DIN-SQL：文本到 SQL 的分解上下文学习与自我修正到SQL推理中的性能

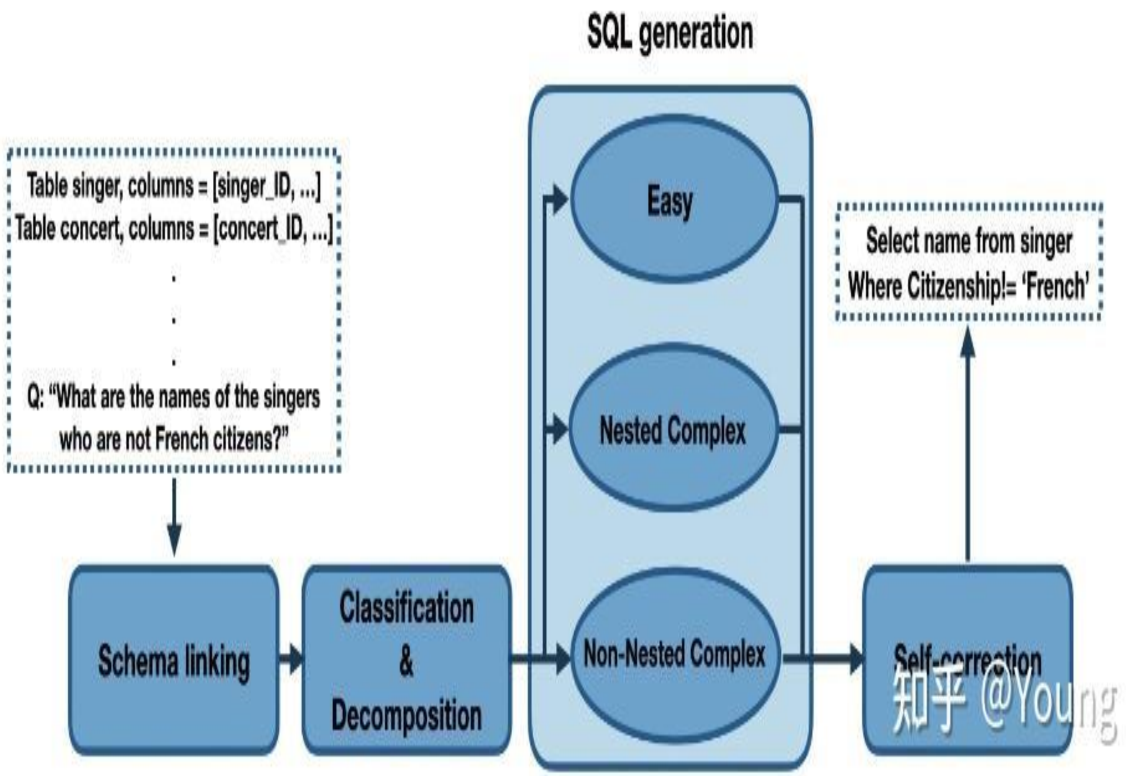
汇报：王嘉豪 王彦军 吉嘉成

2024-06-24

目录

01

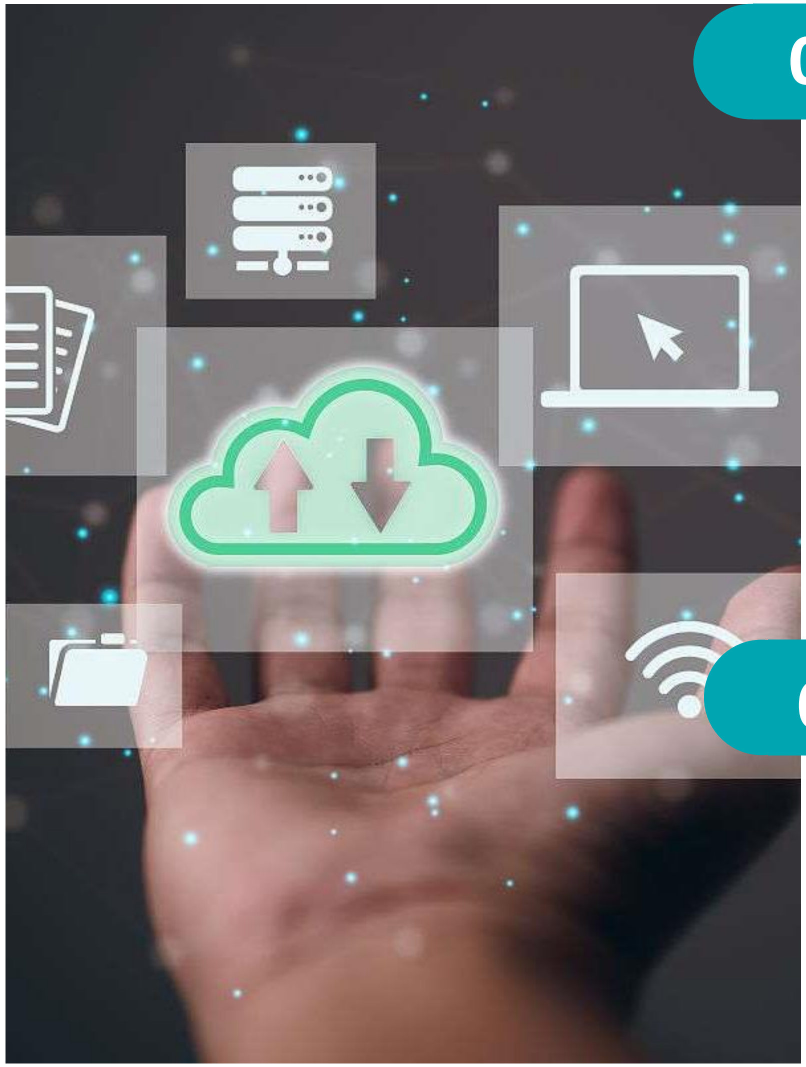
原理和数据集


SQL generation
Table singer, columns = [singer_ID, ...]
Table concert, columns = [concert_ID, ...]
Q: "What are the names of the singers who are not French citizens?"
Schema linking
Classification & Decomposition
Easy
Nested Complex
Non-Nested Complex
Self-correction
Select name from singer Where Citizenship!= 'French'
知乎 @Young

DIN-SQL提出了一个用于将文本转化为SQL任务的分解方法（如图），包括四个模块：(1)模式链接， 检测与查询相关的数据库表和列。(2)查询分类和分解，识别更复杂查询的一般查询结构（例如，按组分组，嵌套，多个连接，集合操作等）。(3)SQL生成，如果可以确定，制定任何过程性的子组件。(4)自我纠正，根据子问题的解决方案编写最终查询。

。这些模块的目标是将复杂的文本转化为更小的子问题，并利用语言模型来解决这些子问题，从而构建出原始问题的解决方案。虽然这些模块可以使用文献中的技术来实现，但作者选择使用提示技术来展示这种方法的可行性。这种方法的关键在于将问题分解到适当的细粒度，以便LLMs能够解决它们。

"din-sql"方法原理



01

Schema Linking Module

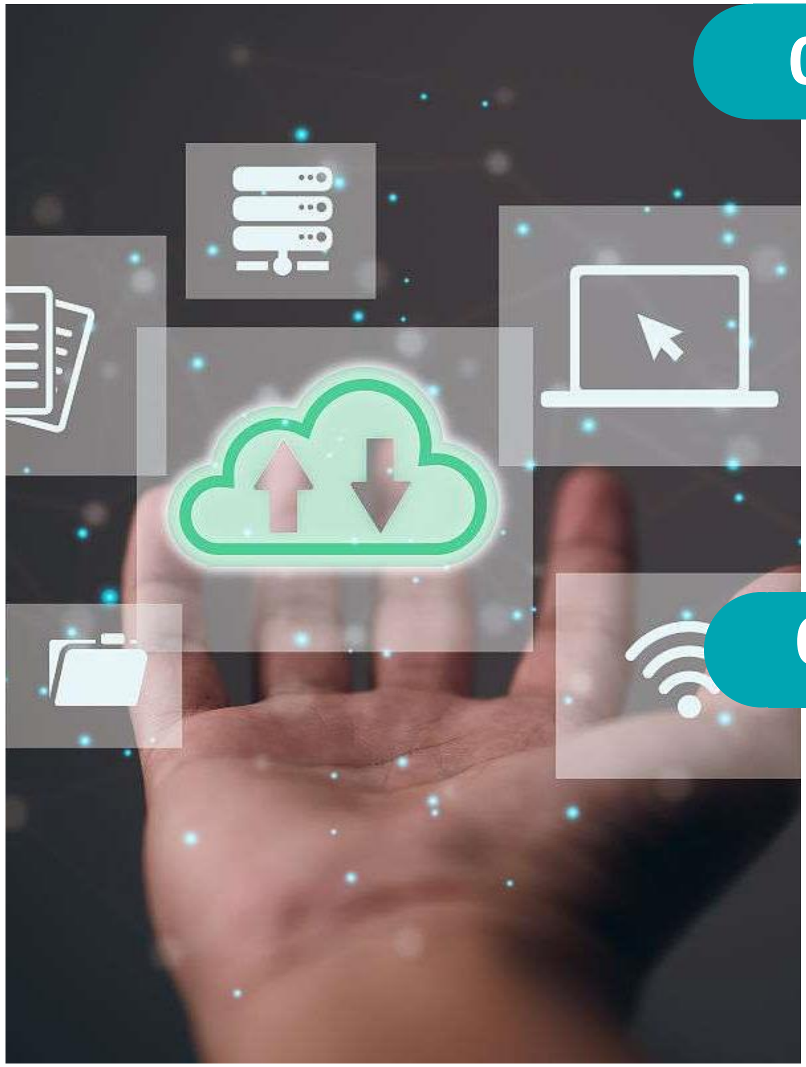
Schema linking在自然语言查询中负责识别数据库模式和条件值的引用。研究表明，它有助于跨领域的泛化性和复杂查询的综合。我们设计了一个基于提示的模块用于schema linking。提示包括从Spider数据集的训练集中随机选择的十个样本。遵循 Chain of Thought 模板（Wei等，2022b），提示以“让我们一步一步思考”开始，如Kojima等（2022）所建议。对于问题中每个列名的提及，从给定的数据库模式中选择相应的列和它们的表。还从问题中提取可能的实体和单元格值。

