题目

摘要

数据库表是存储大型结构化数据的一种普遍方法，为了方便人们使用数据库，越来越多非数据库技术人员开始关注如何将自然语言指令转变为可执行的 SQL，即文本到 SQL 解析任务。现在，大语言模已成为文本到 SQL 解析任务的新范例，尤其是提示词工程在该任务的运用方式成为了研究的焦点，概括能分为三个任务：问题表示方式，示例选择方式，示例信息组织方法。阿里巴巴提出的 DAIL-SQL 在后二者做出了创新，在 Spider 和 BIRD 数据集上取得了不错结果，但是忽略了问题表示方式。所以本文提出了新的问题表示方式，引入了更多数据库信息如列数据类型，列数据描述信息，列数据枚举值，主键和外键信息。利用 DAIL-SQL 并结合新的问题表示形式在 Spider 和 BIRD 数据集上取得了更好的结果。

关键词： test-tp-sql；大语言模型；SQL

1 引言

表已经成为存储来自各种源的大型结构数据的首选方法。它们提供了类似网格的行和列格式，以便于数据分析。熟练的专业人员可以使用 SQL 语句有效地访问这些表，然而现在仅仅使用自然语言接口完成文本到 SQL 任务就能允许非技术用户访问关系数据。将自然语言问题转换为机器可执行 SQL 查询的文本到 SQL 解析任务已经引起了业界和学术界的关注。它允许非专业用户查询表格，并在客户服务、问题回答和机器人导航等应用程序中发挥关键作用。

Text-to-SQL 领域以前的工作 [10, 11, 25] 大多数关注如何提取问题到 SQL 的模板，并利用 Test-to-SQL 的语料训练一个编码器-解码器模型产生这些模板。在最近一年里，大语言模型已经成为了 Text-to-SQL 任务的新范例 [12, 21, 24]。与以往的研究不同，基于大语言模型的 Text-to-SQL 任务的核心问题是如何提示大语言模型生成正确的 SQL 查询，即提示工程。提示工程在基于大语言模型的 Text-to-SQL 任务运用能概括为三个任务：问题表示方式 [3, 5, 19, 20]，样例选择方式 [6, 14, 16]，样例信息组织方法 [6]

2 相关工作

Text-to-SQL 是一项涉及到将自然语言问题自动转换为 SQL 查询的任务。它在弥补非专家用户和数据库系统之间的差距、提高数据处理效率以及实现智能数据库服务和自动化数据

分析等各种应用方面发挥着至关重要的作用。然而，由于理解自然语言问题和准确生成 SQL 查询的复杂性，Text-to-SQL 仍然是一个具有挑战性的任务。

早期的 Text-to-SQL 系统使用基于规则和基于模板的方法 [9,15]，这适合于简单的用户查询和数据库，但需要大量的人类工程和用户交互，而且事先为不同的场景或域设计 SQL 模板也十分困难。近年来，数据库和自然语言处理领域的进步促进了在基准和方法方面探索更复杂的 Text-to-SQL 研究环境。为了弥补研究和实际部署之间的差距，引入了一些大规模的基准数据集，如 WikiSQL [29]，KaggleDBQA [8]，Spider [27]，BIRD [1]。在方法论层面，最近的研究已经把 Text-to-SQL 作为一个序列到序列的任务来处理，并使用编码器-解码器结构来训练机器学习模型 [2]，深度学习技术包括注意机制 [13] ，图表示法 [11,25,26]，语法分析 [7,10] 已被广泛用于提高 Text-to-SQL 方法的性能。

最近，随着大型语言模型 (LLM) 的出现，在自然语言处理和人工智能方面有了显著的进步，比如 OpenAI 的 GPT [17, 18] 和 Meta 的 Llama [22, 23] 这些模型在理解和生成人类语言等任务上取得了重大进展。LLM 不同于一般的机器学习模型，因为它们是在大量文本数据集上预先训练的，这使它们能够普遍在与语言相关任务中表现出色。Text-to-SQL 任务也不例外，LLM 有潜力弥补自然语言查询和结构化 SQL 查询之间的差距。在 Text-to-SQL 任务中使用大型语言模型 (LLM) 有两种主要技术，第一种是 **监督微调**，其中使用了额外的 Text-to-SQL 实例来进一步优化 LLM，以便在特定的下游任务中获得更好的性能。第二种技术是 **in-context learning** [4]，也被称为提示工程学 [12,16] 或 few-shot learning。这种方法侧重于在只有有限的标记数据可用时，使 LLM 适应特定的下游任务。在 Test-to-SQL 问题的背景下，研究人员提出了构造提示符的各种方法。具体而言，Test-to-SQL 的提示构造方法包括问题表示方法 [3,5,19,20]，示例选择 [6,14,16] ，示例信息组织 [6]。这些技术旨在通过仔细设计和显示相关提示来提高 LLM 生成正确 SQL 查询的性能。

