Schema Alignment on BIRD dataset

Gianluca Farinaccio [548013]

Advanced Topics in Computer Science - A.A. 2024/2025

GitHub: https://github.com/gianlucafarinaccio/schema-alignment-bird

1 Introduction

This report describes the solution designed and implemented to address the "Big Data Integration" assignment. The objective was to build a system that, for each natural language question in the BIRD benchmark, identifies the Source Tables (STs) containing data relevant to answering the question. Additionally, the system must evaluate the results against the BIRD ground truth by computing the overall recall, precision, and F1 score for the detected STs. The implemented solution primarily relies on LLMs (specifically, Llama-3.3-70b-8192) to automate these steps.

Due to limited resources for LLM inference, the solution was tested on the BIRD mini-dev dataset, which consists of 500 queries from 10 different databases.

2 Problem analysis

Schema alignment is a fundamental challenge in the field of Big Data integration. In this context, the goal is to identify Source Tables (STs) that may contain relevant data to answer a given natural language query. The main challenges in this process include:

- **Schema Variability:** The databases in the BIRD benchmark span multiple domains (e.g., healthcare, events, etc.), each with different naming conventions, structures, and levels of normalization.
- **Ambiguity in Natural Language Queries:** Questions often contain synonyms, vague references, or implicit relationships that do not directly match database table or column names, requiring contextual interpretation.
- **Ground truth extraction:** The BIRD dataset provides SQL queries as ground truth. However, in this case, we need the names of the STs as

ground truth. Therefore, it is necessary to extract only the table names from the complete SQL query.

Successfully addressing these challenges is key to developing an effective schema alignment system that accurately retrieves relevant source tables.

3 Methodology

The general approach used to develop this system is straightforward. It can be summarized as querying an LLM to infer the relevant Source Tables (STs) based on limited information: a natural language question and the corresponding database schema.

Below is a step-by-step summary:

- **Dataset Preprocessing:** Extracting ground truth from SQL queries and reformatting database schemas.
- **Prompt Construction:** Filling a prompt template with the natural language query (NLQ) and the corresponding database schema.
- **LLM Inference:** Sending an API request to the LLM.
- **Postprocessing LLM Outputs:** Correcting errors related to case sensitivity.
- **Metrics Evaluation:** Computing precision, recall, and F1-score.

3.1 Dataset Preprocessing

This first step focuses on dataset preprocessing. The BIRD mini-dev dataset provides two JSON files containing:

- A list of natural language questions (NLQs) along with their corresponding SQL queries (the ground truth).
- A verbose representation of the database schemas.

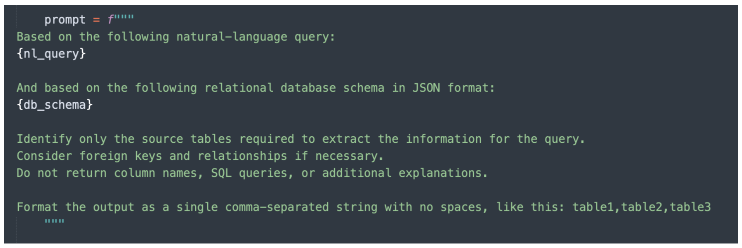
In this preprocessing step, I extracted the names of the Source Tables (STs) from the SQL queries and reformatted the database schemas. As mentioned above, the BIRD dataset provides SQL queries as ground truth. However, in this context, I need the names of the STs as ground truth, so they must be extracted from the queries.

Table name extraction was performed using a function that matches a regex pattern. However, this approach failed for complex queries that included certain SQL keywords in the solution. To mitigate these errors, I added an extra step to verify that the extracted table names are also present in the database schema. For code details, refer to the **src/utils.py - extract_tables_from_sql_v2** function.

The database schema file in the BIRD dataset has a verbose structure. I decided to reformat it to obtain a simpler and cleaner representation, making it more suitable for LLM processing in later steps. In the new format, I included only table names, attribute names, and their types, resulting in a new JSON file. For further details, refer to the **src/utils.py - format_db_schema** function and the **dataset/db_schema.JSON** file.

