User Define Functions to improve limitations introduced by SQL language

Alessandro Di Girolamo (546950), Gianluca Farinaccio (548013),
Marco Napoleone (566041), Daniele Ferneti (546599),
Elia Guglielmi (533629)

Project for Neural Table Rapresentation course
May 2025

Keywords: LLM, Neural Table Rapresentation, Compuret Science, Prompt Engineering, SQL, Text-to-SQL

1. Introduction

Relational databases are a powerful and efficient tool, widely adopted across a broad range of systems and applications. Users may issue a variety of requests, and the ability to accurately translate a natural language query into a valid SQL statement is critical for retrieving and manipulating the data stored within these systems. While SQL provides robust capabilities for data querying and manipulation, more complex user intents (particularly those requiring semantic understanding of textual content within the database) often require human intervention.
With the emergence of Large Language Models (LLMs), which excel in semantic text analysis and natural language understanding, new possibilities have arisen for enhancing traditional SQL-based interactions. Specifically, LLMs can be leveraged to extend the expressive power of SQL by generating user defined functions (UDFs) that perform tasks beyond the native capabilities of SQL (such as text classification, summarization, and other forms of natural language processing).
This paper outlines the motivations behind this approach, discusses existing limitations of current systems, presents the implementation details of the proposed solution, and analyzes the results obtained through empirical evaluation.

2. Problem Analysis

As outlined in the previous section, SQL is a powerful and scalable language for working with relational databases. However, users may encounter scenarios that require more

sophisticated operations tasks (that go beyond the declarative and rule-based structure of SQL).
For instance, consider a user who wants to perform text classification based on semantic content, or who wishes to summarize customer reviews of a product. While SQL allows the selection and filtering of subsets of data tuples, it lacks the inherent ability to understand or manipulate unstructured textual content at a semantic level. These types of operations demand capabilities that traditional SQL engines were not designed to handle.
To address this limitation, SQL can be complemented by Large Language Models (LLMs), which are capable of interpreting and generating natural language, as well as performing complex tasks such as classification, summarization, and sentiment analysis. This hybrid approach aims to extend the inferential capabilities of relational databases by integrating the semantic reasoning power of LLMs.
However, this integration introduces several challenges. The use of LLMs may lead to inconsistent or inaccurate outputs due to their probabilistic nature. Additionally, the computational cost is significantly higher compared to traditional SQL queries, and the solution may not scale as efficiently as pure SQL-based approaches. These trade-offs must be carefully considered when designing systems that aim to blend the reliability of relational databases with the flexibility of language models.

3. User Define Functions

The experiment described in this work aims to evaluate the ability of Large Language Models (LLMs) to address semantically complex tasks typically unmanageable through standard SQL operations. The proposed solution involves the use of User-Defined Functions (UDFs), which serve as modular extensions capable of integrating LLM-based reasoning within relational database queries.
This section presents five representative UDFs, each designed to solve a specific use case where semantic understanding is essential. For each UDF, we describe the underlying scenario, the structure and content of the data involved, and the prompts used to guide the behavior of the LLM. Additionally, we outline the design choices made to optimize the quality and consistency of the model's responses, such as prompt engineering techniques and input formatting strategies.
The results obtained from each UDF are discussed to assess their effectiveness and limitations. This analysis highlights the practical implications of integrating LLMs into database workflows and provides insight into best practices for constructing robust LLM-augmented UDFs.

3.1 UDF: Classification

3.1.1 Methodology

The user-defined function (UDF) named **classification** was designed with the purpose of assigning the records of a table to one of the predefined classes, based on the context of that table. Starting from the BIRD dataset, five tables containing suitable records for a binary classification task were identified. Once the tables were selected, the two possible classes for each were defined, along with the creation of the ground-truth.