3 本文方法

3.1 本文方法概述

目前缺乏对基于大型语言模型的 Text-to-SQL 解决方案进行系统评估的研究，所选文章提出了一种名为 DAIL-SQL 的集成解决方案，用于解决基于大型语言模型的 Text-to-SQL 任务。现有研究主要关注 OpenAI 的语言模型 [18]，但调用其 API 昂贵、耗时且受限于速率限制，特别是对于包含多个示例的上下文学习提示。如何高效地进行 Prompt 工程仍然是一个具有挑战性的问题。所选文章通过对问题表示、示例选择和示例组织等方面进行系统研究，旨在设计出高效、有效和经济的基于大型语言模型的 Text-to-SQL 解决方案 DAIL-SQL。

3.2 问题表示

在文章中研究人员评估了几种问题表示方法，并对它们进行了比较和分析。这些问题表示方法是用于将自然语言问题转化为 SQL 查询的提示，对于大型语言模型（LLMs）在 Text-to-SQL 任务中的正确生成 SQL 查询非常重要。针对零样本场景下，考虑某个数据库 $\mathcal{D}$ 上自然语言的目标问题 q, 问题表示的目标最大化 LLM $\mathcal{M}$, 生成正确 SQL 语句 s^* 的可能性，如

Listing 1: Example of Code Representation Prompt

```
/* Given the following database schema: */
CREATE TABLE continents(
ContId int primary key ,
Continent text ,
foreign key(ContId) references countries(Continent)
);
CREATE TABLE countries(
CountryId int primary key ,
CountryName text , 11 Continent int ,
foreign key ( Continent ) references continents ( ContId )
) ;

/* Answer the following : How many continents are there ?   */
SELECT
```

以下表示:

$$\max_{\sigma} \quad \mathbb{P}_{\mathcal{M}}(s^*|\sigma(q,\mathcal{D})), \tag{1}$$

其中函数 $\sigma(\cdot,\cdot)$ 利用数据库 $\mathcal{D}$ 的模式中有用的信息确决定目标问题 q 的表示。此外，$\sigma(\cdot,\cdot)$ 还可以包括指令语句、规则含义和外键等信息，附录中列举几种常用的问题表示形式如列表 6 7 8 1 9所示。其中列表 1，CR_P 代码表示形式，以 SQL 语法显示文本到 SQL 任务，它直接显示 “CREAT TABLE” SQL，并在注释中用自然语言问题提示 LLM。相比与其他表示形式相比，CR_P 因其提供创建数据库所需的全面信息，例如列类型和主键/外键在 SPIDER 数据集上取得了更好的预测结果，所复现文章也是用 DAIL-SQL 作为其问题表示方法。

3.3 上下文学习

在本节中，文章探讨了在 Text-to-SQL 任务中进行上下文学习的方法和效果。所谓上下文学习，即在给定一个问题和数据库的情况下，通过使用上下文信息来生成 SQL 语句。传统的 Text-to-SQL 方法通常仅考虑单个问题和数据库之间的映射，而上下文学习则通过考虑问题的前后文以及与之相关的数据库内容，从而更好地理解问题和生成准确的 SQL 查询。在 Text-to-SQL 中，给定一组三元组 $Q = \{(q_i, s_i, \mathcal{D}_i)\}$，自然语言问题及其对应的 SQL 查询在数据库 $\mathcal{D}$ 中的位置和位置，Textto-SQL 在上下文中学习的目标是最大限度地提高 LLM 在目标问题 q 和数据库 $\mathcal{D}$ 上生成正确 SQL 的可能性，如下所示:

$$\begin{aligned} \max_{Q',\sigma} \quad & \mathbb{P}_{\mathcal{M}}(s^*|\sigma(q,\mathcal{D},Q')), \\ \text{s.t.} \quad & |Q'| = k \quad \text{and} \quad Q' \subset Q, \end{aligned} \tag{2}$$

其中函数 $\sigma(\cdot,\cdot)$ 决定目标问题的表示形式，从数据库 $\mathcal{D}$ 的模式中获得有用的信息，并从 $\mathcal{Q}$ 中选取例子。

Text-to-SQL 的上下文学习包括选择最有用的示例 $\mathcal{Q}'$，并决定如何将这些示例的信息组织成提示。接下来我们讨论这两个子任务: 示例选择和示例组织。