02

Classification & Decomposition Module

该模块将查询分为三个类别：简单、非嵌套复杂和嵌套复杂。简单类别包括不需要连接或嵌套就可以回答的单表查询。非嵌套类别包括需要连接但不需要子查询的查询，而嵌套类别的查询可能需要连接、子查询和集合操作。这个模块的目的是帮助解决更复杂的查询问题，并将查询问题分解为更小的子问题，以便更好地处理和生成SQL查询语句。

"din-sql"方法原理



03

SQL Generation Module

我们的非嵌套复杂类别包括需要连接(join)的查询我们采用了一种中间表示方法来连接查询和SQL语句。引入了SemQL (Guo等，2019)去除了在自然语言查询中没有明确对应的JOIN ON、FROM和GROUP BY操作符，并合并了HAVING和WHERE子句。NatSQL (Gan等，2021)在SemQL的基础上进一步去除了集合操作符。作为我们的中间表示，我们使用了NatSQL，该方法与其他模型结合使用时表现出色。嵌套复杂类别是最复杂的类型，生成最终答案之前需要多个中间步骤为了进一步分解问题，我们设计了这个类别的提示方式，要求LLM首先解决子查询，然后使用它们生成最终答案。

04

Self-correction Module

生成的SQL查询有时可能会出现缺少或冗余的关键词，例如DESC、DISTINCT和聚合函数。我们在使用多个LLM（语言模型）时发现，这些问题在较大的LLM中较少出现（例如，由GPT-4生成的查询比CodeX生成的查询有更少的错误），但仍然存在。为了解决这个问题，我们提出了一个自我修正模块，指示模型纠正这些小错误。

数据集介绍

01

在评估"din-sql"方法时，使用了多个数据集，其中最重要的是Spider数据集。Spider是一个用于评估文本到SQL技术的基准数据集，包含大量的自然语言查询和对应的SQL查询。

02

Spider数据集：Spider数据集是一个大型、复杂的文本到SQL数据集，包含超过10,000个自然语言查询和对应的SQL查询。这些查询涉及多个数据库表、复杂的查询结构和丰富的语义信息。Spider数据集被广泛应用于评估文本到SQL技术的性能。

03

评估指标：在Spider数据集上，"din-sql"方法通过执行准确度来评估其性能。执行准确度是指生成的SQL查询能够正确执行并返回与原始查询相同结果的比例。通过使用"din-sql"方法，研究人员在Spider的Holdout测试集上实现了85.3%的执行准确度，显著超过了之前的最佳性能（79.9%）。

04

通过"din-sql"方法，研究人员成功地将复杂的文本到SQL任务分解为更小的子任务，并通过将子问题的解决方案输入到LLM中来提高其性能。这种方法不仅提高了LLM在推理过程中的准确性，还为其在更广泛的自然语言处理任务中的应用提供了新的思路。

02

主要预测代码

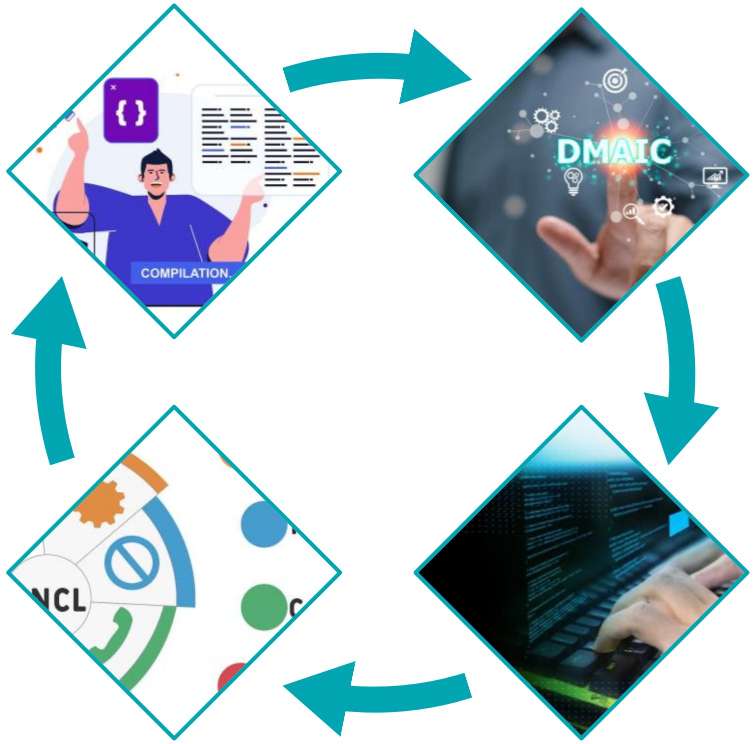
代码框架与模块

模式链接

根据数据库模式和外键,为每个问题生成SQL查询的模式链接

查询分类

生成不同难度的提示：hard，medium，easy



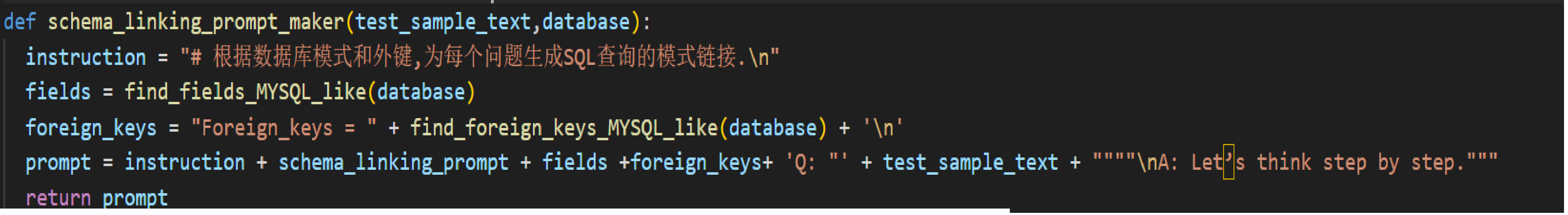
分解

根据给定的问题文本、数据库和模式链接，生成一个分类提示，指导用户将SQL查询分类为“EASY”、“NON-NESTED”或“NESTED”

