM singer WHERE Singer_ID NOT IN (SELECT Singer_ID FROM song)
LLM Generation	SELECT T1.name FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id WHERE T2.singer_id IS NULL
Execution Accuracy	0
Question	Show the citizenship shared by singers with birth year before 1945 and after 1955.
Ground Truth	SELECT Citizenship FROM singer WHERE Birth_Year < 1945 INTERSECT SELECT Citizenship FROM singer WHERE Birth_Year > 1955
LLM Generation	SELECT T1.citizenship FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id WHERE T1.birth_year < 1945 INTERSECT SELECT T1.citizenship FROM singer AS T1 JOIN song AS T2 ON T1.singer_id = T2.singer_id WHERE T1.birth_year > 1955
Execution Accuracy	0
Question	What are the citizenships that are shared by singers with a birth year before 1945 and after 1955?
Ground Truth	SELECT Citizenship FROM singer WHERE Birth_Year < 1945 INTERSECT SELECT Citizenship FROM singer WHERE Birth_Year > 1955
LLM Generation	SELECT citizenship FROM singer WHERE birth_year < 1945 INTERSECT SELECT citizenship FROM singer WHERE birth_year > 1955
Execution Accuracy	1
Question	How many available features are there in total?
Ground Truth	SELECT count(*) FROM Other_Available_Features
LLM Generation	SELECT count(*) FROM other_available_features
Execution Accuracy	1
Question	What is the feature type name of feature AirCon?
Ground Truth	SELECT T2.feature_type_name FROM Other_Available_Features AS T1 JOIN Ref_Feature_Types AS T2 ON T1.feature_type_code = T2.feature_type_code WHERE T1.feature_name = "AirCon"
LLM Generation	SELECT T1.feature_type_name FROM ref_feature_types AS T1 JOIN other_available_features AS T2 ON T1.feature_type_code = T2.feature_type_code WHERE T2.feature_name = "AirCon"
Execution Accuracy	1
Question	Show the property type descriptions of properties belonging to that code.
Ground Truth	SELECT T2.property_type_description FROM Properties AS T1 JOIN Ref_Property_Types AS T2 ON T1.property_type_code = T2.property_type_code GROUP BY T1.property_type_code
LLM Generation	SELECT property_type_description FROM ref_property_types WHERE property_type_code IN (SELECT property_type_code FROM properties)
Execution Accuracy	1
Question	What are the names of properties that are either houses or apartments with more than 1 room?
Ground Truth	SELECT property_name FROM Properties WHERE property_type_code = "House" UNION SELECT property_name FROM Properties WHERE property_type_code = "Apartment" AND room_count > 1
LLM Generation	SELECT property_name FROM properties WHERE property_type_code = 'House' OR property_type_code = 'Apartment' AND room_count > 1
Execution Accuracy	1

Table 7: SQLPrompt: randomly sampled generated SQL from LLM