3.3.1 示例选择

本节探讨了 Text-to-SQL 任务中的示例选择策略。示例选择是指从候选集中选择与目标问题最相似的示例，以帮助大型语言模型（LLM）学习生成正确的 SQL 查询。首先，常见的示例选择策略包括：

1. **随机选择**: 这个策略从可用的候选集中随机抽取 k 个示例

2. **问题相似度选择** ($\mathcal{QTS_S}$): $\mathcal{QTS_S}$ 选择问题最相似的例子。具体来说，它将示例问题和目标问题都嵌入一个预先训练好的语言模型中。然后它对每个例子-目标对应用一个预定义的距离度量，比如欧几里得度量或负余弦距离。最后利用神经网络算法从 $\mathcal{Q}$ 中选取与目标问题最为匹配的实例。

3. **屏蔽问题相似度选择** ($\mathcal{MQS_S}$): 对于跨域的 Text-to-SQL 任务，$\mathcal{MQS_S}$ 通过使用掩码标记替换所有问题中的表名、列名和值来消除领域特定信息的负面影响，然后使用计算 KNN 算法它们的嵌入相似性。

4. **查询相似度选择** ($\mathcal{QRS_S}$): $\mathcal{QRS_S}$ 不使用目标问题 q，而是旨在选择与目标 SQL 查询 s^* 相似的 k 个示例。具体地说，它使用了一个初步的模型结果问题 q 和数据库 $\mathcal{D}$ 来使用目标生成 SQL 查询 $s^{'}$，其中生成的 $s^{'}$ “可以看作是近似的 s^*。然后根据实例的关键字将查询编码为二进制离散语法向量。然后，通过考虑近似查询的相似性和所选范例的多样性来选择范例。

受 ($\mathcal{MQS_S}$) 和 $\mathcal{QRS_S}$ 的启发，文章提出了 **DAIL SELECTION**$\mathcal{DAIL_S}$，同时考虑问题和查询来选择示例。具体来说，DAIL SELECTION 首先屏蔽候选集 $\mathcal{D}$ 中目标问题 q 和示例问题 q_i 中的特定领域单词。然后，它根据屏蔽 q 和 q_i 的嵌入之间的欧几里得距离对候选示例进行排名。同时，它计算预先预测的 SQL 查询 $s^{'}$ 和 $\mathcal{Q}$ 中的 q_i 之间的查询相似度。最后，选择标准根据查询相似度大于预定义阈值 T 的查询相似度对排序候选进行优先级排序。这样，所选择的前 k 个例子与问题和查询都具有良好的相似性。

3.3.2 示例信息组织

示例组织涉及如何呈现示例的信息，以引导 LLM 生成正确的 SQL 查询。首先，大多数先前的研究在示例组织中使用了包含指令、模式、问题和正确的 SQL 查询的完整信息。这种方法可以提供更多的上下文和背景知识，但也增加了示例的长度和复杂性。其次，Guo 等人提出了一种仅保留所选示例中的 SQL 查询的方法。通过只保留 SQL 查询，可以减少示例的长度，并且可以更有效地指导 LLM 生成 SQL 查询。然而，这种方法可能会导致上下文的缺失，从而影响 LLM 的性能。为了进一步探索示例组织，文章提出了一种将示例表示为问题-SQL 对的选项。这种方法将示例的问题和对应的 SQL 查询配对 **DAIL Organization**($DAIL_O$) 2，以帮助 LLM 理解问题和查询之间的映射关系。这种组织策略可以提供更直观的示例表示，并且可以更好地指导 LLM 生成正确的 SQL 查询。$\$TARGET_QUESTION$ 表示列表 1问题表示形式。

Listing 2: Example of Alpaca SFT Prompt

```
/* Some example questions and corresponding SQL queries  are provided based on
   similar problems: */
/* Answer the following: How many authors are there? */
SELECT count(*) FROM authors

/* Answer the following: How many farms are there?. */
SELECT count(*) FROM farm

\${TARGET_QUESTION}
```

4 复现细节

4.1 创新点

在源码改进部分，本文在源码的基础上主要改进有四点：(1)**OPENSQL-PROMPT** 问题表示设计 (2) **BIRD** 在源代码只有 SPIDER 数据的处理代码的基础上增加了 BIRD 数据集处理 (3) **并行访问 OPENAI API**, 在源代码中提供的 OPENAI 接口访问只能使用单一 api-key 值进行访问，速度慢，效率低下，本文增加了并行访问代码，提高实验效率 (4) **错误分析**，本文中在测试完 BIRD 数据后，增加额外代码分析模型生成的 SQL 错误类型分布。

