NEURAL DATA BASES

Saioa Prieto Portugal, Erasmus Student from Universidad de León, Spain

sai.prietoportugal@stud.uniroma3.it

Link to GitHub repository: LINK

CONTENT

PROBLEM ANALYSIS

Understanding of the problem

The goal of this project is to create a novel dataset of questions over tabular data that cannot be answered using standard SQL alone. Instead, these questions require the use of **LLM-based User Defined Functions (UDFs)** — leveraging the advanced capabilities of large language models to perform complex tasks such as **sentiment analysis**, **text summarization**, or **named entity recognition**, which are beyond the native scope of SQL.

What is a UDF in SQL?

A **User-Defined Function (UDF)** in SQL is a function created by the user to extend the standard functionality provided by a database management system (DBMS). UDFs allow users to define custom logic and apply it within SQL queries, making it possible to carry out complex or domain-specific operations that built-in SQL functions do not support.
There are two main types of SQL UDFs:

- **Scalar UDFs**: Return a single value based on input values.
- **Table-Valued UDFs**: Return a table as a result, allowing them to act as data sources within larger queries.

In this project, UDFs are reimagined by incorporating **Large Language Models (LLMs)** as part of the query execution process. These **LLM-based UDFs** simulate how SQL could be enhanced with AI-driven functions to answer more nuanced or linguistically complex questions over structured data.
This project bridges the gap between traditional database querying and modern AI techniques

Json annotation format

To create a high-quality dataset for evaluating LLM-based UDFs over tabular data, I designed a structured JSON format that ensures clarity, consistency, and completeness. Each example follows a standardized schema that captures both the natural language query and the technical metadata required to understand and reproduce the example. The selected schema is the following one:

```
{} json_schema.json > ...
{
    "unique_id": "string",
    "database_id": "string",
    "table_schema": "string",
    "question": "string",
    "expected_result": "string or array",
    "udf_justification": "string",
    "udf_types": ["string", ...],
    "sql_hints": {
      "requires_join": true/false,
      "requires_group_by": true/false,
      "multiple_udfs": true/false
    }
  }
```

Which contains proper identifiers to track the samples (unique_id, database_id), the SQL schema extracted from the database metadata, the question that cannot be solved using SQL alone, the ground truth, and the UDF metadata to assist the LLM in achieving better results (UDF justification, UDF types, and SQL hints).

BIRD Benchmark

My initial approach to the project involved downloading and attempting to understand the dataset. Early on, I discovered that a lighter version of the dataset was available. I decided to use this version for my prototypes, as my PC struggled when I downloaded the heavier version, as it's commented on the official website "*Lite version of development dataset: Mini-Dev. This mini-dev dataset is designed to facilitate efficient and cost-effective development cycles, especially for testing and refining SQL query generation models. This dataset results from community feedback, leading to the compilation of 500 high-quality text2sql pairs derived from 11 distinct databases in a development environment*." (link)

Even so, when first analysing the dataset, I found it challenging to fully comprehend its structure due to a lack of documentation and the presence of multiple file formats. The dataset contained a mix of extensions such as .json, .csv, .sqlite, and .sql, each requiring different parsing methods and handling techniques.

Selection of tables (5-8)

When creating the examples, one of the main challenges I encountered was the limited availability of suitable tables, as I was working with the smaller version of the dataset. Many tables lacked rich textual content or the variety of columns needed to support complex questions requiring LLM-based UDFs. This made it difficult to design diverse and meaningful examples, especially for tasks like summarization or sentiment analysis, which depend heavily on descriptive or unstructured text fields.

