open proceedings

LogicLM: Robust Application of Large Language Models with Logic Programming for Data Analytics

Evgeny Skvortsov
Google LLC
Kirkland, WA, USA
evgenys@google.com

Shayan Mirjafari
Google LLC
Kirkland, WA, USA
shayanmj@google.com

Ojaswa Garg
Google LLC
Kirkland, WA, USA
ojaswagarg@google.com

Yilin Xia
University of Illinois
Urbana-Champaign, IL, USA
yilinx2@illinois.edu

Shawn Bowers
Gonzaga University
Spokane, WA, USA
bowers@gonzaga.edu

Bertram Ludäscher
University of Illinois
Urbana-Champaign, IL, USA
ludaesch@illinois.edu

ABSTRACT

We present LogicLM, an OLAP-style interactive data analysis system that leverages large language models (LLMs) and is configured using Logica, an enhanced logic programming language with aggregation support that compiles to SQL. LogicLM uses an LLM to translate natural language queries by end users into executable code for automatically generating data visualizations. For each natural-language query, LogicLM provides a verifiable OLAP-based configuration that users can view and modify to help ensure results are reliable and accurate. This configuration, with measures, dimensions, and filters defined as logical predicates, offers a unified and user-friendly approach to natural-language data exploration, while keeping end users in control of the analysis process.

1 INTRODUCTION

Recent advances in Large Language Models (LLMs) are opening new avenues for solving software engineering and data analysis problems [3, 7, 8, 15]. One natural application of LLMs for data analysis is to automatically generate SQL from natural-language prompts (e.g., see [2, 4, 13]). Using LLMs in this way enables analysts and data scientists to explore datasets more efficiently by reducing manual coding effort (via low-code or no-code solutions), thereby enabling faster iteration to data insights. However, without transparency into the use of the underlying LLM and its output, these opportunities come with the potential risk of erroneous results (or worse, poor decisions) due to mistakes made by the LLM.

In this paper, we introduce LogicLM: a lightweight, open source, and freely available natural-language data analytics platform [10]. LogicLM is designed to address the challenges of potential inaccuracies in LLM-generated queries and the need for user control and transparency in the query generation process. It employs predicate calculus as an intermediate representation between natural language and data retrieval, leveraging the expressive power of logic programming to enhance query reliability and efficiency. By utilizing Logica [11, 12], a free and open-source logic programming language, LogicLM translates natural-language user requests into structured query configurations that are both human-readable and machine-interpretable. This dual representation ensures that LLMs interpret requests with high accuracy and that users remain in control of the query generation process, with the ability to view and edit the generated query configurations directly.

Figure 1, for example, shows two different LogicLM user sessions. Users enter a natural language description of the information they would like to visualize, as well as the type of visualization to use, in the upper left-hand pane of the LogicLM interface. After clicking the *Run Analysis* button, a simplified form based on the underlying structured query configuration generated by LogicLM is shown in the lower-left pane. Clicking *Run Analysis* additionally generates the corresponding visualization from the underlying data, as shown in the pane on the right of the UI. Users can also interact with and modify the query configuration in the bottom-left pane to generate a revised visualization.

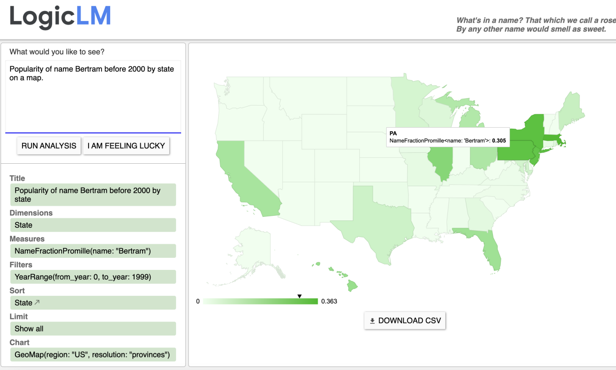
In LogicLM, measures, dimensions, and filters are defined as predicates, making the initial data setup and configuration process simple and enabling robust queries for performing Online Analytical Processing (OLAP) [6]. This predicate-based approach streamlines the definition of analytical concepts and leads to reusable and clear query construction. LogicLM also supports multiple database back-ends through Logica, including SQLite, DuckDB, and BigQuery. To interpret and translate natural language queries, LogicLM integrates with LLMs, including Google Gemini [14], OpenAI [1], and MistralAI [5].