4.2 与已有开源代码对比

4.2.1 OPEN-PROMPT

为了提高 SQL 生成过程中对各种外部知识的有效利用，本文引入了一种创新的问题表示机制 **OPENSQL-PROMPT**,详见列表 **??**。此外,本文引入 **EXTERNAL-KNOWLEDGE** 来封装相关的外部信息，并使用 **TARGET-QUESTION** 来识别所考虑的特定问题。在我们最初的实验结果中，我们强调了理解数据库定义的重要性。为了增强为模式链接提供的多样化信息，我们设计 **openschema** 来更好地连接 LLM 和数据库，提供了数据的全面而细致的表示。DAIL-SQL 代码如列表 **??**所示,**DATABASE-SCHEMA** 的详细定义如下

- **Database definition**: 相关表的定义是按顺序组织的，每一行表示一个单独的表。随后是外键定义，其中每一行表示一个键映射。
- **Table definition**: 表的定义以表名开始，后跟括号中的所有列定义。这些列是按顺序组织的，每一行表示一个单独的列。
- **Column definition**: 列的定义以列名开头，后面跟着 “:”，还有四个可选元素:
  - **TYPE**: 指示列类型。在数据库定义中，对应的值包括 “text”、“int”、“date”、“datetime”、“real” 和 “varchar”，它们与 SQLite 中的定义保持一致。
  - **DESCRIPTION**: 包含详细的列信息 (如果有的话)

Listing 3: Example of , where **DATABASE_SCHEMA** symbolizes a database schema, **DATABASE_DEFN** symbolizes a database definition according to **DATABASE_SCHEMA**, **EXTERNAL_KNOWLEDGE** represents pertinent external information, and **TARGET_QUESTION** denotes the specific question in focus.

```
### Complete sqlite SQL query only and with no explanation
### SQLite SQL tables are requested to be represented in the following schema.
${DATABASE_SCHEMA}
### Here are SQLite SQL tables, with their properties:
${DATABASE_DEFN}
### Question: ${TARGET_QUESTION}.
### Note that: ${EXTERNAL_KNOWLEDGE}.
SELECT
```

Listing 4: Definition of full database schema **DATABASE_SCHEMA**.

```
# TABLE_NAME (
#   COLUMN_NAME:  TYPE, DESCRIPTION, VALUES, PRIMARY_KEY
#   ...
# )
# ...
# FOREIGN KEYS:
# TABLE_NAME1.COLUMN_NAME1=TABLE_NAME2.COLUMN_NAME2
# ...
```

- **VALUES**: 枚举具有最多五个不同类别的列的分类值，格式为 “(value1，value2，...)”，以便清晰和简洁。
- **PRIMARY_KEY**: 指示此列是否用作主键。

4.2.2 BIRD

由于 BIRD 数据集和 Spider 数据集结果并不一致，为了方便起见，本文现将 BIRD 数据集转变为了 Spider 数据集类似的格式。但由于 BIRD 数据集相比与 Spider 数据集引入了额外的 evidience 字段以及列表描述信息字段，文本在预处理的部分更改了 DataSet 类和数据库模式处理更加简单获取目标问题对应的数据库信息如列表5:

4.2.3 并行访问 OPENAI API

源码中单单只使用了一个 *key* 对 *openaiapi*，但由于 OPENAI 官网有每个 key 每 3 分钟允许访问一次，且每天至多访问 200 次的限制，而 Spider 数据集和 BIRD 数据的测试集大小分别为 1134 和 1567 条数据，单单使用一个 key 测完两个测试数据实验周期过长。所以本文设计了 Key 的轮询机制，使用多个 key 访问 api 接口，避免了单个 key 的限制，极大提高了实验效率。代码如列表10

Listing 5: BIRD: precoss data schema

```
def get_schema_info(self, db_id):
        table_data = json.load(open(self.schema_json))[db_id]
        schema_info = "#\n"
        table_format = "# {table_name}(\n{columns_info}# )\n"
        foreign_keys =  self.foreign_keys[db_id]
        for key, value in table_data.items():
            columns_info = ""
            if key == "foreign_keys":
                continue
            for col, col_value in value.items():
                columns_info += "#    " + col_value + "\n"
            schema_info += table_format.format(table_name=key,
               columns_info=columns_info)
        foreign_keys_set = set()
        foreign_keys_info = "# FOREIGN KEYS:\n"
        for key, value in foreign_keys.items():
            if value not in foreign_keys_set or key not in
               foreign_keys_set:
                foreign_keys_set.add(key)
                foreign_keys_info += "#    " +  key + " = " + value
                   + "\n"
                foreign_keys_set.add(value)
        schema_info = schema_info + foreign_keys_info + "#"
        return schema_info
```