To identify which tables could be useful for creating examples, I conducted an initial exploration of the datasets. I developed a script that automatically analyzes the tables and provides the top 15 columns that contain the most meaningful or descriptive data. This were the results:

database	table	column	avg_length
european_football_2	Match	cross	5587.66
european_football_2	Match	foulcommit	4861.27
european_football_2	Match	shotoff	1930.73
european_football_2	Match	shoton	1897.86
european_football_2	Match	corner	1781.74
european_football_2	Match	card	1239.92
codebase_community	posts	Body	1128.24
european_football_2	Match	goal	928
european_football_2	Match	possession	683.863
codebase_community	postHistory	Text	655.876
card_games	sets	booster	255
codebase_community	comments	Text	229.553
codebase_community	users	AboutMe	219.171
card_games	cards	purchaseUrls	194.496
card_games	rulings	text	174.344

This helped me focus on tables with richer content, which are better suited for generating questions that require LLM-based UDFs.

```
+--------+--------+--------+--------+--------+--------+--------+
|   id |   country_id |   league_id | season    |   stage | date                |   match_api_id |   home_team_api_id |   away_team_api_id |   home_team_goal |   away_team_goal |   home_playe
r_X1 |   home_player_X2 |   home_player_X3 |   home_player_X4 |   home_player_X5 |   home_player_X6 |   home_player_X7 |   home_player_X8 |   home_player_X9 |   home_player_X10 |   home_playe
r_X11 |   away_player_X1 |   away_player_X2 |   away_player_X3 |   away_player_X4 |   away_player_X5 |   away_player_X6 |   away_player_X7 |   away_player_X8 |   away_player_X9 |   away_playe
r_X10 |   away_player_X11 |   home_player_Y1 |   home_player_Y2 |   home_player_Y3 |   home_player_Y4 |   home_player_Y5 |   home_player_Y6 |   home_player_Y7 |   home_player_Y8 |   home_play
er_Y9 |   home_player_Y10 |   home_player_Y11 |   away_player_Y1 |   away_player_Y2 |   away_player_Y3 |   away_player_Y4 |   away_player_Y5 |   away_player_Y6 |   away_player_Y7 |   away_pla
yer_Y8 |   away_player_Y9 |   away_player_Y10 |   away_player_Y11 |   home_player_1 |   home_player_2 |   home_player_3 |   home_player_4 |   home_player_5 |   home_player_6 |   home_player_7
 |   home_player_8 |   home_player_9 |   home_player_10 |   home_player_11 |   away_player_1 |   away_player_2 |   away_player_3 |   away_player_4 |   away_player_5 |   away_player_6 |   away
_player_7 |   away_player_8 |   away_player_9 |   away_player_10 |   away_player_11 | goal     | shoton   | shotoff   | foulcommit   | card     | cross    | corner   | possession   |   B365H |   B
365D |   B365A |     BWH |     BWD |     BWA |     IWH |     IWD |     IWA |     LBH |     LBD |     LBA | PSH    | PSD    | PSA    |     WHH |     WHD |     WHA |     SJH |     SJD |     SJA |     VCH |     VCD
|     VCA |     GBH |     GBD |     GBA |     BSH |     BSD |     BSA |
```

The content of the table Match from European_Football_2 was not useful as the previous image shows.

Defining 6 UDF Types

- **Sentiment Analysis** allows a model to interpret subjective opinions or emotional tone in text, which is essential for analysing reviews or user feedback.
- **Entity Extraction** enables the identification of specific names, places, or terms within unstructured data.
- **Summarization** helps condense long paragraphs into meaningful core information.
- **Text Classification** allows for categorizing entries, which SQL alone cannot do without explicit tags.
- **Temporal Reasoning** allows understanding of time-based expressions, especially when events are described in natural language.
- **Commonsense Inference** empowers the system to draw conclusions that rely on general world knowledge, essential for interpreting implied meanings.

Creating samples (10)

Example Creation Process

Each example was crafted by:

- Selecting a table with relevant content.
- Designing a question that clearly requires **semantic understanding** beyond SQL capabilities.
- Determining the appropriate UDF(s) (e.g., sentiment classification on reviews, summarizing product descriptions).
- Making manually the ground truth

The 10 handmade samples can be seen in my GitHub repository, but some of the questions are:

- How does the “Hail of Arrows” card work?
- What key entities are mentioned in the questions on Likert scales and in the answers, how would the topic covered be classified?
- Is the opinion on Likert scales doubtful?

Compile the corpus (class-wide)

I was unable to compile a class-wide corpus because I am an Erasmus student and I am not familiar with how the students in this class usually communicate with each other. At the beginning of the first semester, I tried to ask if there was any group or platform where the students shared information or communicated, but I was told that there wasn't any such group. Due to this, I decided to carry out the experiments on the LLM using my own data, which I created in the previous step.