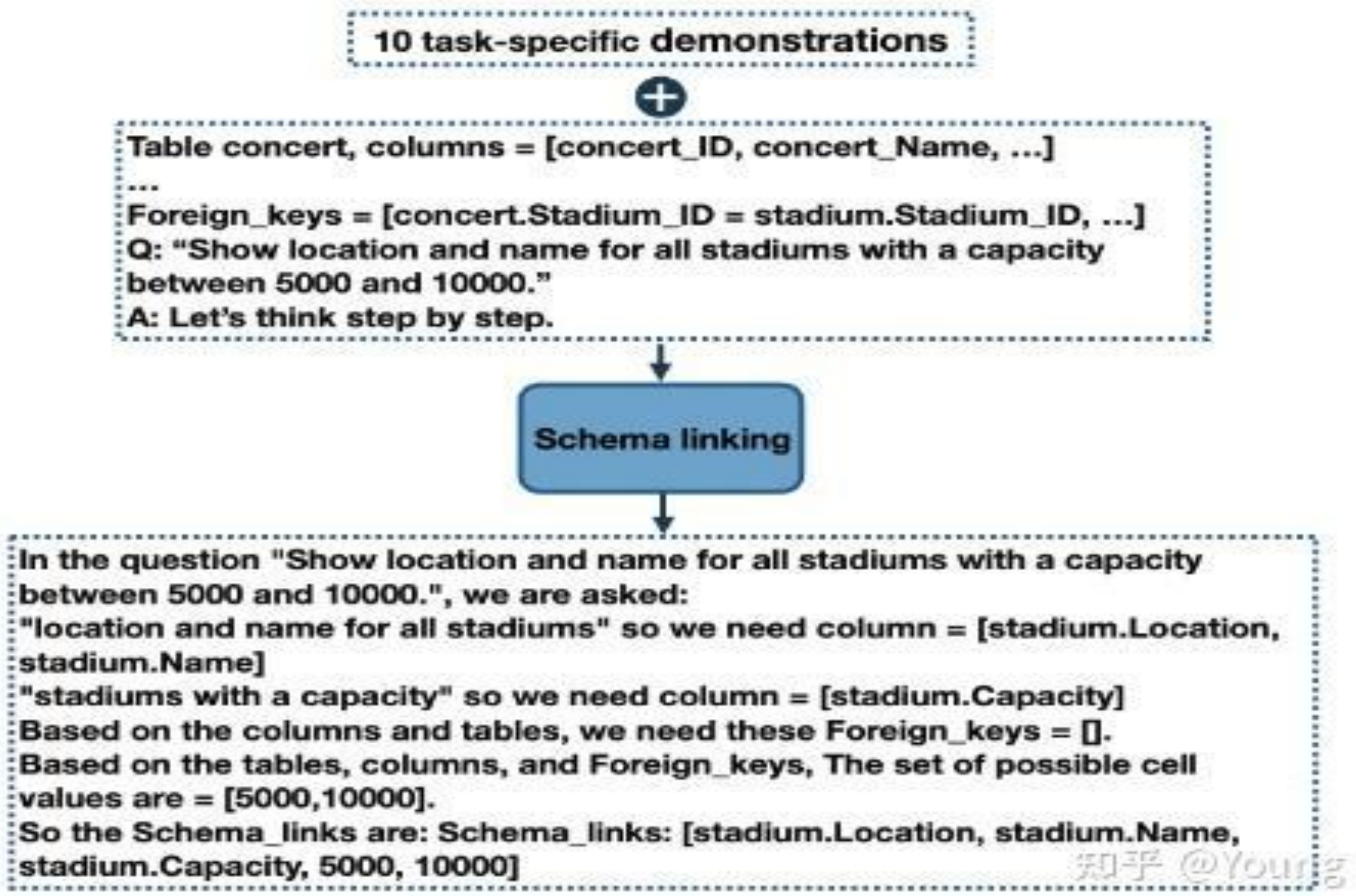
自我纠正

基于语义理解的结果，结合预定义的SQL模板和规则，生成符合数据库查询需求的SQL语句。

模式链接



```python
def schema_linking_prompt_maker(test_sample_text,database):
  instruction = "# 根据数据库模式和外键,为每个问题生成SQL查询的模式链接.\n"
  fields = find_fields_MYSQL_like(database)
  foreign_keys = "Foreign_keys = " + find_foreign_keys_MYSQL_like(database) + '\n'
  prompt = instruction + schema_linking_prompt + fields +foreign_keys+ 'Q: "' + test_sample_text + """"\nA: Let’s think step by step."""
  return prompt
```

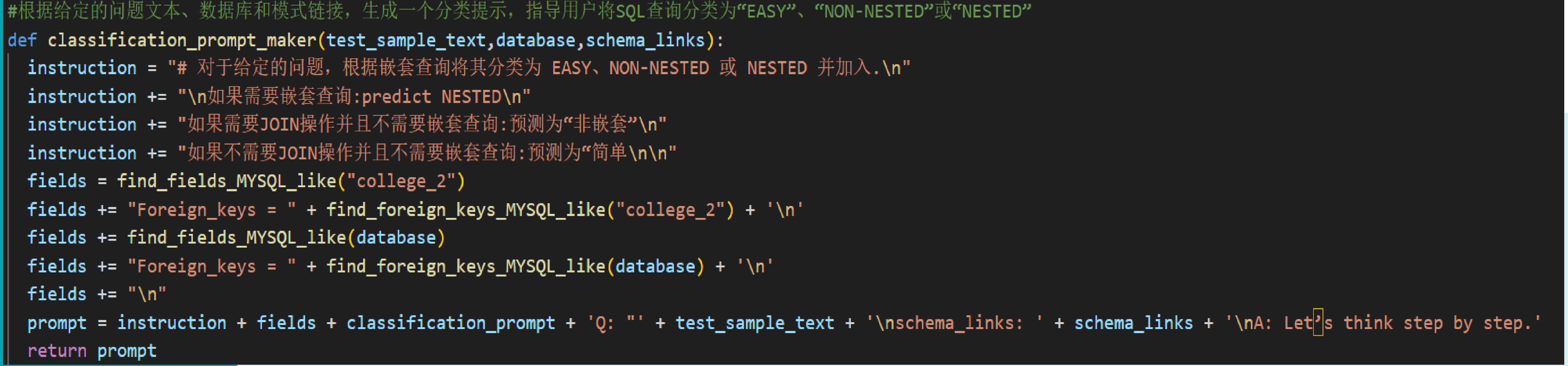


10 task-specific demonstrations

Table concert, columns = [concert_ID, concert_Name, ...]
...
Foreign_keys = [concert.Stadium_ID = stadium.Stadium_ID, ...]
Q: "Show location and name for all stadiums with a capacity between 5000 and 10000."
A: Let's think step by step.

Schema linking

In the question "Show location and name for all stadiums with a capacity between 5000 and 10000.", we are asked:
"location and name for all stadiums" so we need column = [stadium.Location, stadium.Name]
"stadiums with a capacity" so we need column = [stadium.Capacity]
Based on the columns and tables, we need these Foreign_keys = [].
Based on the tables, columns, and Foreign_keys, The set of possible cell values are = [5000,10000].
So the Schema_links are: Schema_links: [stadium.Location, stadium.Name, stadium.Capacity, 5000, 10000]



```
#根据给定的问题文本、数据库和模式链接，生成一个分类提示，指导用户将SQL查询分类为“EASY”、“NON-NESTED”或“NESTED”
def classification_prompt_maker(test_sample_text,database,schema_links):
  instruction = "# 对于给定的问题，根据嵌套查询将其分类为 EASY、NON-NESTED 或 NESTED 并加入.\n"
  instruction += "\n如果需要嵌套查询:predict NESTED\n"
  instruction += "如果需要JOIN操作并且不需要嵌套查询:预测为“非嵌套”\n"
  instruction += "如果不需要JOIN操作并且不需要嵌套查询:预测为“简单\n\n"
  fields = find_fields_MYSQL_like("college_2")
  fields += "Foreign_keys = " + find_foreign_keys_MYSQL_like("college_2") + '\n'
  fields += find_fields_MYSQL_like(database)
  fields += "Foreign_keys = " + find_foreign_keys_MYSQL_like(database) + '\n'
  fields += "\n"
  prompt = instruction + fields + classification_prompt + 'Q: "' + test_sample_text + '\nschema_links: ' + schema_links + '\nA: Let’s think step by step.'
  return prompt
```

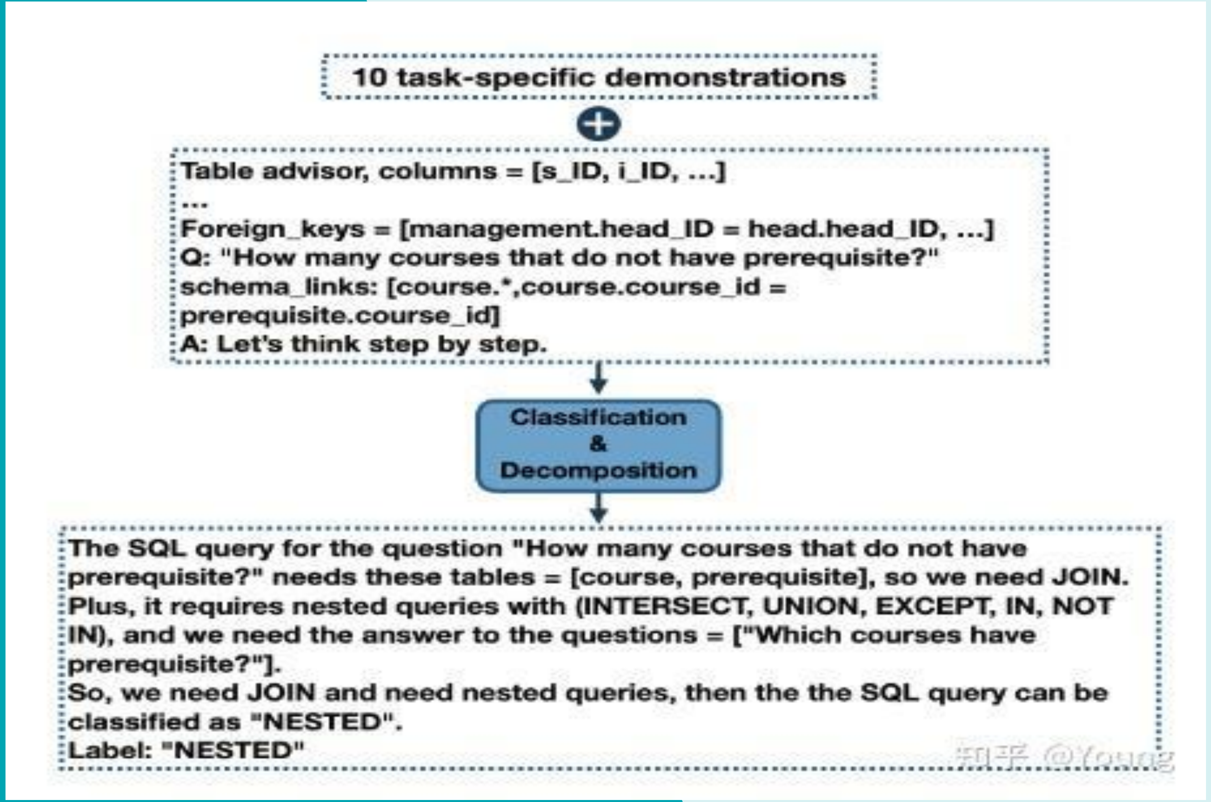
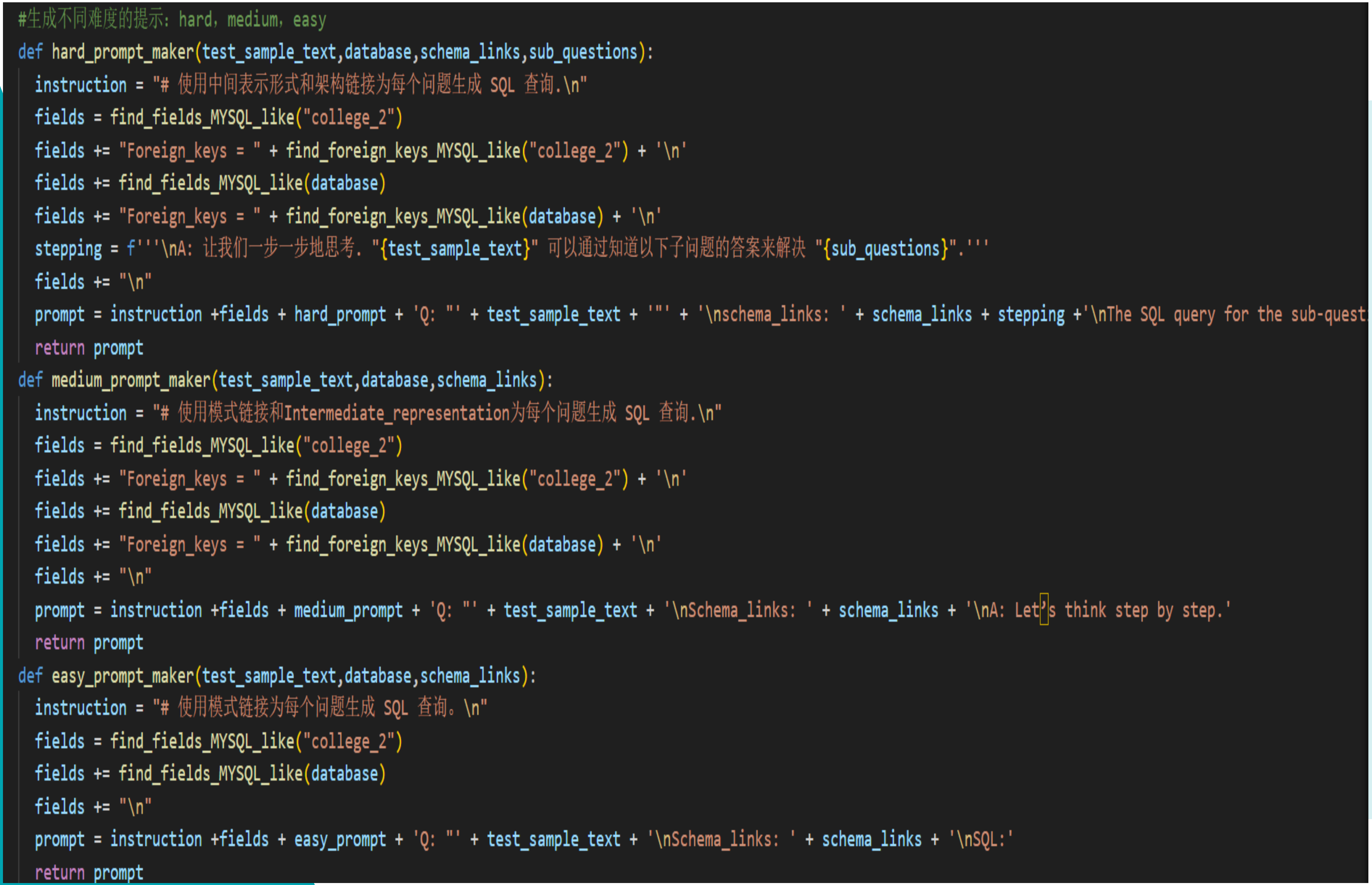