3.2 Prompt construction

This step focuses on prompt construction. After several attempts, I decided to use the prompt shown in Figure 1. For each dataset entry, the variables **{nl_query}** and **{db_schema}** were replaced with a natural language query and the corresponding database schema. As shown in the figure, the desired output format is explicitly specified within the prompt.



```
    prompt = f"""
Based on the following natural-language query:
{nl_query}

And based on the following relational database schema in JSON format:
{db_schema}

Identify only the source tables required to extract the information for the query.
Consider foreign keys and relationships if necessary.
Do not return column names, SQL queries, or additional explanations.

Format the output as a single comma-separated string with no spaces, like this: table1,table2,table3
    """
```

Figure 1: Prompt template

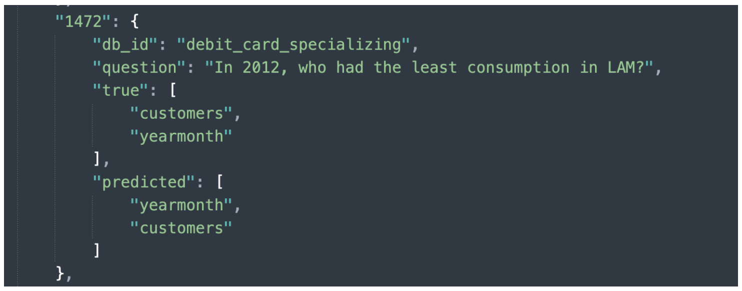
This approach can be considered zero-shot prompting since no examples of correct or incorrect answers are provided within the prompt.

3.3 LLM inference

As stated in the introduction, this solution primarily relies on an LLM to achieve the goal. Due to limited resources, the implementation and testing were carried out using the Groq Free API tier. Based on this constraint, the model used in this case is Llama-3.3-70b-8192, as it allows the highest number of free requests per minute/day via Groq. However, I also consider a 70B model to be a solid baseline for evaluating the solution.

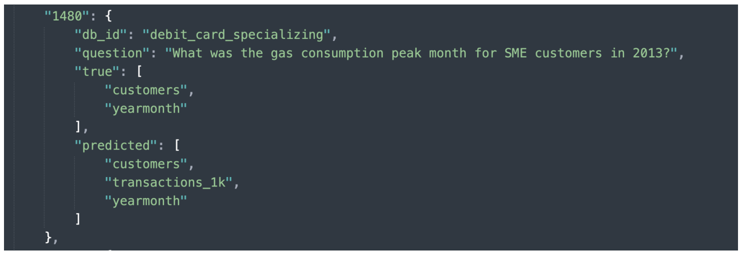
The source code was designed to run within the limitations of the Groq free tier, which imposes a mandatory delay of a few seconds between requests to avoid exceeding the rate limit. The execution on the BIRD mini-dev dataset, which contains 500 queries, takes approximately 20 minutes to complete. After execution, the system generates an output JSON file that, for each query, includes the predicted STs names and the ground truth reference.

Figures 2 and 3 show two prediction examples. As seen in the first figure, the prediction is correct because the STs names in the "predicted" field exactly match those in "true." Conversely, in the second example, the query was not correctly predicted by the LLM, as the predicted and true names do not match.



```
"1472": {
    "db_id": "debit_card_specializing",
    "question": "In 2012, who had the least consumption in LAM?",
    "true": [
        "customers",
        "yearmonth"
    ],
    "predicted": [
        "yearmonth",
        "customers"
    ]
},
```

Figure 2: Output file - correct prediction



```
"1480": {
    "db_id": "debit_card_specializing",
    "question": "What was the gas consumption peak month for SME customers in 2013?",
    "true": [
        "customers",
        "yearmonth"
    ],
    "predicted": [
        "customers",
        "transactions_1k",
        "yearmonth"
    ]
},
```

Figure 3: Output file - wrong prediction

Finally, to mitigate errors related to case sensitivity, the system converts all LLM responses to lowercase, to ensure that "predicted" and "true" fields of JSON file contain only lowercase STs names.

4 Results

In this final step, I will discuss the evaluation of the metrics used to assess the described solution. The metrics.py script calculates Precision, Recall, and F1-score based on a JSON file provided as input (the same file mentioned earlier, which contains the "true" and "predicted" fields).

The repository includes a directory named **"bench_25032025"**, which contains the output files and the corresponding calculated metrics. The results seem promising for a simple schema alignment system like this, achieving the following scores:

- Precision: 0.68
- Recall: 0.77
- F1-score: 0.72

However, there are databases where the metrics are significantly lower compared to others. In this benchmark, for instance, **formula_1** and **debit_card_specializing** have scores below 0.5.