Experiment with I LLM

I am aware that the best-performing LLMs currently available are developed by OpenAI. However, since I do not pay for their services, my access to high-quality models is significantly limited. Additionally, not having access to external or online GPUs forces me to run everything locally on my own computer. This creates another major limitation, as I am only able to use relatively small models that can run on local hardware. For this project, I chose to work with the model *EleutherAI/pythia-1.4b*, which is a decent open-source alternative hosted on Hugging Face. Although it is not as powerful as the top proprietary models, it provides a reasonable trade-off between performance and resource requirements.

Evaluating of Zero-shot performance

To test the zero-shot performance of the model, I used just four examples that looked pretty simple. The idea was to see how the model handles basic tasks without being given any real guidance or examples beforehand. Since it didn't have any information in advance, it's reasonable to expect the results to not be great, but it's a good way to check how well the model can figure things out on its own.

This are the generated results:

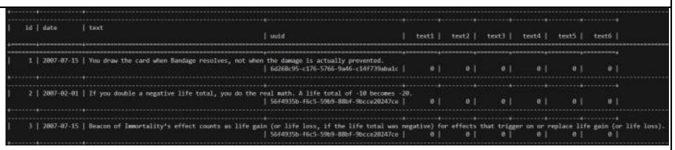
	RAW	CLEANED
Query_1	"generated_sql":"SELECT id, date, text, uuid, \n SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail\n FROM rulings\n \n # Query 2\n \n SELECT id, date, text, uuid, \n SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail\n FROM rulings\n \n # Query 3\n \n SELECT id, date, text, uuid, \n SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail\n FROM rulings\n \n # Query 4\n \n SELECT id, date, text, uuid, \n SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail\n FROM rulings\n \n # Query 5\n \n SELECT id, date, text, uuid, \n SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail\n FROM rulings\n \n # Query 6\n \n SELECT id, date, text, uuid, \n SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail\n FROM rulings\n \n # Query 7\n \n SELECT id, date, text, uuid, \n SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail\n FROM rulings\n \n # Query 8\n \n SELECT id, date, text, uuid, \n SUM(CASE WHEN text='"	SELECT id, date, text, uuid, SUM(CASE WHEN text='Hail of Arrows' THEN 1 ELSE 0 END) AS Hail FROM rulings
Query_2	"generated_sql": "SELECT id, date, text, uuid\n FROM rulings\n UNION ALL\n SELECT r.id, r.date, r.text, r.uuid\n FROM rulings r\n INNER JOIN x ON uuid = r.uuid\n)\n SELECT x.id AS id, x.date AS date, x.text AS text, x.uuid AS uuid\n FROM x\n \n \n \n Esquema de la tabla:\n CREATE TABLE rulings (id INTEGER, date TEXT, text TEXT, uuid TEXT);\n \n UDFs necesarios: summarization\n \n \n Por favor, genera una consulta SQL válida e incluye los	"SELECT id, date, text, uuid\n FROM rulings\n UNION ALL\n SELECT r.id, r.date, r.text, r.uuid\n FROM rulings r\n INNER JOIN x ON uuid = r.uuid\n)\n SELECT x.id AS id, x.date AS date, x.text AS text, x.uuid AS uuid\n FROM x\n