4.2.4 错误分析

本在 DAIL-SQL 测试出 Spider 集和 BIRD 集 EX 指标之后，希望获知 LLM 在 Text-to-SQL 任务上哪方面比较薄弱，为未来的工作做出进一步决定。下面是本文总结的错误类型:

1. **错误的模式链接**: 查询失败次数最多的类别包括模型难以识别列名、表名和语句的实例，从而导致 SQL 语句不准确。这突出显示了 chatgpt 对于 Text-to-SQL 任务的性能的一个显著限制。例如，在查询 “2014 年有多少用户收到评论员徽章?” 其中数据库模式指定了一个 “徽章” 表，模型错误地引用了一个不存在的 “用户” 表。

2. **联表**: 此类别包含需要 JOIN 操作的查询。但是，模型在准确识别所有必需的表或确定建立这些表之间连接的正确外键方面遇到了挑战。模型通常无法确定是否需要根据问题联接多个表以获得正确的结果。或者，在关联两个表时，模型可能不会使用正确的外键执行表联接。

3. **嵌套**: 在这个类别中，目标查询语句 s^* 使用嵌套或设置操作，但是模型未能准确地识别嵌套结构，或者未能检测到正确的嵌套或设置操作。

4. **其他**: 这个类别包含了与之前定义的类别不一致的案例。它特色 SQL 查询与各种问题: 语法错误和不正确的 GROUPBY

4.3 实验环境搭建

本文使用的 python 环境为 Python 3.11.5，利用的 gpt-3.5-turbo 版本为 gpt-3.5-turbo-0613.

5 实验结果分析

5.1 实验设置

数据集. 我们不仅使用广泛使用的 Spider 数据集 [27]，还使用 BIRD 数据集 [1] 进行 Text-to-SQL 方法的评估。但与 Spider 数据集相比，BIRD 数据集为全面评估提供了更具挑战性的文本到 SQL 任务。我们的评估特别使用了开发集，表示为 *BIRD-dev*，因为测试集还没有公开可用。在涉及少数镜头场景的实例中，我们利用 BIRD 的训练集来作为示例候选集。
指标. 为了保证公平的比较, 本文采用前人的指标 [28] 执行精度 (**EX**) 评估预测的 SQL 查询和跨各种数据库实例的地面真相 SQL 查询之间的执行输出的一致性。考虑到对于给定的问题可能有多个有效的 SQL 查询，这个度量提供了对模型性能的更精确的评估。

5.2 消融实验

本文在 BIRD 数据上，本文使用了 GPT-3.5-TURBO 进行了 OPENSQL-Prompt 的各个元素的消融实验。注意，随着可选列元素的增加，表定义将提供更多信息。因此，我们比较以下 9 种类型的列定义，每种定义都包含各种可选的列元素:

- F: With none of the optional column elements.
- FT: With **TYPE**.
- FD: With **DESCRIPTION**.
- FV: With **VALUES**.
- FP: With **PRIMARY_KEY**.
- FVD: With **DESCRIPTION** and **VALUES**.
- $FVDT$: With **DESCRIPTION**, **VALUES**, and **TYPE**.
- $FVDT$: With **DESCRIPTION**, **VALUES**, and **TYPE**.
- $FVDP$: With **DESCRIPTION**, **VALUES**, and **PRIMARY_KEY**.

- *FVDPT*: With **TYPE**, **DESCRIPTION**, **VALUES**, and **PRIMARY_KEY**.

在图 1中，我们提供了跨 BIRD-dev 上的各种表模式的 GPT-3.5-TURBO 上各个元素的比较分析。检查 GPT-3.5-TURBO 在各种列定义中的性能 (图 1)，我们注意到合并列值具有最实质性的积极影响，并且添加列描述也有助于性能提高。相反，包括数据类型和主键似乎会产生负面影响，特别是当只引入一个可选列时。将列值和描述结合在一起可以进一步提高性能，而列类型和主键的引入可以提高性能，其中前者对性能的提高最为显著。但是，与仅包含列值和描述相比，引入所有四个列元素会导致性能下降。