10 task-specific demonstrations
Classification & Decomposition

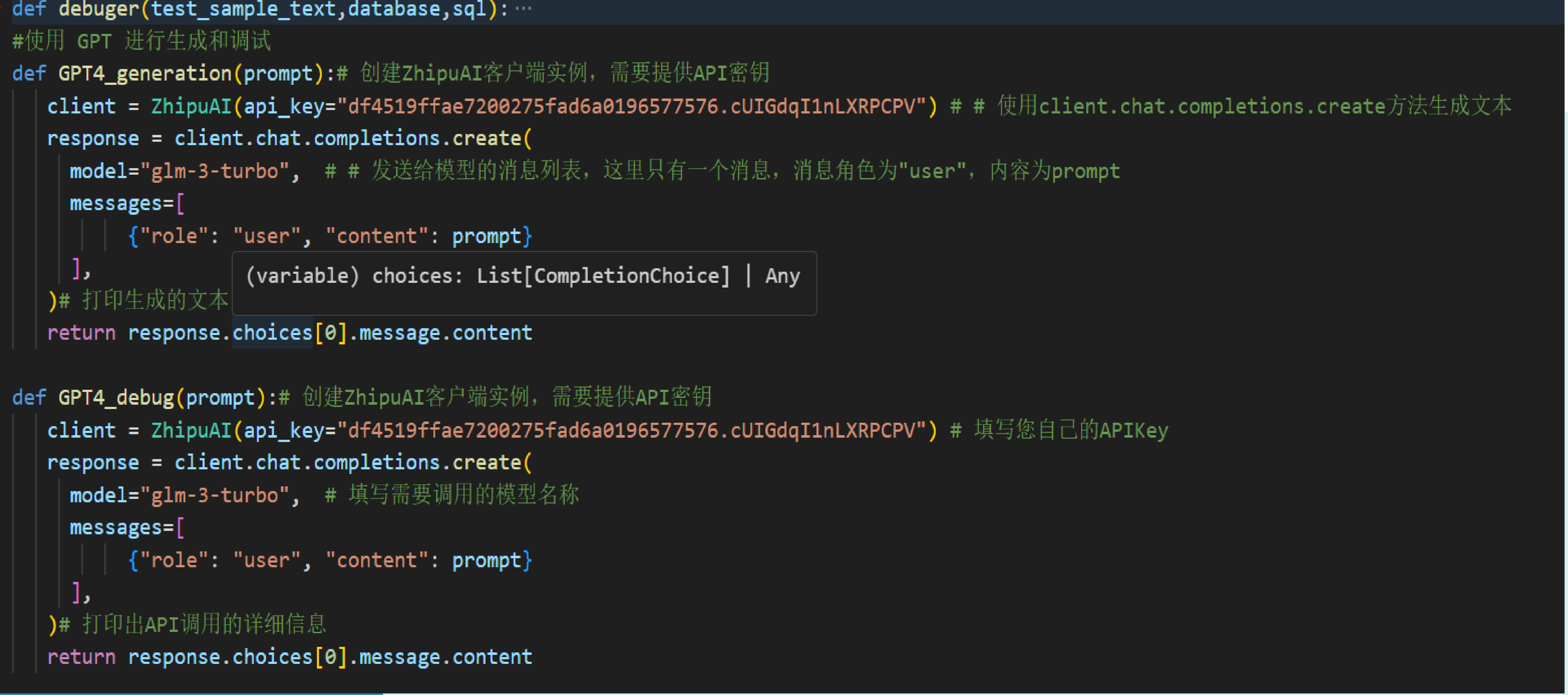
Table advisor, columns = [s_ID, i_ID, ...]
...
Foreign_keys = [management.head_ID = head.head_ID, ...]
Q: "How many courses that do not have prerequisite?"
schema_links: [course.*,course.course_id = prerequisite.course_id]
A: Let's think step by step.

The SQL query for the question "How many courses that do not have prerequisite?" needs these tables = [course, prerequisite], so we need JOIN.
Plus, it requires nested queries with (INTERSECT, UNION, EXCEPT, IN, NOT IN), and we need the answer to the questions = ["Which courses have prerequisite?"].
So, we need JOIN and need nested queries, then the the SQL query can be classified as "NESTED".
Label: "NESTED"

分类



```python
#生成不同难度的提示：hard，medium，easy
def hard_prompt_maker(test_sample_text,database,schema_links,sub_questions):
  instruction = "# 使用中间表示形式和架构链接为每个问题生成 SQL 查询.\n"
  fields = find_fields_MYSQL_like("college_2")
  fields += "Foreign_keys = " + find_foreign_keys_MYSQL_like("college_2") + '\n'
  fields += find_fields_MYSQL_like(database)
  fields += "Foreign_keys = " + find_foreign_keys_MYSQL_like(database) + '\n'
  stepping = f'''\nA: 让我们一步一步地思考. "{test_sample_text}" 可以通过知道以下子问题的答案来解决 "{sub_questions}".'''
  fields += "\n"
  prompt = instruction +fields + hard_prompt + 'Q: "' + test_sample_text + '"' + '\nschema_links: ' + schema_links + stepping +'\nThe SQL query for the sub-quest
  return prompt
def medium_prompt_maker(test_sample_text,database,schema_links):
  instruction = "# 使用模式链接和Intermediate_representation为每个问题生成 SQL 查询.\n"
  fields = find_fields_MYSQL_like("college_2")
  fields += "Foreign_keys = " + find_foreign_keys_MYSQL_like("college_2") + '\n'
  fields += find_fields_MYSQL_like(database)
  fields += "Foreign_keys = " + find_foreign_keys_MYSQL_like(database) + '\n'
  fields += "\n"
  prompt = instruction +fields + medium_prompt + 'Q: "' + test_sample_text + '\nSchema_links: ' + schema_links + '\nA: Let's think step by step.'
  return prompt
def easy_prompt_maker(test_sample_text,database,schema_links):
  instruction = "# 使用模式链接为每个问题生成 SQL 查询。\n"
  fields = find_fields_MYSQL_like("college_2")
  fields += find_fields_MYSQL_like(database)
  fields += "\n"
  prompt = instruction +fields + easy_prompt + 'Q: "' + test_sample_text + '\nSchema_links: ' + schema_links + '\nSQL:'
  return prompt
```



```
def debuger(test_sample_text,database,sql): ...
#使用 GPT 进行生成和调试
def GPT4_generation(prompt):# 创建ZhipuAI客户端实例，需要提供API密钥
    client = ZhipuAI(api_key="df4519ffae7200275fad6a0196577576.cUIGdqI1nLXRPCPV") # # 使用client.chat.completions.create方法生成文本
    response = client.chat.completions.create(
        model="glm-3-turbo",  # # 发送给模型的消息列表，这里只有一个消息，消息角色为"user"，内容为prompt
        messages=[
            {"role": "user", "content": prompt}
        ],
    )# 打印生成的文本
    return response.choices[0].message.content

def GPT4_debug(prompt):# 创建ZhipuAI客户端实例，需要提供API密钥
    client = ZhipuAI(api_key="df4519ffae7200275fad6a0196577576.cUIGdqI1nLXRPCPV") # 填写您自己的APIKey
    response = client.chat.completions.create(
        model="glm-3-turbo",  # 填写需要调用的模型名称
        messages=[
            {"role": "user", "content": prompt}
        ],
    )# 打印出API调用的详细信息
    return response.choices[0].message.content
```