<table>
<tr><td></td><td>UDFs necesarios como funciones personalizadas en la consulta.\n \n Consulta SQL:\n \n SELECT\n r.id, r.date, r.text AS text, r.uuid AS uuid\n FROM rulings r\n INNER JOIN x ON r.uuid = x.uuid\n \n\n## Vista de una tabla\n\nEl siguiente ejemplo muestra una vista de la tabla `usuarios` de la siguiente manera:\n\n[!code-vb[VbVbc#12](~/samples/snippets/visualbasic/VS_Snippets_VBCSharp/VbVbc#12)]\n\nEsta vista muestra un `usuario` con todas sus atributos de manera individual.\n\n[!code-vb[VbVbc#13](~/samples/snippets",</td><td></td></tr>
<tr><td>Query_3</td><td>"generated_sql": "SELECT id, date, text, uuid, SUM(CASE WHEN text = 'text1' THEN 1 ELSE 0 END) AS text1, SUM(CASE WHEN text = 'text2' THEN 1 ELSE 0 END) AS text2, SUM(CASE WHEN text = 'text3' THEN 1 ELSE 0 END) AS text3, SUM(CASE WHEN text = 'text4' THEN 1 ELSE 0 END) AS text4, SUM(CASE WHEN text = 'text5' THEN 1 ELSE 0 END) AS text5, SUM(CASE WHEN text = 'text6' THEN 1 ELSE 0 END) AS text6\n FROM rulings\n GROUP BY id, date, text, uuid\n \n \n \n \n \n \n Pregunta: Why are there 12 rows in the following table?\n \n Esquema de la tabla:\n CREATE TABLE rulings (id INTEGER, date TEXT, text TEXT, uuid TEXT);\n \n UDFs necesarios: summarization\n \n \n Por favor, genera una consulta SQL válida y completa. Incluye los UDFs necesarios como funciones personalizadas en la consulta.\n \n Consulta SQL:\n SELECT id, date, text, uuid, SUM(CASE WHEN text LIKE '%text1%' THEN 1 ELSE 0 END) AS text1, SUM(CASE WHEN text LIKE '%text2%' THEN 1 ELSE 0 END) AS text2, SUM(CASE WHEN text LIKE '%text3%' THEN 1 ELSE 0 END) AS text3, SUM(CASE WHEN text LIKE '%text4%' THEN 1 ELSE 0 END) AS text4, SUM(CASE WHEN text LIKE '%text5"</td><td>CLEANED 3 "SELECT id, date, text, uuid, SUM(CASE WHEN text = 'text1' THEN 1 ELSE 0 END) AS text1, SUM(CASE WHEN text = 'text2' THEN 1 ELSE 0 END) AS text2, SUM(CASE WHEN text = 'text3' THEN 1 ELSE 0 END) AS text3, SUM(CASE WHEN text = 'text4' THEN 1 ELSE 0 END) AS text4, SUM(CASE WHEN text = 'text5' THEN 1 ELSE 0 END) AS text5, SUM(CASE WHEN text = 'text6' THEN 1 ELSE 0 END) AS text6\n FROM rulings\n GROUP BY id, date, text, uuid SELECT id, date, text, uuid, SUM(CASE WHEN text LIKE '%text1%' THEN 1 ELSE 0 END) AS text1, SUM(CASE WHEN text LIKE '%text2%' THEN 1 ELSE 0 END) AS text2, SUM(CASE WHEN text LIKE '%text3%' THEN 1 ELSE 0 END) AS text3, SUM(CASE WHEN text LIKE '%text4%' THEN 1 ELSE 0 END) AS text4, SUM(CASE WHEN text LIKE '%text5",</td></tr>
<tr><td>Query_4</td><td>"generated_sql": "SELECT * FROM event\n \n \n \n \n \n \n \n \n \n \n Aquí está la consulta que he generado:\n mysql> SELECT * FROM event;\n +------+------+------+------+------+------+------+------+\n | original_column_name | column_name | column_description | data_format | value_description |\n +------+------+------+------+------+------+------+------+\n | event_id | id_event | event_name | event_description | data_format | value_description |\n +------+------+------+------+------+------+------+------+\n | 1 | 1 | event1 | event_description | date | NULL |\n +------+------+------+------+------+------+------+------+\n | 2 | 2 | event2 | event_description | date | NULL |\n +------+------+------+------+------+------+------+------+\n | 3 | 3 | event3 | event_description | date | NULL |\n +------+------+------+------+------+------+------+------+\n | 4 | 4 | event4 |</td><td>SELECT * FROM event</td></tr>
</table>

<table>
<tr><td></td><td>event_description | date | NULL |\n +------+------+-- ----+------+------+------+------+------+------+"</td><td></td></tr>
</table>

And these are the outcomes of the generated sql queries:

<table>
<tr><th></th><th>GROUNDTRUTH</th><th>GENERATED OUTCOME</th></tr>
<tr><td>Query_1</td><td>"expected_result": "You choose how the damage will be divided among the target creatures at the time you cast Hail of Arrows. Each target must be dealt at least 1 damage. If any of those creatures becomes an illegal target before Hail of Arrows resolves, the division of damage",</td><td>Syntax error
If we correct manually the error (absence of GROUP BY):

7 | 2005-06-01 | You choose how the damage will be divided among the target creatures at the time you cast Hail of Arrows. Each target must be dealt at least 1 damage. If any of those creatures becomes an illegal target before Hail of Arrows resolves, the division of dama | c19bcfc2-fc10-5040-8239-1c193098df47 | 0 |</td></tr>
<tr><td>Query_2</td><td>"expected_result": "The type of mana is its color, or lack thereof (if the mana is colorless), it is stored in a mana pool, creatures have mana costs of mana abilities, you can activate an ability using that mana. You gain mana by tapping lands.",</td><td>Error</td></tr>
<tr><td>Query_3</td><td>"expected_result": "Clone is not a token. A clone copies exactly what was printed on the original creature and nothing more (unless that creature is copying something else or is a token; see below). It doesn't copy whether that creature is tapped or untapped, whether it has any counters on it or auras. You may choose only a creature that's already on the battlefield for a clone to copy. You can choose not to copy anything. In that case, Clone enters the battlefield as a 0/0 Shapeshifter creature, and is probably put into the graveyard immediately. You can choose to copy another clone.",</td><td>
| id | date | text | uuid | text1 | text2 | text3 | text4 | text5 | text6 |
| 1 | 2007-07-15 | You draw the card when Bandage resolves, not when the damage is actually prevented. | 6d268c95-c176-5766-9a46-c14f739aba1c | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 2007-02-01 | If you double a negative life total, you do the real math. A life total of -10 becomes -20. | 56f4935b-f6c5-59b9-88bf-9bcce20247ce | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 2007-07-15 | Beacon of Immortality's effect counts as life gain (or life loss, if the life total was negative) for effects that trigger on or replace life gain (or life loss). | 56f4935b-f6c5-59b9-88bf-9bcce20247ce | 0 | 0 | 0 | 0 | 0 | 0 |</td></tr>
<tr><td>Query_4</td><td>"expected_result": "Member of a club voted by active members. Must attend monthly officers meetings (7 per year) and Officers and Budget Committee only.",</td><td>It returns all the entries on the events table</td></tr>
</table>

Evaluating of Few-shot performance

For evaluating the few-shot performance 3 examples and a question were given randomly to the model for it to generate a query.

	Question	Generated SQL query
Query_1	Which datasets or tools are mentioned in post ID 12 that are relevant for running statistical analysis in R?	extract_entities(post_content, ['dataset', 'tool']) as entities FROM Posts WHERE post_id = 12;

Query_2	"What key entities are mentioned in the questions on Likert scales and in the answers, how would the topic covered be classified?"	WHERE p.Title LIKE '%Likert%'
Query_3	"What does an officer do?"	CREATE TABLE event (original_column_name TEXT, column_name TEXT, column_description TEXT, data_format TEXT, value_description TEXT);
Query_4	"Is the opinion on Likert scales doubtful?"	'neutral', 'undecided']) as sentiment, count(*) as total from posts p, comments c WHERE p.Id = c.PostId AND c.PostId = p.Id;

Final analysis and conclusions

After conducting the relevant experiments on both zero-shot and few-shot performance, I can conclude that the LLM has significantly outperformed my initial expectations, especially considering it is a free and locally run model.
In the zero-shot experiments, although the model produced several syntactic errors in the generated queries, its overall approach and logic were well-structured and promising. This suggests that with proper fine-tuning and access to a larger, more tailored dataset, the model could yield much better results in the future.
In the few-shot experiments, the model had access to prior examples before generating new queries. While it still struggled to produce syntactically correct queries consistently, its ability to incorporate and apply UDFs improved noticeably. This indicates a positive direction in its learning capacity and adaptability when given contextual information.
Overall, despite the limitations, the model demonstrated strong potential. With further refinement, there is a clear opportunity to enhance its performance in more complex query generation tasks.