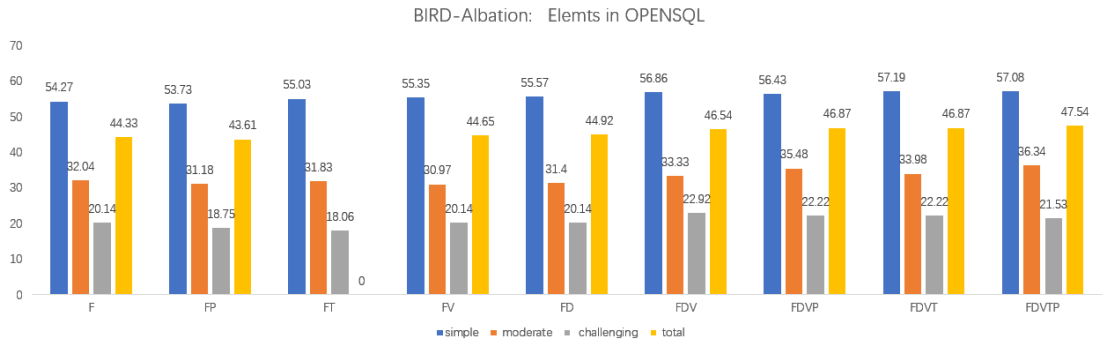

图 1. OPENSQL Prompt 各个元素消融实验结果

5.3 复现实验

在本节中，我们分别利用源码在 GPT-3.5-TURBO 上复现了 DAIL-SQL 实验，值得注意的是，DAIL-SQL 首先利用 $\mathcal{MQS}_{\mathcal{S}}$ 方法先生成 $\mathcal{DAIL}_{\mathcal{S}}$ 所需要的 s'，然后再用 s' 进行 DAIL-SELECTION。首先，在 Spder 数据集复现结果如图 2。可以看到我们设计的 OPENSQL-prompt 在引入更多的数据库信息量后比 $\mathcal{DAIL}_{\mathcal{S}}$ 效果更好，总体上有 0.02% 的提高。

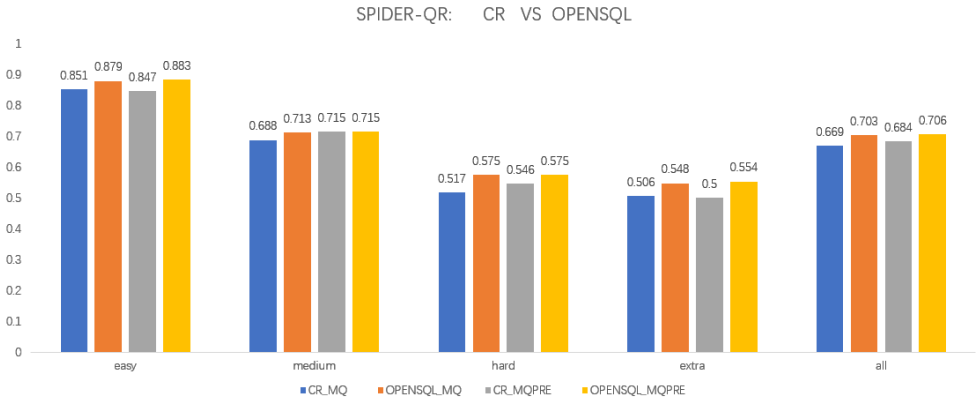

图 2. $\mathcal{DAIL}_{\mathcal{S}}$ vs OPENSQL Prompt 在 Spider 实验结果

但是在针对更难的数据集 BIRD 时，实验结果如图, DAIL-SQL 方法并不理想，在利用

$\mathcal{MQS}_\mathcal{S}$ 在 BIRD 集到达 46.15% 的情况下，再次利用 $\mathcal{DAIL}_\mathcal{S}$ 效果反而下降为了 44.85%，本文推测是 $\mathcal{MQS}_\mathcal{S}$ 在 BIRD 集上表现不佳造成，大量预测错误的 SQL 在示例选择阶段给 $\mathcal{DAIL}_\mathcal{S}$ 带来了更多的误差。而我们的 OPENSQL-Prompt 方法依然在 BIRD 数据集上领先 $\mathcal{DAIL}_\mathcal{S}$，但是值得注意的是通过 OPENSQL-Prompt 方法零样本学习在 BIRD 已经达到了 47.54%, 而在加入上下文学习后效果只有 47.59%，并没有太大提升，猜测是 BIRD 的训练集和测试集在数据分布上有较大的差异，导致 BIRD 的训练集作为示例选择的候选集效果不佳。