使用智谱AI API接入

预测算法实现

01 **基于Transformer的编码器**

采用Transformer架构作为编码器，对输入文本进行编码，捕捉文本中的长距离依赖关系和上下文信息。

02 **注意力机制**

在编码过程中引入注意力机制，使模型能够关注到输入文本中的关键信息，提高语义理解的准确性。

03 **模板与规则匹配**

结合预定义的SQL模板和规则，将语义理解的结果映射到具体的SQL语句结构上，实现文本到SQL的转换。

04 **解码器与生成策略**

设计合适的解码器和生成策略，确保生成的SQL语句既符合语法规范，又能满足用户查询的需求。

代码优化与效率提升

利用多核CPU或GPU进行并行计算，加速模型的训练和推理过程。

并行计算

模型压缩

采用模型剪枝、量化等技术对模型进行压缩，减少模型参数和计算量，提高推理速度。

引入缓存机制，对已经处理过的输入和输出进行缓存，避免重复计算，提高整体效率。

缓存机制

异步处理

采用异步处理方式，将输入处理、语义理解、SQL生成和输出处理等模块解耦，实现并发执行，进一步提高处理速度。

03

评价标准与准确率分析

方法：使用测试套件对文本到SQL进行评估

评估原理：

执行测试套件中的查询，并比较模型的输出与标准输出。测试套件是一组数据库路径，用于比较表示和提取常量值。如果模型的输出在语义上与标准输出相同，则认为模型通过了该测试用例。

评估指标：

1. 执行精度（execution accuracy）
2. 语义匹配精度（semantic matching accuracy）

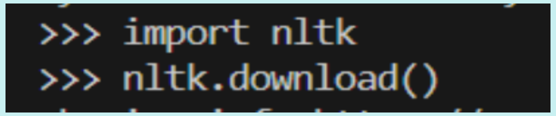
```
choices=('all','exec','match'
```

3. 部分匹配精度（partial matching accuracy）SELECT 、FROM 、WHERE
4. 难度等级:easy;medium;hard;extra;all

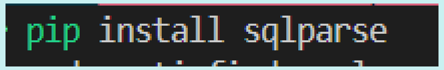
评估过程：

解析命令行参数：

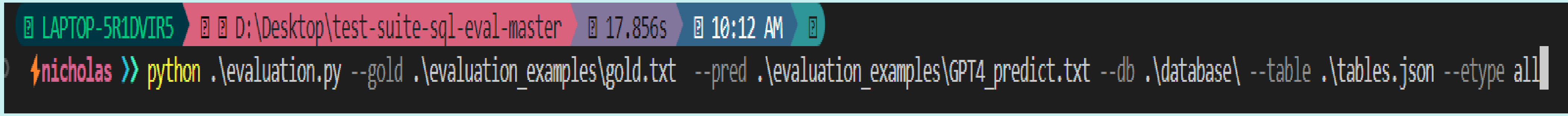
1.	正确查询文件路径	[gold file]
2.	预测查询文件路径	[predict file]
3.	数据库目录路径	[data dir]
4.	外键信息路径	[table file]
5.	评估类型	[evaluation type]



```
>>> import nltk
>>> nltk.download()
```

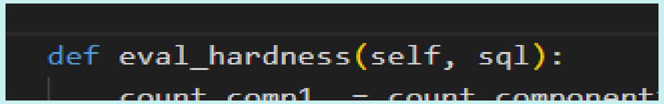


```
pip install sqlparse
```

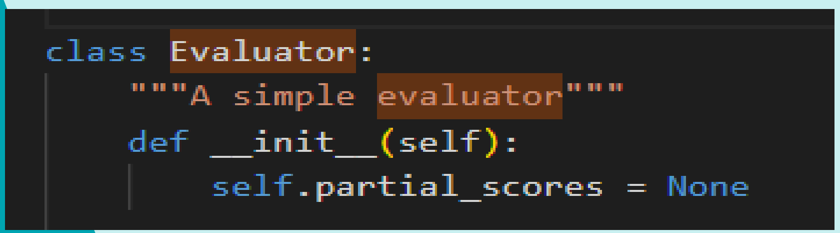


```
LAPTOP-5R1DVIR5  D:\Desktop\test-suite-sql-eval-master  17.856s  10:12 AM
nicholas ❯❯ python .\evaluation.py --gold .\evaluation_examples\gold.txt  --pred .\evaluation_examples\GPT4_predict.txt --db .\database\ --table .\tables.json --etype all
```

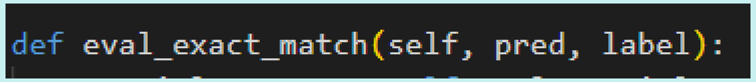
使用解析的参数构建 Evaluator 对象,评估查询：



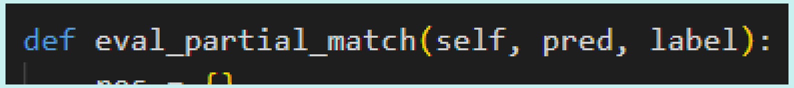
```
def eval_hardness(self, sql):
```



```
class Evaluator:
    """A simple evaluator"""
    def __init__(self):
        self.partial_scores = None
```



```
def eval_exact_match(self, pred, label):
```



```
def eval_partial_match(self, pred, label):
```