The tables were chosen to span a variety of domains, including geographic knowledge, medical information, natural language understanding and more. This type of processing goes beyond the capabilities of standard SQL, which cannot natively perform these complex operations. As result of these steps we obtained five examples on different tables, were each is composed by twenty records manually selected. Below are the tables and the classes to which each record can be associated:

DB.Table	**Column**	**Info**
codebase.comments	text	`[question, comment]`. If the text contains at least one question, it is considered a `question`; otherwise, it is not.
codebase.users	location	`[coastal-city, non coastal-city]`. If the city is located near the sea, it is considered `coastal-city`; otherwise, it is not.
superhero.superhero	full-name	`[american name, not-american name]`. If the name has American origins, it is considered `american name`; otherwise, it is not.
trombosis.patient	diagnosis	`[rheumatoid-arthritis, other diagnosis]`. If the diagnosis acronym refers to rheumatoid arthritis, the record is assigned to that class; otherwise, it is not.
card_games.cards	originaltext	`[player, opponents]`. If the card's power requires an action by the player who owns the card, it is considered `player`; otherwise, it is not.

Table 1: Tables, columns and classification details

With this setup, two LLMs — **claude-sonnet-4-20250514** and **gpt-4o** — were used to perform binary classification on the records. The prompt was built using the **question** field from the **classification-example.json** file, along with the corresponding table data in CSV format. Given the limited scope of the task, the prompts and the analysis of the LLMs' responses were carried out manually using their respective web interfaces. The results were stored in the same JSON file as above.

3.1.2 Results

In this section we analyze the result of both selected LLMs on the above descripted benchmark. The results are available in JSON file **classification-results**. In the first task, which involved identifying comments that do not contain questions, Claude 4 Sonnet achieved perfect performance, correctly identifying all relevant records without any false positives or negatives. GPT-4o, while equally precise in its predictions, missed a third of the relevant entries, leading to a lower recall. This suggests that while GPT-4o was cautious and accurate in what it labeled, it was also more conservative, possibly overlooking ambiguous cases.

The second task required identifying users located in coastal cities. Both models performed reasonably well, but GPT-4o outperformed Claude 4 slightly by achieving a better balance between precision and recall. Claude 4 showed a tendency toward higher precision but suffered in recall, implying a stricter, more risk-averse classification strategy.

Task three focused on cultural and linguistic inference—specifically, recognizing superheroes whose names are not of American origin. This is inherently subjective and less well-defined, and the models' performances reflected this ambiguity. GPT-4o cast a wider net, identifying more relevant instances (higher recall), but at the cost of precision. Claude 4, on the other hand, predicted fewer total instances but made fewer mistakes proportionally, resulting in higher precision but lower recall.

In the fourth task, which involved identifying patients diagnosed with rheumatoid arthritis, both models performed flawlessly. This suggests that when the classification criteria are relatively well-bounded and the signal is strong—even if the content is technical—both GPT-4o and Claude 4 are capable of high accuracy.

The final task involved determining whether a card's action targets the opponent, which required some semantic understanding of text descriptions. Here, Claude 4 Sonnet again edged out GPT-4o, showing perfect precision and a higher F1 score, although GPT-4o also delivered solid, balanced performance.

Overall, Claude 4 Sonnet tends to be more precise, favoring correct predictions over coverage. This makes it well-suited for use cases where false positives are costly or undesirable. GPT-4o, by contrast, demonstrates a more recall-oriented strategy, often capturing more of the correct instances but occasionally introducing some noise. In conclusion, both models demonstrate strong capabilities, but their strengths diverge in meaningful ways. Claude 4 Sonnet is preferable for tasks requiring careful judgment and high precision, while GPT-4o is better suited for scenarios where it's critical to retrieve as many relevant results as possible. The choice between them should ultimately be guided by the priorities of the specific use case.