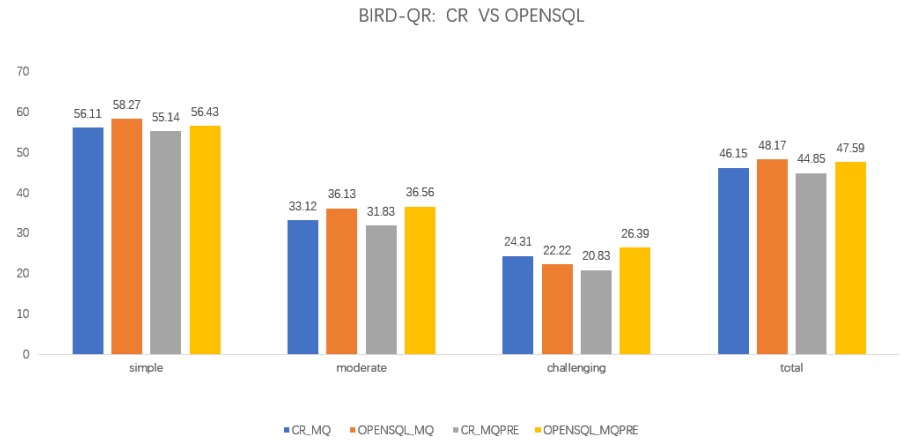

图 3. $\mathcal{DAIL}_\mathcal{S}$ vs OPENSQL Prompt 在 BIRD 实验结果

5.4 错误分析

在本节中，我们分析$\mathcal{DAIL}_\mathcal{S}$ 和 OPEN Prompt 在 BIRD 数据集上的错误数据类型，找到 OPEN Prompt 的具体提高点和潜在的改进点，实验结果如图 4。从图中我们可以看到在错误类型 **Wrong schema** 上，OPEN Prompt 比 $\mathcal{DAIL}_\mathcal{S}$ 少很多，但是在 Join 错误类型上增加了。经过数据检查发现，Join 出错的很多问题在数据库中存在相同列名在不同表中出现的情况，尽管 OPEN Prompt 找对了更多的列名，但是表依然预测错误。

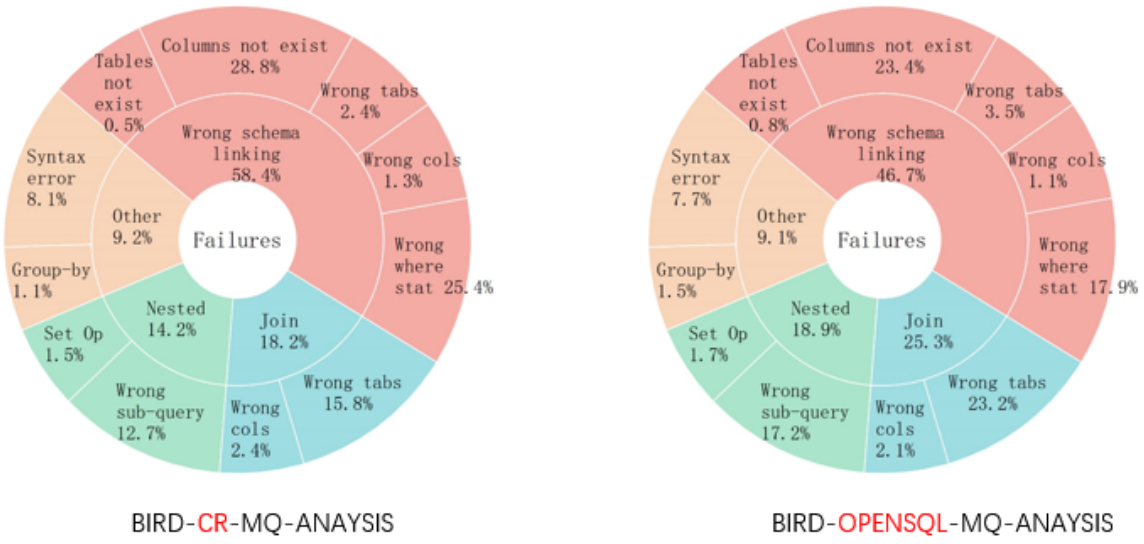

图 4. $\mathcal{DAIL}_\mathcal{S}$ vs OPENSQL Prompt 在 BIRD 错误分析

6 总结与展望

本文在 DAIL-SQL 的基础上提出了新的问题表达形式 OPEN Prompt。但是 OPEN Prompt 依然有缺点，特别是对于包含多个示例的上下文学习提示，使用现有的语言模型 API 仍然昂贵、耗时且受限于速率限制。因此，未来的研究可以集中在如何高效地进行 Prompt 工程方面，以提高大型语言模型在 Text-to-SQL 任务中生成正确 SQL 查询的性能，或者尝试微调开源大模型比如 llama 等减少花费。

参考文献

[1] Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. *CoRR*, abs/2305.03111, 2023.

[2] Ruichu Cai, Boyan Xu, Zhenjie Zhang, Xiaoyan Yang, Zijian Li, and Zhihao Liang. An encoder-decoder framework translating natural language to database queries. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 3977–3983, 2018.

[3] Shuaichen Chang and Eric Fosler-Lussier. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings, 2023.