准确率提升策略探讨

模型结构优化

针对文本到SQL推理任务的特点，优化模型的结构和参数设置。通过引入注意力机制、多任务学习等技术，提高模型对文本和SQL语句的理解能力。

数据增强与预训练

通过数据增强技术生成更多的训练样本，提高模型的泛化能力。同时，利用预训练技术在大规模语料库上进行预训练，为模型提供丰富的背景知识和先验信息。

后处理优化

对生成的SQL语句进行后处理优化，包括语法检查、逻辑修正和性能优化等。通过后处理优化，可以进一步提高生成的SQL语句的准确性和执行效率。

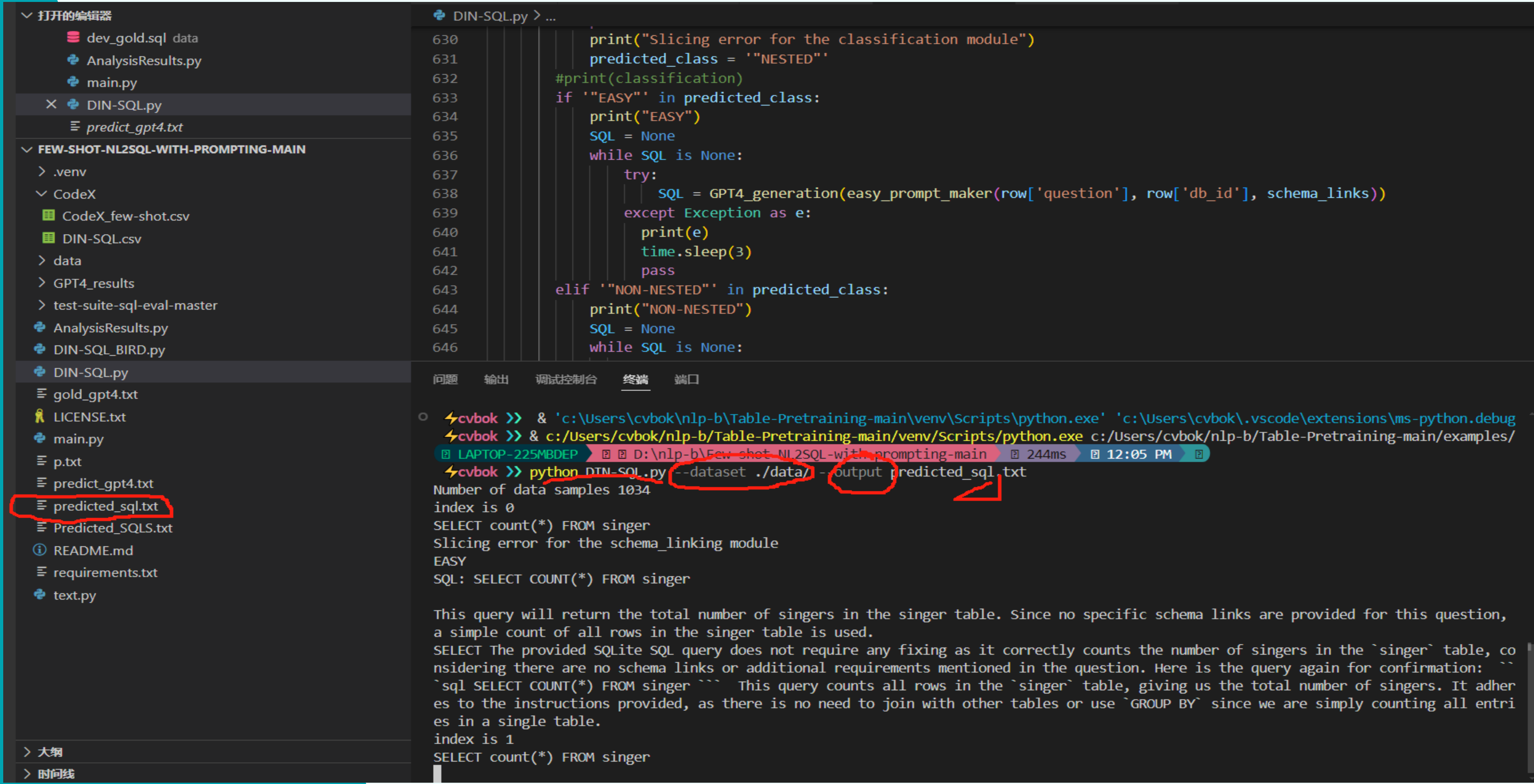
用户反馈与迭代优化

收集用户反馈和错误报告，对模型进行迭代优化。通过不断地改进和优化模型，提高其在文本到SQL推理任务中的性能和准确率。

04

实验与结果展示

TAPEX： TABLE PRE-TRAINING VIA LEARNING ：

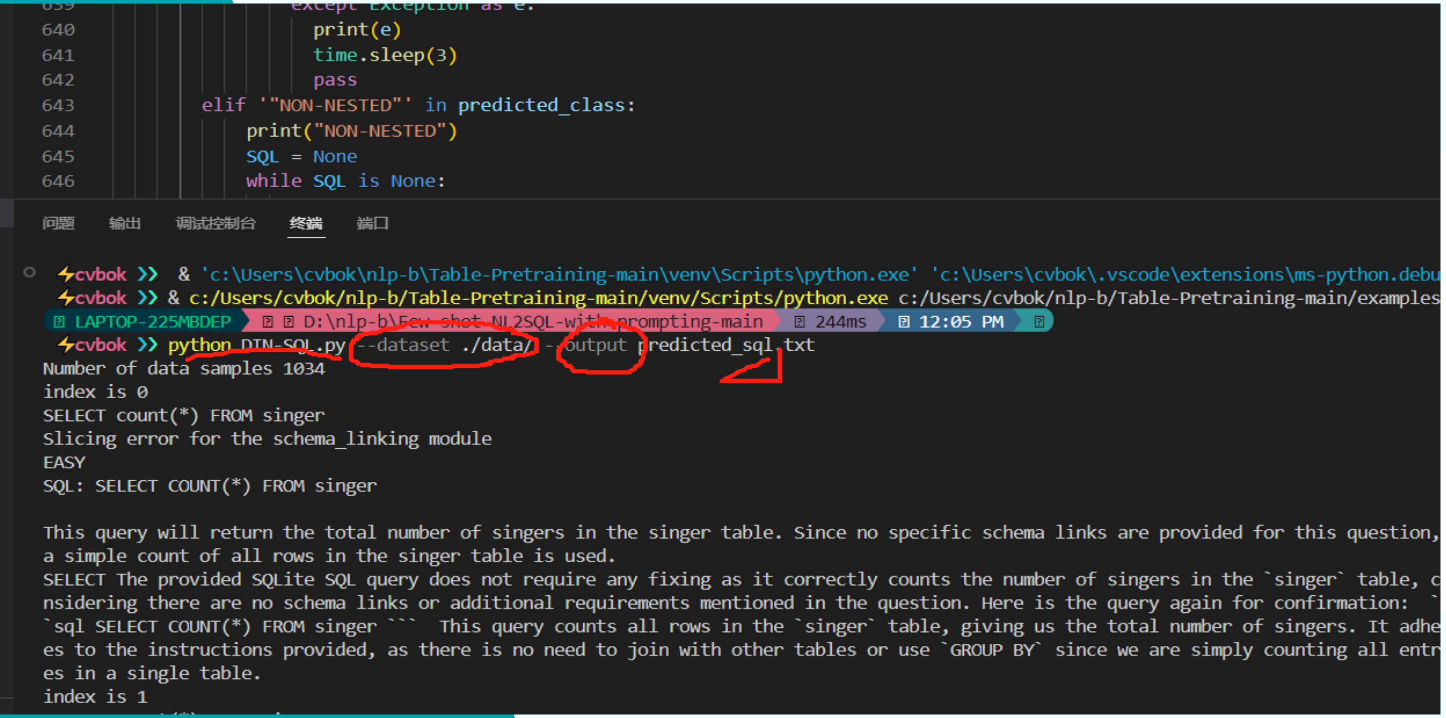


```python
                print("Slicing error for the classification module")
                predicted_class = '"NESTED"'
            #print(classification)
            if '"EASY"' in predicted_class:
                print("EASY")
                SQL = None
                while SQL is None:
                    try:
                        SQL = GPT4_generation(easy_prompt_maker(row['question'], row['db_id'], schema_links))
                    except Exception as e:
                        print(e)
                        time.sleep(3)
                        pass
            elif '"NON-NESTED"' in predicted_class:
                print("NON-NESTED")
                SQL = None
                while SQL is None:
```

```
⚡cvbok ❯❯  & 'c:\Users\cvbok\nlp-b\Table-Pretraining-main\venv\Scripts\python.exe' 'c:\Users\cvbok\.vscode\extensions\ms-python.debug
⚡cvbok ❯❯ & c:/Users/cvbok/nlp-b/Table-Pretraining-main/venv/Scripts/python.exe c:/Users/cvbok/nlp-b/Table-Pretraining-main/examples/
 LAPTOP-225MBDEP   D:\nlp-b\Few-shot-NL2SQL-with-prompting-main   244ms   12:05 PM
⚡cvbok ❯❯ python DIN-SQL.py --dataset ./data/ --output predicted_sql.txt
Number of data samples 1034
index is 0
SELECT count(*) FROM singer
Slicing error for the schema_linking module
EASY
SQL: SELECT COUNT(*) FROM singer

This query will return the total number of singers in the singer table. Since no specific schema links are provided for this question,
a simple count of all rows in the singer table is used.
SELECT The provided SQLite SQL query does not require any fixing as it correctly counts the number of singers in the `singer` table, co
nsidering there are no schema links or additional requirements mentioned in the question. Here is the query again for confirmation: ``
`sql SELECT COUNT(*) FROM singer ```  This query counts all rows in the `singer` table, giving us the total number of singers. It adher
es to the instructions provided, as there is no need to join with other tables or use `GROUP BY` since we are simply counting all entri
es in a single table.
index is 1
SELECT count(*) FROM singer
```

DIN-SQL预测训练：

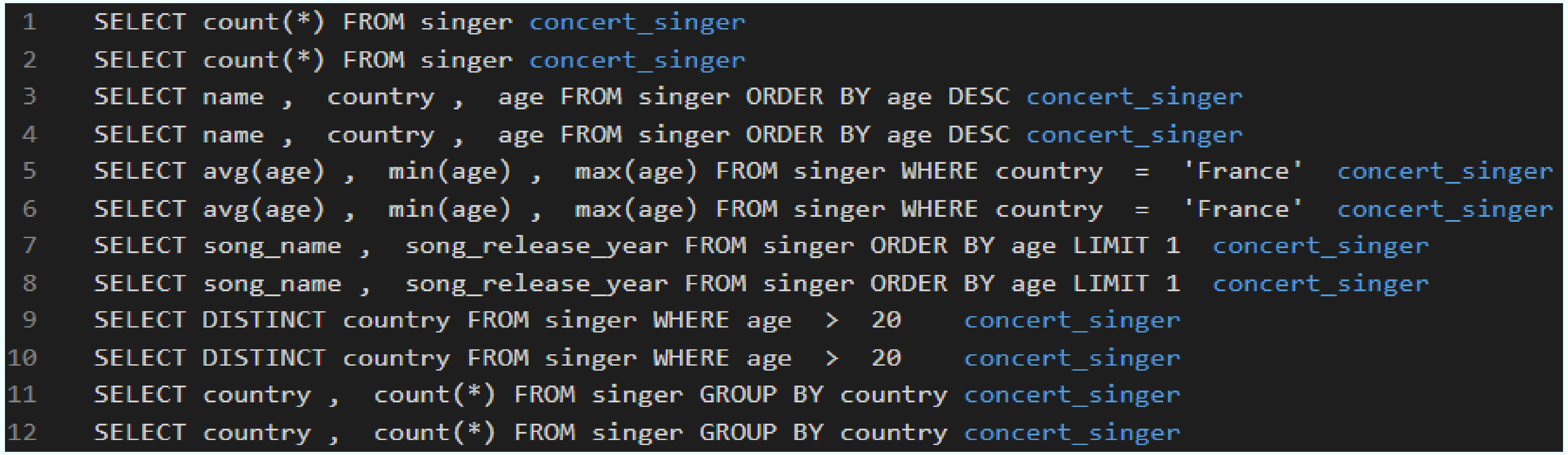


```
                except Exception as e:
                    print(e)
                    time.sleep(3)
                    pass
            elif '"NON-NESTED"' in predicted_class:
                print("NON-NESTED")
                SQL = None
                while SQL is None:
```