ID	Modello	Precision	Recall	F1 Score
classification_001	GPT-4o	1.00	0.6667	0.80
	Claude 4 Sonnet	1.00	1.00	1.00
classification_002	GPT-4o	0.75	0.75	0.75
	Claude 4 Sonnet	0.7143	0.625	0.6667
classification_003	GPT-4o	0.50	0.80	0.6154
	Claude 4 Sonnet	0.6667	0.40	0.50
classification_004	GPT-4o	1.00	1.00	1.00
	Claude 4 Sonnet	1.00	1.00	1.00
classification_005	GPT-4o	0.75	0.75	0.75
	Claude 4 Sonnet	1.00	0.75	0.8571
Media (Macro)	GPT-4o	**0.80**	**0.7933**	**0.7831**
	Claude 4 Sonnet	**0.8761**	**0.775**	**0.8056**

Table 2: Comparison of results metrics between GPT-4o and Claude 4 Sonnet

3.2 UDF: GroupBy

3.2.1 Methodology

In this work, the **BIRD Benchmark (MiniDev version)** was explored, focusing on tasks where standard SQL is insufficient to group data meaningfully. Specifically, the objective was to employ **User Defined Functions (UDFs)** to simulate human-like semantic grouping based on natural language, geographical inference, or domain knowledge.

Unlike traditional SQL group-by operations that rely solely on exact field matches, many real-world scenarios require a deeper understanding of the data—such as recognizing that "Berlin" implies "Germany", or that a user's job title "PhD student in AI" suggests a role related to academia. These are inherently *semantic groupings* that involve interpretation, disambiguation, and context awareness.

To address these limitations, UDFs were designed to bridge the gap between raw text and conceptual categories, enabling group-by operations to capture latent meaning not directly encoded in the database schema. This approach aims to approximate human reasoning in classification tasks that span geography, profession, language style, prognosis, and domain-specific taxonomies.

The tables analyzed include:

- `location_users.csv` – cities to countries
- `aboutme_users.csv` – user bios to professional roles
- `superpower_power.csv` – power names to elemental categories
- `name_superhero.csv` – hero names to publishers
- `diagnosis_patient.csv` – diagnoses to prognosis and severity
- `text_cards.csv` – card text to functionality

- `text_comments.csv` – comment text to language style

Minimal preprocessing was performed (e.g., filling a few missing values in `location` and `aboutme`) to preserve data integrity. Each task was expressed via a natural-language "question" field in a shared JSON format, alongside the database schema, a manually curated `expected_result`, and predictions from two LLMs: **ChatGPT** and **Claude 4 Sonnet**.

3.2.2 Results

The results from both models were compared to the `expected_result`, focusing on:

- **Coverage**: Are all IDs included?
- **Semantic correctness**: Are the groupings conceptually valid?
- **Consistency**: Does the model generalize across tasks?

Task ID	Description	Claude 4 Sonnet	ChatGPT	Notes
groupby_001	Location to Country	**100%**	Partial	Claude included all IDs
groupby_002	Role from About Me	Overgeneralizes	**Accurate**	Claude misclassified 1-2 roles
groupby_003	Superpower Grouping	No 'Other'	**Robust**	ChatGPT handled outliers
groupby_004	Publisher from Hero	Missed Marvel	**Aligned**	GPT classified ambiguous names better
groupby_005	Prognosis Category	**Correct**	**Correct**	Both aligned with manual data
groupby_006	Diagnosis Severity	Inconsistent	**Correct**	Claude confused medium/high
groupby_007	Card Functionality	Misplaced IDs	**Balanced**	GPT preserved granularity
groupby_008	Language Style	**Accurate**	**Accurate**	Both matched register labels

Table 3: Summary of UDF groupBy task accuracy per model.

In summary:

- **Claude 4 Sonnet** performed well in tasks requiring linguistic nuance but tended to *overassign* and sometimes misplace values in categorical boundaries.
- **ChatGPT** achieved **greater semantic precision**, introduced fallback groups like `Other` where appropriate, and provided high coverage with minimal overfitting.