By combining the principles of predicate calculus, logic programming, and natural language processing, LogicLM offers a novel approach to data analytics in natural language. It empowers users with varying technical expertise to perform sophisticated data analyses efficiently. LogicLM's open-source nature and support for multiple back-ends make it a flexible and accessible tool for addressing contemporary data analysis challenges. The rest of this paper provides an overview of the LogicLM architecture, a more detailed example of the use of Logica for modeling measures, dimensions, and filters, and an overview of the LogicLM demonstration.

2 SYSTEM OVERVIEW

LogicLM leverages the power of LLMs to interpret and translate natural-language requests into structured queries. However, recognizing the potential for errors in direct LLM-generated queries, LogicLM introduces an intermediary step: it represents the user's request as an OLAP-style query configuration. This query configuration includes the core components of the analysis, the measures to be calculated, the dimensions along which to analyze the data, and the filters to apply for specific subsets. These components are defined in the logic programming language Logica.

© 2025 Copyright held by the owner/author(s). Published in Proceedings of the 28th International Conference on Extending Database Technology (EDBT), 25th March-28th March, 2025, ISBN 978-3-89318-099-8 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.



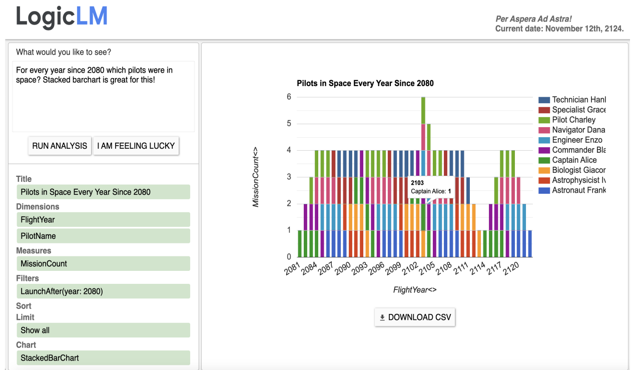


Figure 1: The LogicLM user interface for two different (synthetic) dataset examples: on the left, a request for a map showing the popularity of a specific baby name prior to 2000; and on the right a request for a stacked barchart showing the pilots in space starting after a certain year.

Figure 2 shows the overall workflow and basic components of LogicLM. To add a new dataset, a data engineer (Step 1 in the figure) creates a Logica *Cube Configuration* script, which includes the rules for defining OLAP-style dimension, measures, and filters (cf. Section 3). Within the LogicLM interface (as shown in Figure 1), an end user can select a configured dataset and (Step 2) specify a natural-language query via the *Query Input Panel* (the top left of the LogicLM UI). The Query Input Panel (Step 3) combines the user's natural language query and the Cube Configuration script to generate a prompt that is given to the LLM. The LLM then produces a *Query Request* (as a JSON file) that is displayed in the *Query Config Panel* (the bottom left of the LogicLM UI). Note that a visualization is also generated (the right panel of the LogicLM UI). The end user (Steps 5 and 6) can optionally modify the Query Request via the Query Config Panel. The Query Request is then parsed using the LogicLM *Query Request Parser*, which generates a Logica program for (Step 7) querying the underlying SQL Engine and (Step 8) generating the corresponding visualization. The result is then displayed in the *Visualization* panel of the LogicLM UI. Users can iteratively modify their natural language prompt and the resulting Query Configuration.

A key strength of LogicLM lies in its transparency and user control. Because the generated OLAP-style query configuration (which acts as a blueprint for the underlying data query) is presented to the user for review and modification, regardless of a user's expertise in programming, they can easily verify that the system has correctly understood their intent and can fine-tune the analysis as needed.