[4] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey for in-context learning. *CoRR*, abs/2301.00234, 2023.

[5] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. C3: zero-shot text-to-sql with chatgpt. *CoRR*, abs/2307.07306, 2023.

[6] Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. A case-based reasoning framework for adaptive prompting in cross-domain text-to-sql. *CoRR*, abs/2304.13301, 2023.

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

[8] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. Kaggledbqa: Realistic evaluation of text-to-sql parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 2261–2273, 2021.

[9] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, page 73–84, Sep 2014.

[10] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. AAAI-23, page 13067–13075. AAAI Press, 2023.

[11] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *37th AAAI Conference on Artificial Intelligence*, pages 13076–13084, 2023.

[12] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *CoRR*, abs/2303.13547, 2023.

[13] Hu Liu, Yuliang Shi, Jianlin Zhang, Xinjun Wang, Hui Li, and Fanyu Kong. Multi-hop relational graph attention network for text-to-sql parsing. In *International Joint Conference on Neural Networks*, pages 1–8, 2023.

[14] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? In *Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, 2022.

[15] Tanzim Mahmud, K. M. Azharul Hasan, Mahtab Ahmed, and Thwoi Hla Ching Chak. A rule based approach for nlp based query processing. In *2015 2nd International Conference on Electrical Information and Communication Technologies (EICT)*, 2015.

[16] Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies. *CoRR*, abs/2305.12586, 2023.

[17] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[18] OpenAI. Introducing chatgpt. https://openai.com/blog/chatgpt, 2023. Last accessed on 2023-07-24.

[19] OpenAI. Sql translate. https://platform.openai.com/examples/default-sql-translate, 2023. Last accessed on 2023-07-24.

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

[21] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models. *CoRR*, abs/2204.00498, 2022.

[22] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[23] 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, 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. Llama2: Open foundation and fine-tuned chat models. *CoRR*, 2023.

[24] Immanuel Trummer. Codexdb: Synthesizing code for query processing from natural language instructions using gpt-3 codex. *Proceedings of the VLDB Endowment*, 15(11):2921–2928, 2022.

[25] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 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, 2020.

[26] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. Sql-to-text generation with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 931–936, 2018.

[27] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 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, 2018.

[28] Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-sql with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 396–411, 2020.

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

Listing 6: Example of Basic Prompt

```
Table continents , columns = [ContId , Continent]
Table countries , columns = [CountryId , CountryName  Continent]
Q: How many continents are there?
A: SELECT
```

Listing 7: Example of Text Representation Prompt

```
Given the following database schema:
continents: ContId , Continent
countries: CountryId , CountryName , Continent

Answer the following: How many continents are there?
SELECT
```

Listing 8: Example of OpenAI Demostration Prompt

```
### Complete sqlite SQL query only and with no explanation
### SQLite SQL tables , with their properties:
#
# continents(ContId , Continent)
# countries(CountryId , CountryName , Continent)
#
### How many continents are there?
SELECT
```

Listing 9: Example of Alpaca SFT Prompt

```
Below is an instruction that describes a task , paired with an input that
    provides further context.
Write a  response that appropriately completes the request.

### Instruction:
Write a sql to answer the question "How many continents are there?"

### Input:
continents(ContId , Continent)
countries(CountryId , CountryName , Continent)

### Response :
SELECT
```

Listing 10: OPENAI API: key 轮序机制代码

```python
def request_openai():
        openai_model.set_key(key)

        completion = openai_model.generate(instruction=prompt.
            instruction, input=prompt.input)
        logging.info(f"{LOG_LABEL}key {key} ,request ok")
        key_manager.release_key(key)
        return completion

    while attempts < max_retries:
        try:
            # Attempt to generate a completion
            completion = func_timeout(timeout=60, func=
                request_openai)
            break
        except KeyboardInterrupt:
            sys.exit(0)
        except FunctionTimedOut:
            # key_manager.remove_key(key)
            key_manager.release_key(key)
            key = key_manager.get_new_key(key)
            print("--------time out --------", key)
            # logging.error("{LOG_LABEL}Error occurred while
                accessing openai API")
            attempts += 1
            continue
        except Exception as e:
            # Handle different types of errors
            if "exceeded your current quota" in str(e) or "<empty
                message>" in str(e) or "Limit: 200 / day" in str(e)
                :
                # If the quota has been exceeded, remove the key
                    and try again
                key_manager.remove_key(key)
                key = key_manager.get_new_key()
                continue
            elif "Limit: 3 / min" in str(e) or "Limit: 40000 / min"
                in str(e):
                # If the rate limit is hit, switch the API key and
                    try again
                key = key_manager.get_new_key(key)
                continue
```