Dataset	**Modello**	**Precision**	**Recall**	**F1**
groupby_001	GPT	0.86	0.86	0.86
groupby_001	Claude	1.00	1.00	1.00
groupby_002	GPT	1.00	1.00	1.00
groupby_002	Claude	0.81	0.81	0.81
groupby_003	GPT	0.71	0.71	0.71
groupby_003	Claude	0.62	0.62	0.62
groupby_004	GPT	1.00	1.00	1.00
groupby_004	Claude	0.86	0.86	0.86
groupby_005	GPT	1.00	1.00	1.00
groupby_005	Claude	0.95	0.95	0.95
groupby_006	GPT	1.00	1.00	1.00
groupby_006	Claude	0.83	0.83	0.83
groupby_007	GPT	0.81	0.81	0.81
groupby_007	Claude	0.76	0.76	0.76
groupby_008	GPT	0.78	0.78	0.78
groupby_008	Claude	0.91	0.91	0.91

Table 4: Precision, recall e F1 score per task, modello e dataset.

This confirms the viability of LLMs as tools for semantic data grouping in tabular contexts where SQL alone is insufficient.

3.3 UDF: NER

3.3.1 Methodology

Named Entity Recognition (NER) is a foundational task in Natural Language Processing (NLP) that involves identifying and classifying spans of text into predefined categories such as persons (PER), organizations (ORG), locations (LOC), dates, and more. In computer science, NER is widely studied because it enables the transformation of unstructured text into structured, machine-readable data, which is essential for tasks such as information extraction, question answering, semantic search, and knowledge graph construction.

The NER UDFs developed in this project were designed to extract such structured information from free-text fields that appear across multiple relational tables. These fields often contain semantically rich but structurally ambiguous content that standard SQL cannot interpret.

A diverse range of ten NER tasks was created, covering tables from student management systems, technical forums, event logs, and financial records. Each task involved:

- A real or realistic free-text field (e.g., `bio`, `description`, `AboutMe`, `notes`)
- A natural language query specifying one or more target entity types (e.g., PER, LOC, ORG)
- A manually annotated expected output in structured JSON format, categorized by entity type

The data used for NER examples included personal biographies, comments, event notes, technical documentation, and budget descriptions. This variety ensures coverage across multiple linguistic registers, narrative styles, and entity categories.

The prompts were designed to be domain-agnostic but structured, encouraging the LLM to return entity types grouped under appropriate labels. For instance, a bio field mentioning "Alice Rossi", "Italia", and "University of Cambridge" would be expected to return these under the PER, LOC, and ORG categories respectively.

3.3.2 Why NER is a Valuable UDF Use Case

NER offers several complementary benefits in a relational setting. First, its *cross-domain relevance* makes it universally applicable: nearly all textual attributes mention persons, organizations, or locations. Second, NER enables *schema enrichment*; extracted entities can populate auxiliary columns or materialised views, unlocking new joins and filters without altering the original schema. Third, because it relies on contextual understanding, NER goes *beyond pattern matching*, outperforming `LIKE` or regex approaches when facing ambiguous phrasing or abbreviations. It is also robust to *multilingual and mixed-format text*, an advantage in international databases. Finally, recognised entities become *composable query primitives*, driving follow-on analyses (e.g., grouping comments by company mentions) and completing the crucial transformation from unstructured to structured data that SQL engines can optimise.

3.3.3 Illustrative Examples

- **ner_udf_01** – *student_club.member.bio Query:* extract PER, LOC, and ORG entities from the biography of `member_id = 21`. *Expected output:* {PER: "Alice Rossi"; LOC: "Italy"; ORG: "University of Cambridge"}.

- **ner_udf_004** – *codebase_community.posts.Body Query:* list all programming languages mentioned in the body of `post Id = 128`. *Expected output:* [Python, Go, TypeScript].

Additional examples in the JSON file cover other entity classes—such as dates in `ner_udf_009` or mixed LOC/ORG extraction from financial notes in `ner_udf_007` demonstrating the breadth of NER applications across heterogeneous tables.

In database contexts where metadata is sparse and text fields are dense with meaning, NER is a foundational building block for unlocking advanced analytics via LLM integration.

