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

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 ChatGPT1 (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 finetuned 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 mixture 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.

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 $T_i$: $S = \{T_1, T_2, ... T_i\}$. Each table $T_i$ has table name $N_i$, 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

1https://chat.openai.com/chat.
most popular prompt design approaches only include database schema, 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 linearize 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.

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.

Suppose $F = \{f_1, f_2, \ldots\}$ 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:

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

where $p(f)$ is mixing coefficient. Since we evenly mixing prompts, $p(f) = 1/nF$ where $nF$ is number of design functions. $p(sql|LLM, f, q)$ is sampling probability of generating sql.

MixPrompt 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:

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

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

We sample $B$ times from the LLM with the same
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.

$$O_q = \{O_{qf}^b : O_{qf}^b = \text{Exec}(sql_{qf}^b, D_q), sql_{qf}^b \in M_q\}$$

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 $R_q = \{R_{q1}, ..., R_{qf}, ..., M_{NF}\}$, by concatenating all the results across multiple designs and obtain:

$$M_q = [M_{q1}, ..., M_{qf}, ..., M_{NF}]$$

$$Q_q = [O_{q1}, ..., O_{qf}, ..., O_{NF}]$$

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 $M_q$.

$$sql_{select} = \{sql_{qk}^b : O_{qk}^b = \text{Majority}(O_q)$$

$$O_{qk}^b \in Q_q, sql_{qk}^b \in M_q\}$$. }

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:

$$p(sq|q) = \sum_{LLM} \sum_f p(sq|LLM, f, q)p(f)p(LLM)$$

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
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>
<thead>
<tr>
<th>Methods</th>
<th>SPIDER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EX</td>
</tr>
<tr>
<td>Fine-tuning</td>
<td></td>
</tr>
<tr>
<td>TS-3B + PICARD</td>
<td>79.3</td>
</tr>
<tr>
<td>RASAT + PICARD</td>
<td>80.5</td>
</tr>
<tr>
<td>RESDSQL-3B + NatSQL (SOTA)</td>
<td>84.1</td>
</tr>
<tr>
<td>In-context learning</td>
<td></td>
</tr>
<tr>
<td>GPT-3 ada (0-shot)</td>
<td>2.3</td>
</tr>
<tr>
<td>GPT-3 babbage (0-shot)</td>
<td>5.7</td>
</tr>
<tr>
<td>GPT-3 curie (0-shot)</td>
<td>12.6</td>
</tr>
<tr>
<td>GPT-3 davinci (0-shot)</td>
<td>26.3</td>
</tr>
<tr>
<td>Codex cushman (0-shot)</td>
<td>63.7</td>
</tr>
<tr>
<td>Codex davinci (0-shot)</td>
<td>67.0</td>
</tr>
<tr>
<td>ChatGPT (0-shot)</td>
<td>70.1</td>
</tr>
<tr>
<td>SQLPrompt (0-shot)</td>
<td>76.6</td>
</tr>
<tr>
<td>SQLPrompt (4-shot)</td>
<td>77.1</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Models</th>
<th>Concise</th>
<th>Verbose</th>
<th>MixPrompt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EX</td>
<td>TS</td>
<td>EX</td>
</tr>
<tr>
<td>PaLM FLAN 62B q</td>
<td>67.7</td>
<td>61.3</td>
<td>70.8</td>
</tr>
<tr>
<td>PaLM FLAN 540B q</td>
<td>72.3</td>
<td>64.1</td>
<td>71.6</td>
</tr>
</tbody>
</table>

Table 3: Ablation Study: Few-shots

<table>
<thead>
<tr>
<th>Models</th>
<th>Concise</th>
<th>Verbose</th>
<th>MixPrompt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EX</td>
<td>TS</td>
<td>EX</td>
</tr>
<tr>
<td>PaLM FLAN 62B q</td>
<td>65.9</td>
<td>59.6</td>
<td>71.8</td>
</tr>
<tr>
<td>PaLM FLAN 540B q</td>
<td>71.2</td>
<td>63.2</td>
<td>70.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SQLPrompt (Prompt Design)</th>
<th>EX</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Consistency</td>
<td>70.5</td>
<td>63.2</td>
</tr>
<tr>
<td>+ Execution Filtering</td>
<td>66.4</td>
<td>57.3 (↓ 5.9)</td>
</tr>
<tr>
<td>No MixPrompt</td>
<td>55.9</td>
<td>49.0 (↓ 14.9)</td>
</tr>
<tr>
<td>No Execution Filtering</td>
<td>55.2</td>
<td>48.7 (↓ 12.5)</td>
</tr>
</tbody>
</table>

Table 5: Ablation Study: SQLPrompt with Mix LLMs

<table>
<thead>
<tr>
<th>Num of Mixture</th>
<th>Zero-shots</th>
<th>Few-shots</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX</td>
<td>74</td>
<td>76.6</td>
</tr>
<tr>
<td>TS</td>
<td>65.5</td>
<td>68.0</td>
</tr>
</tbody>
</table>

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: ↑ 7% for execution accuracy (EX) and ↑ 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 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 SQLprompt. Row 2 is SQLPrompt with PALM FLAN 62B q; Row 3-6 remove only one component. We can see each component 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


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 necessity 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 hornet 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, LLMs 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.

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 | model_list : modelid , maker , model ( amc ) | cars_data : id , mpg , cylinders , edispl , horsepower , weight , accelerate , year; [Column names (type)]: continents : contid , continent | countries : countryid , countryname | car_makers : id , maker ( amc ) , fullname | model_list : modelid , maker , model ( amc ) | 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 : names (text) | car_names : model (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;


What is the accelerate of the car make amc hornet sportabout (sw)?

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 text question to be answered: 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 modelid from Table cars_data, 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 make have values: amc; Table cars_data Column cars_data 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:

"
make amc hornet sportabout (sw)? SELECT"

<table>
<thead>
<tr>
<th>Methods/Datasets</th>
<th>Concise EX</th>
<th>Concise TS</th>
<th>Verbose EX</th>
<th>Verbose TS</th>
<th>MixPrompt EX</th>
<th>MixPrompt TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PaLM FLAN 62B q</td>
<td>68.1</td>
<td>61.7</td>
<td>71.9</td>
<td>64.3</td>
<td>74.5</td>
<td>66.7</td>
</tr>
</tbody>
</table>

**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.