问题　输出　调试控制台　终端　端口

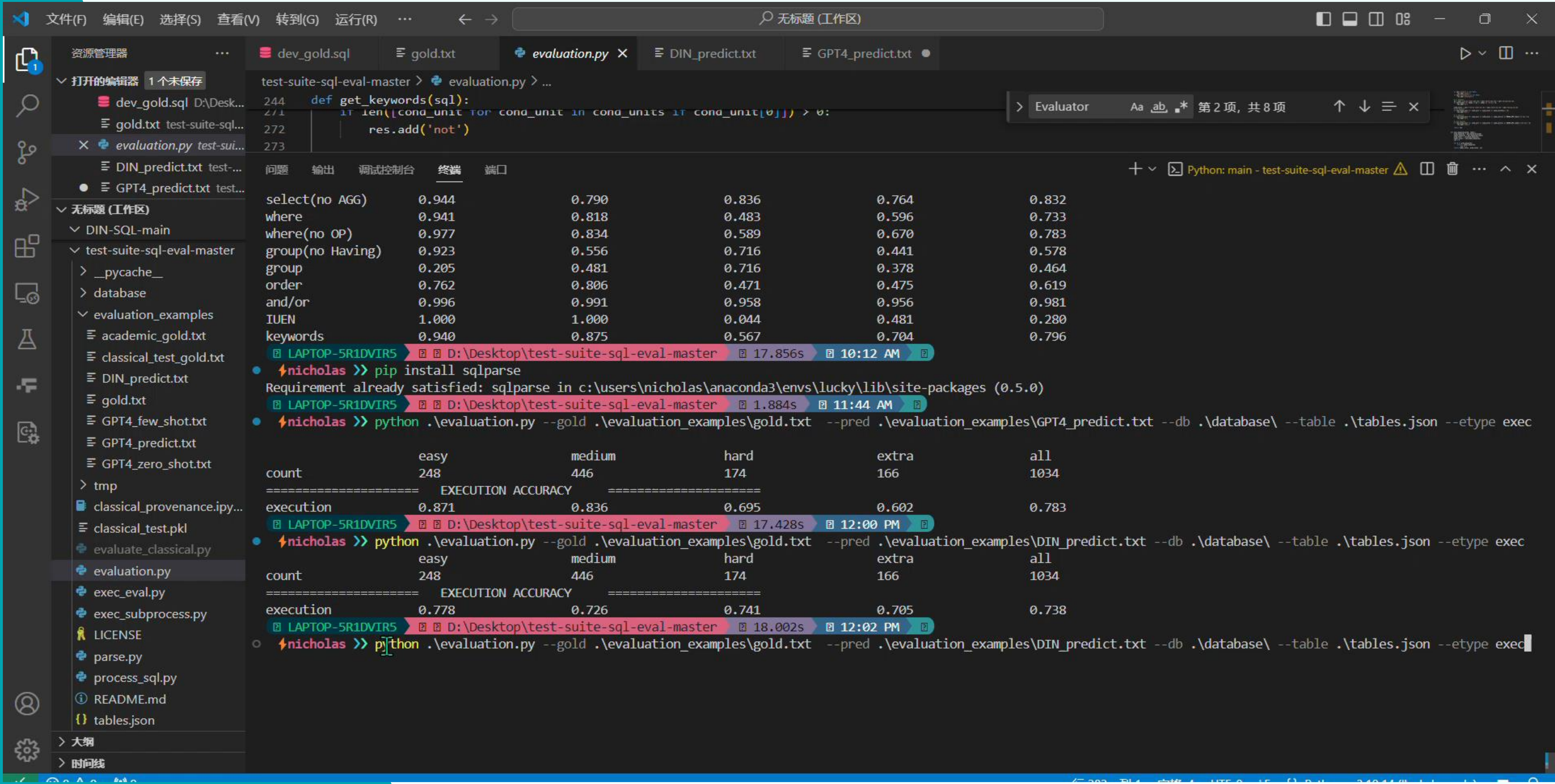
```
⚡cvbok ❯❯  & 'c:\Users\cvbok\nlp-b\Table-Pretraining-main\venv\Scripts\python.exe' 'c:\Users\cvbok\.vscode\extensions\ms-python.debu
⚡cvbok ❯❯ & c:/Users/cvbok/nlp-b/Table-Pretraining-main/venv/Scripts/python.exe c:/Users/cvbok/nlp-b/Table-Pretraining-main/examples
 LAPTOP-225MBDEP   D:\nlp-b\Few-shot-NL2SQL-with-prompting-main   244ms   12:05 PM
⚡cvbok ❯❯ python DIN-SQL.py --dataset ./data/ --output predicted_sql.txt
Number of data samples 1034
index is 0
SELECT count(*) FROM singer
Slicing error for the schema_linking module
EASY
SQL: SELECT COUNT(*) FROM singer

This query will return the total number of singers in the singer table. Since no specific schema links are provided for this question,
a simple count of all rows in the singer table is used.
SELECT The provided SQLite SQL query does not require any fixing as it correctly counts the number of singers in the `singer` table, c
nsidering there are no schema links or additional requirements mentioned in the question. Here is the query again for confirmation:  `
`sql SELECT COUNT(*) FROM singer ```  This query counts all rows in the `singer` table, giving us the total number of singers. It adhe
es to the instructions provided, as there is no need to join with other tables or use `GROUP BY` since we are simply counting all entr
es in a single table.
index is 1
```



```
SELECT count(*) FROM singer concert_singer
SELECT count(*) FROM singer concert_singer
SELECT name ,  country ,  age FROM singer ORDER BY age DESC concert_singer
SELECT name ,  country ,  age FROM singer ORDER BY age DESC concert_singer
SELECT avg(age) ,  min(age) ,  max(age) FROM singer WHERE country  =  'France'  concert_singer
SELECT avg(age) ,  min(age) ,  max(age) FROM singer WHERE country  =  'France'  concert_singer
SELECT song_name ,  song_release_year FROM singer ORDER BY age LIMIT 1  concert_singer
SELECT song_name ,  song_release_year FROM singer ORDER BY age LIMIT 1  concert_singer
SELECT DISTINCT country FROM singer WHERE age  >  20   concert_singer
SELECT DISTINCT country FROM singer WHERE age  >  20   concert_singer
SELECT country ,  count(*) FROM singer GROUP BY country concert_singer
SELECT country ,  count(*) FROM singer GROUP BY country concert_singer
```

DIN-SQL准确率评估：

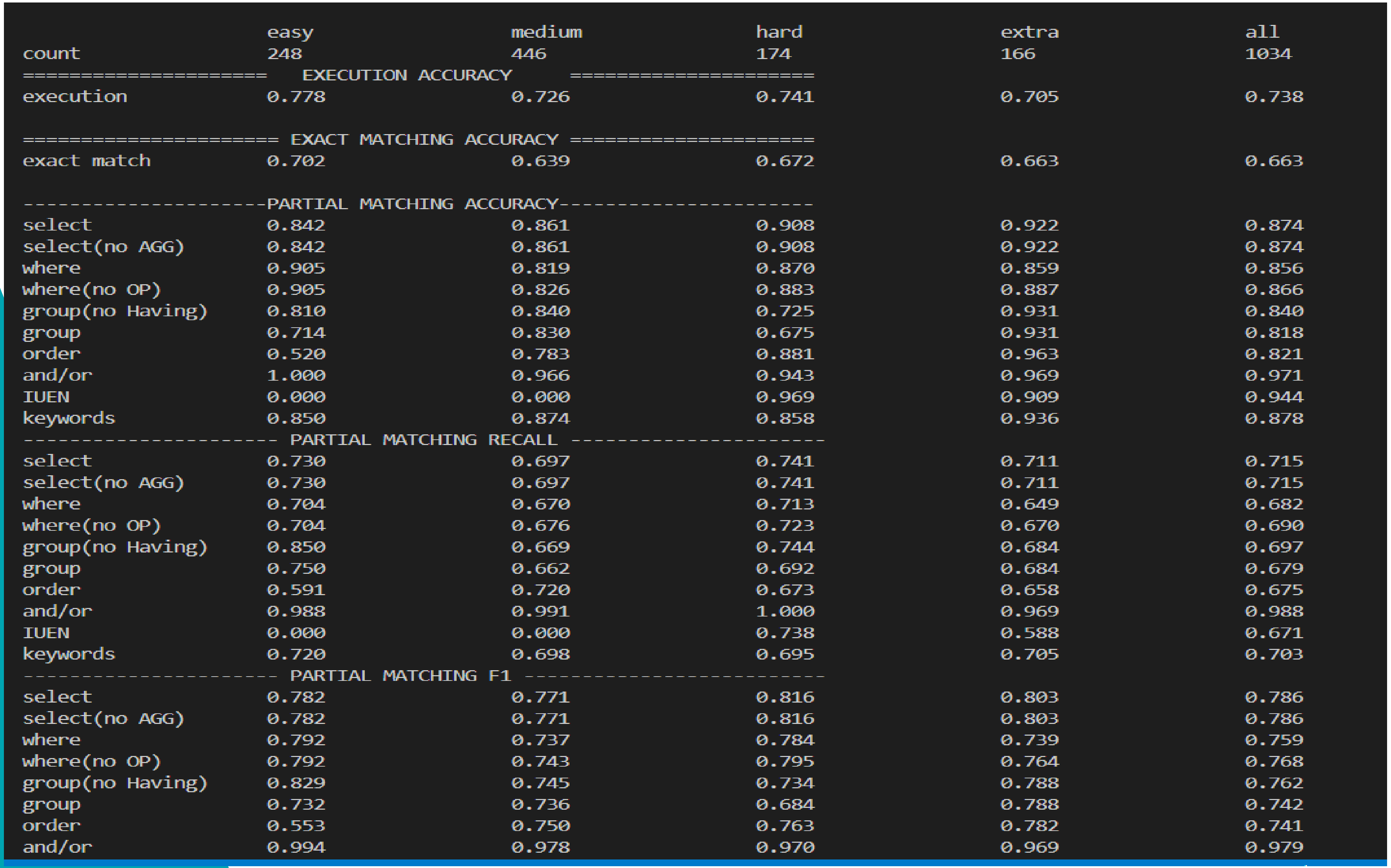


```python
def get_keywords(sql):
    if len([cond_unit for cond_unit in cond_units if cond_unit[0]]) > 0:
        res.add('not')
```

```
select(no AGG)      0.944          0.790          0.836          0.764          0.832
where               0.941          0.818          0.483          0.596          0.733
where(no OP)        0.977          0.834          0.589          0.670          0.783
group(no Having)    0.923          0.556          0.716          0.441          0.578
group               0.205          0.481          0.716          0.378          0.464
order               0.762          0.806          0.471          0.475          0.619
and/or              0.996          0.991          0.958          0.956          0.981
IUEN                1.000          1.000          0.044          0.481          0.280
keywords            0.940          0.875          0.567          0.704          0.796
 LAPTOP-5R1DVIR5   D:\Desktop\test-suite-sql-eval-master   17.856s   10:12 AM
 nicholas >> pip install sqlparse
Requirement already satisfied: sqlparse in c:\users\nicholas\anaconda3\envs\lucky\lib\site-packages (0.5.0)
 LAPTOP-5R1DVIR5   D:\Desktop\test-suite-sql-eval-master   1.884s   11:44 AM
 nicholas >> python .\evaluation.py --gold .\evaluation_examples\gold.txt  --pred .\evaluation_examples\GPT4_predict.txt --db .\database\ --table .\tables.json --etype exec

                    easy           medium         hard           extra          all
count               248            446            174            166            1034
=====================   EXECUTION ACCURACY     =====================
execution           0.871          0.836          0.695          0.602          0.783
 LAPTOP-5R1DVIR5   D:\Desktop\test-suite-sql-eval-master   17.428s   12:00 PM
 nicholas >> python .\evaluation.py --gold .\evaluation_examples\gold.txt  --pred .\evaluation_examples\DIN_predict.txt --db .\database\ --table .\tables.json --etype exec
                    easy           medium         hard           extra          all
count               248            446            174            166            1034
=====================   EXECUTION ACCURACY     =====================
execution           0.778          0.726          0.741          0.705          0.738
 LAPTOP-5R1DVIR5   D:\Desktop\test-suite-sql-eval-master   18.002s   12:02 PM
 nicholas >> python .\evaluation.py --gold .\evaluation_examples\gold.txt  --pred .\evaluation_examples\DIN_predict.txt --db .\database\ --table .\tables.json --etype exec
```