3.4 UDF: Sentiment Analysis

3.4.1 Methodology

The sentiment analysis UDFs were designed to address scenarios where standard SQL cannot perform semantic analysis of textual content. The examples were carefully crafted to cover a range of sentiment analysis tasks, including:

- Basic sentiment classification (positive/negative/neutral)
- Domain-specific sentiment analysis (gaming app reviews)
- Combined sentiment analysis and summarization tasks

The dataset construction process involved:

1. Identifying real database schemas from the **BIRD Benchmark** that would contain sentiment-rich text fields (user profiles, product reviews, card flavor text)
2. Creating natural language questions that would require sentiment understanding
3. Manually annotating expected results through careful analysis of the text content
4. Documenting the SQL limitations that justify each UDF

Key considerations in example creation included:

- **Granularity**: Examples range from simple sentiment classification (Example 001) to complex multi-table operations (Example 010)
- **Domain Coverage**: Examples span multiple domains including app reviews, user profiles, and trading cards
- **Sentiment Spectrum**: Questions target different sentiment ranges (highly negative, neutral, highly positive)
- **Task Complexity**: Some examples combine sentiment analysis with other operations like filtering or summarization

The prompt engineering approach focused on creating a single, versatile template that could handle all sentiment analysis cases while ensuring structured output. Key features of this prompt design:

- **Domain Agnostic**: Single template handles all sentiment tasks across different domains
- **Structured Output**: Clear instructions enforce consistent response formats

- **Precision Focus**: Numeric sentiment scale (-1.0 to 1.0) for quantitative analysis
- **Minimalism**: Strict prohibition of extraneous information ensures clean parsing
- **Input Understanding**: Information on how to interpret the input in particoular the csv format for the tables

3.4.2 Results

The sentiment analysis examples were evaluated on two LLM architectures, with performance measured through both exact ID matching and BERTScore for summarization quality. Results are presented in the combined performance table below.

Table 5: Comparative Sentiment Analysis Performance

Example ID	**deepseek-r1-distill-llama-70b**			**llama3-70b-8192**		
	P	**R**	**F1**	**P**	**R**	**F1**
sentiment_analysis_001	0.000	0.000	0.000	0.000	0.000	0.000
sentiment_analysis_002	1.000	1.000	1.000	1.000	0.500	0.667
sentiment_analysis_003	0.333	1.000	0.500	0.667	1.000	0.800
sentiment_analysis_004	0.500	0.250	0.333	0.000	0.000	0.000
sentiment_analysis_005	0.600	1.000	0.750	0.625	0.833	0.714
sentiment_analysis_006	0.800	0.800	0.800	0.600	0.600	0.600
sentiment_analysis_007	0.750	0.600	0.667	0.143	0.200	0.167
sentiment_analysis_008	0.000	0.000	0.000	0.000	0.000	0.000
sentiment_analysis_classify_001	0.667	1.000	0.800	0.000	0.000	0.000
analyze_sentiment_summarize_001 (ID matching)						
	1.000	1.000	1.000	0.200	0.500	0.286
analyze_sentiment_summarize_001 (BERTScore)						
	1.000	1.000	1.000	0.200	0.500	0.286
Average (ID tasks)	0.565	0.665	0.585	0.323	0.363	0.323

Key findings from the dual-metric evaluation:

- **Model Comparison**: deepseek outperformed llama3 across all metrics (ID F1: 0.585 vs 0.323; BERTScore F1: 1.000 vs 0.286)
- **Example 10 Analysis**:
  - deepseek achieved perfect scores in both ID matching and summary quality (F1=1.000)
  - llama3 showed poor performance in both aspects (F1=0.286)
  - Strong correlation between ID retrieval accuracy and summary quality
- **Task Difficulty**:

  - Basic sentiment tasks (002-007) showed moderate performance
  - Complex tasks (009-010) revealed largest model capability gaps
  - Complete failures (001,008) suggest zero-shot limitations

- **Metric Correlation**:
  - BERTScore confirmed ID matching results for summarization
  - Discrepancy in Example 10 highlights importance of multi-metric evaluation

The results demonstrate that while the generic prompt structure worked adequately for simple sentiment tasks, model selection becomes crucial for more complex operations.