For all Q&A interactions with the LLM and the generated output file related to this benchmark, please refer to **"bench_25032025/output_completed.JSON"**.

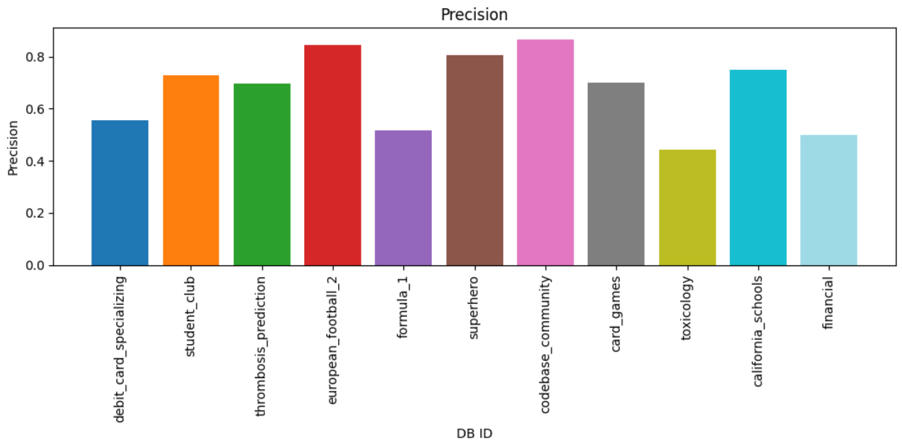


Figure 4: Metrics - Precision

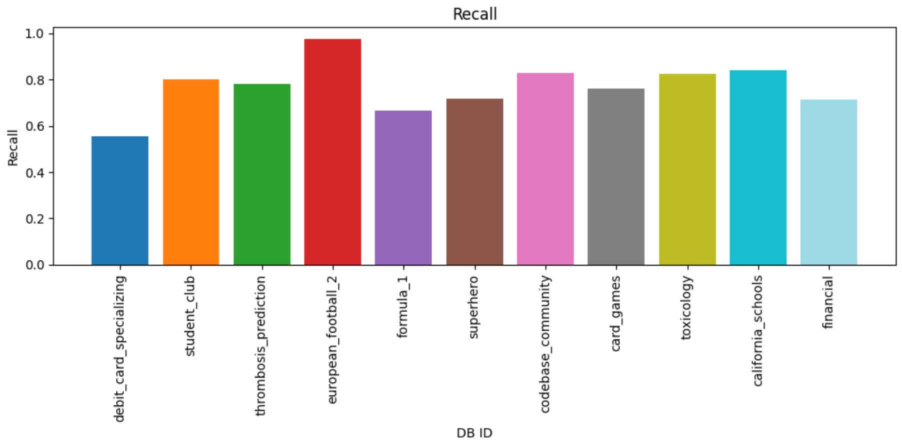


Figure 5: Metrics - Recall

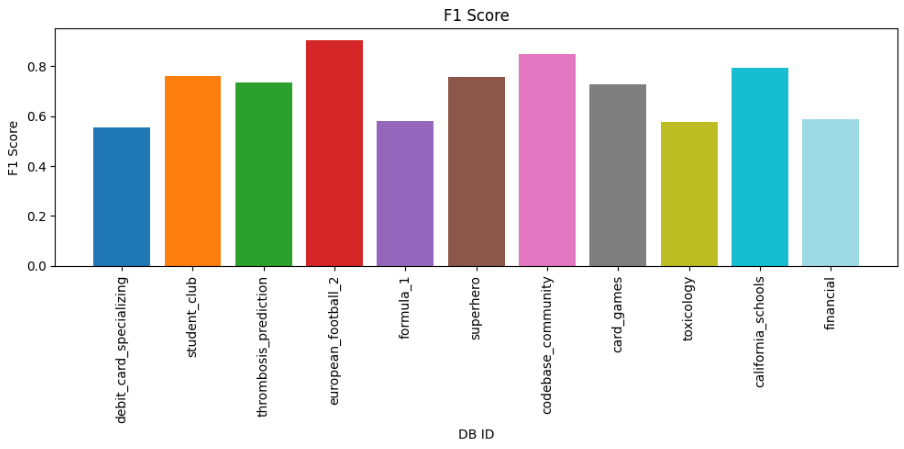


Figure 6: Metrics - F1_score