详细数据：



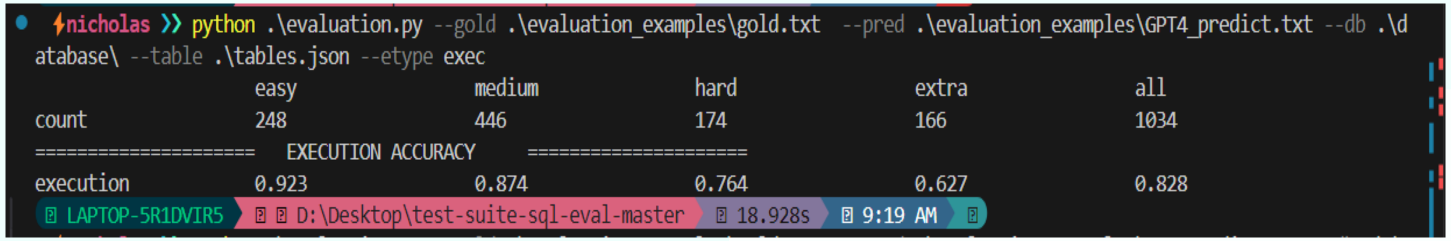
```
                     easy                 medium               hard                 extra                all
count                248                  446                  174                  166                  1034
=====================   EXECUTION ACCURACY     =====================
execution            0.778                0.726                0.741                0.705                0.738

====================== EXACT MATCHING ACCURACY =====================
exact match          0.702                0.639                0.672                0.663                0.663

---------------------PARTIAL MATCHING ACCURACY----------------------
select               0.842                0.861                0.908                0.922                0.874
select(no AGG)       0.842                0.861                0.908                0.922                0.874
where                0.905                0.819                0.870                0.859                0.856
where(no OP)         0.905                0.826                0.883                0.887                0.866
group(no Having)     0.810                0.840                0.725                0.931                0.840
group                0.714                0.830                0.675                0.931                0.818
order                0.520                0.783                0.881                0.963                0.821
and/or               1.000                0.966                0.943                0.969                0.971
IUEN                 0.000                0.000                0.969                0.909                0.944
keywords             0.850                0.874                0.858                0.936                0.878
---------------------- PARTIAL MATCHING RECALL ----------------------
select               0.730                0.697                0.741                0.711                0.715
select(no AGG)       0.730                0.697                0.741                0.711                0.715
where                0.704                0.670                0.713                0.649                0.682
where(no OP)         0.704                0.676                0.723                0.670                0.690
group(no Having)     0.850                0.669                0.744                0.684                0.697
group                0.750                0.662                0.692                0.684                0.679
order                0.591                0.720                0.673                0.658                0.675
and/or               0.988                0.991                1.000                0.969                0.988
IUEN                 0.000                0.000                0.738                0.588                0.671
keywords             0.720                0.698                0.695                0.705                0.703
---------------------- PARTIAL MATCHING F1 --------------------------
select               0.782                0.771                0.816                0.803                0.786
select(no AGG)       0.782                0.771                0.816                0.803                0.786
where                0.792                0.737                0.784                0.739                0.759
where(no OP)         0.792                0.743                0.795                0.764                0.768
group(no Having)     0.829                0.745                0.734                0.788                0.762
group                0.732                0.736                0.684                0.788                0.742
order                0.553                0.750                0.763                0.782                0.741
and/or               0.994                0.978                0.970                0.969                0.979
```

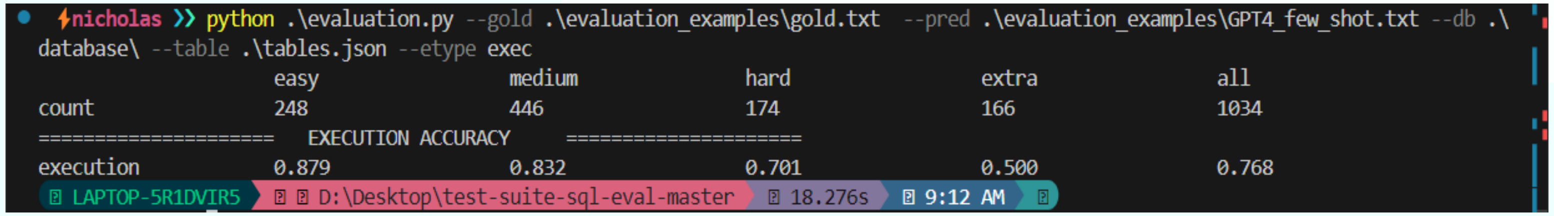
实验结果对比分析：

GPT4_predict：0.828



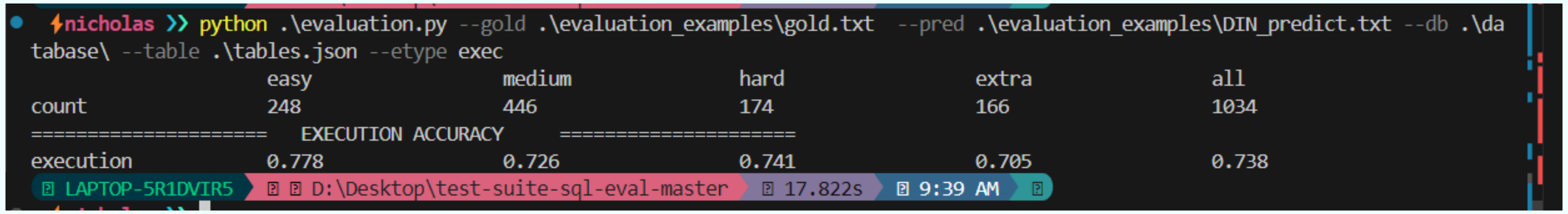
```
● ⚡nicholas ❯❯ python .\evaluation.py --gold .\evaluation_examples\gold.txt  --pred .\evaluation_examples\GPT4_predict.txt --db .\d
atabase\ --table .\tables.json --etype exec
                     easy                 medium               hard                 extra                all
count                248                  446                  174                  166                  1034
=====================    EXECUTION ACCURACY     =====================
execution            0.923                0.874                0.764                0.627                0.828
 □ LAPTOP-5R1DVIR5  □ □ D:\Desktop\test-suite-sql-eval-master  □ 18.928s  □ 9:19 AM  □
```

GPT4_Few_Shot：0.768



```
● ⚡nicholas ❯❯ python .\evaluation.py --gold .\evaluation_examples\gold.txt  --pred .\evaluation_examples\GPT4_few_shot.txt --db .\
database\ --table .\tables.json --etype exec
                     easy                 medium               hard                 extra                all
count                248                  446                  174                  166                  1034
=====================    EXECUTION ACCURACY     =====================
execution            0.879                0.832                0.701                0.500                0.768
 □ LAPTOP-5R1DVIR5  □ □ D:\Desktop\test-suite-sql-eval-master  □ 18.276s  □ 9:12 AM  □
```

DIN-SQL：0.738



```
● ⚡nicholas ❯❯ python .\evaluation.py --gold .\evaluation_examples\gold.txt  --pred .\evaluation_examples\DIN_predict.txt --db .\da
tabase\ --table .\tables.json --etype exec
                     easy                 medium               hard                 extra                all
count                248                  446                  174                  166                  1034
=====================    EXECUTION ACCURACY     =====================
execution            0.778                0.726                0.741                0.705                0.738
 □ LAPTOP-5R1DVIR5  □ □ D:\Desktop\test-suite-sql-eval-master  □ 17.822s  □ 9:39 AM  □
```

THANKS

THANK YOU FOR YOUR WATCHING