3.5 UDF: Summarization

3.5.1 Methodology

The user-defined function (UDF) named "summarization" is designed to condense descriptions, reviews, or general comments related to a given entity. The objective of this UDF is to generate concise summaries that provide the user with an overview of the key strengths and weaknesses of an entity without requiring them to read each individual comment.

To ensure scalability and domain-independence, a generic prompt was used in this case. The intention was to develop a solution that could be easily adapted across various domains, such as electronics products, accommodations, films, or even individual people. The summarization UDF collects all textual descriptions associated with a given entity and generates a coherent summary. This type of semantic processing goes beyond the capabilities of standard SQL, which cannot natively perform natural language understanding. To address this, a Large Language Model, *deepseek-r1-distill-llama-70b*, was employed via the GROQ platform. The full prompt configuration can be found in the accompanying file summarization_udf.py.
An essential component of the methodology was the construction of a synthetic dataset designed to span multiple domains and include varied description styles. This was done to test the generalizability and robustness of the summarization process. The dataset was initially generated using LLMs and then manually reviewed to ensure the accuracy and completeness of the synthetic content.
For each domain specific table (representing the descriptions of a single entity) a single summary was produced. The summarization task required the model to synthesize all available descriptions into one coherent, informative paragraph that accurately captured the overall sentiment and key points. A total of six examples were generated for this evaluation.
Once the UDF, prompt, and dataset were finalized, the inputs were passed to the *deepseek-r1-distill-llama-70b* model. The resulting summaries were collected and stored in the output file 'llmResponse.json'.

The next section presents the evaluation of the generated summaries using a custom assessment methodology, which is explained in detail under Results.

3.5.2 Results

The evaluation of the system was conducted in two complementary phases. The first phase involved the use of an automated semantic evaluation method based on a Language Model, while the second phase consisted of a manual human review to validate the coherence and relevance of the generated summaries.
For the automated evaluation, the BERTScore model was employed. BERTScore is a widely used metric for assessing the semantic similarity between two pieces of text. Unlike traditional n-gram-based approaches, BERTScore leverages contextual embeddings from pre-trained language models to determine how closely two texts align in meaning. It computes three key performance metrics: Precision, Recall, and F1-score, which collectively provide insight into the quality and semantic fidelity of the generated summaries compared to a reference.
This approach allows for a nuanced evaluation of how well the generated summaries capture the essential content of the source descriptions, even in cases where the exact wording differs. By focusing on semantic similarity rather than lexical overlap, BERTScore offers a more meaningful assessment for natural language generation tasks.
Following the automated evaluation, a manual review was conducted to further assess the coherence, informativeness, and factual accuracy of the generated summaries. This human-in-the-loop validation was essential for identifying potential shortcomings not captured by automated metrics.
The results of the evaluation are summarized in the table below.

	Precision	**Recall**	**F1-score**
Average Performance	0.91	0.9	0.89

Table 6: Summarization UDF average performance metrics (zero shot)

4. Materials

All code and documentation for reproducing these experiments are available at:

https://github.com/aledigirm3/ATCS-HW3

5. Conclusion

This work explores the integration of LLM-powered User Defined Functions into relational query pipelines, overcoming the semantic limitations of traditional SQL. By designing and evaluating UDFs across tasks such as classification, summarization, sentiment analysis, and semantic grouping, we demonstrated that modern language models can significantly enhance query expressiveness. While models like Claude 4 Sonnet excel in precision and linguistic nuance, others like GPT-4o offer broader recall capabilities. Our experiments reveal that LLMs can effectively support complex analytical tasks in databases, though challenges in consistency, cost, and interpretability remain. Future directions include prompt refinement, hybrid execution frameworks, and cost-aware query optimization.