3 LOGICAL OLAP

LogicLM's key feature is its ability to dynamically define OLAP structures using Logica scripts. With Logica, measures, dimensions, and filters can all be treated as logic-based predicates. Filters are represented using standard predicates, whereas dimensions and measures are expressed as special *functional* predicates supported within Logica. This section further explains how these predicates are constructed using an example synthetic data source whose schema is shown in Figure 3.

Logica extends Datalog syntax to make logic programming applicable to data analysis and its key features help with making LogicLM configuration efficient and intuitive:

- **Functional notation**: Functions in Logica are simply predicates with a column dedicated as the (returned) value. We use functional predicates to define dimensions in LogicLM.
- **Aggregation**: Logica extends Datalog with aggregation and allows users to define custom aggregation operations. These features are used in LogicLM to define measures.
- **Named and positional arguments**: Arguments of Logica predicates can be used positionally or via a name. LogicLM adopts the convention that measures, dimensions, and filters use the first positional argument as the treated fact, while named arguments represent parameters of the query.

In LogicLM, **Fact Tables** serve as primary data sources[1] that are used as the foundation for defining dimensions, measures, and filters. Leveraging Logica's support for composite data types, fact tables in LogicLM are typically stored as single-column tables, where each fact represents a record, with the individual fields corresponding to the columns of the record.

While LogicLM supports multiple fact tables, here we assume that only one fact table is used, namely, the `FoodOrder` table shown in Figure 3. Depending on the chosen database engine in Logica, data engineers can configure a `FoodOrderDataSource(fact)` predicate either using a local file address or by connecting to an existing cloud database, such as Google BigQuery.

```
1 FoodOrderDataSource(fact) :- `path_to_datasource`(..fact);
```

Dimensions represent various perspectives or attributes by which data can be analyzed. They are expressed as functional predicates in LogicLM. For example, `CustomerDistrictName` is a dimension that associates an order to the name of the district where the customer is located. Data engineers can define the functional predicate `CustomerDistrictName(fact)` in Logica, where the value `district_name` is derived from the `District` table. In this case, the `district_id:` should match `fact.customer_district`, which refers to the *customer_district* from the table `FoodOrder`.

```
1 CustomerDistrictName(fact) = district_name :-
2   District(district_id: fact.customer_district,
3            name:district_name);
```

[1]For details of how data sources are defined in relation to the underlying database backend, refer to the LogicLM open-source example code [10]

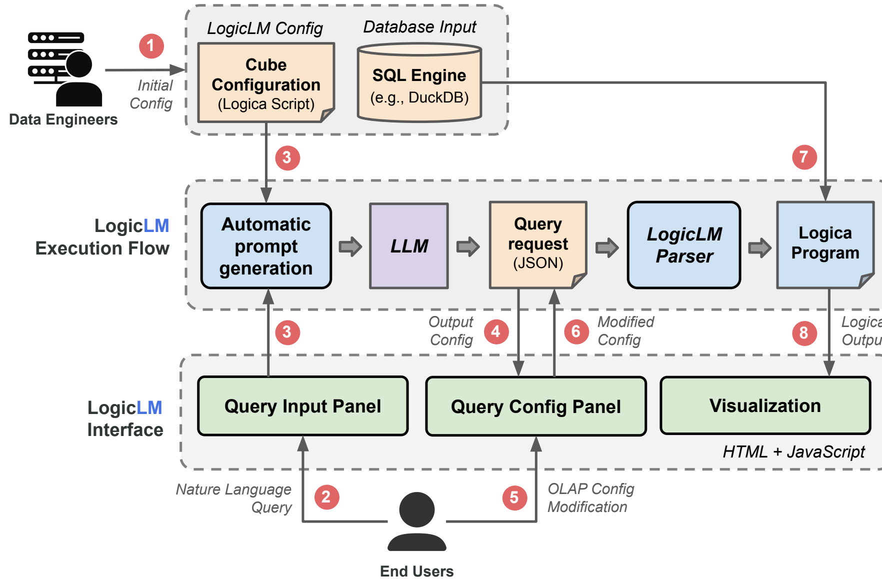


Figure 2: LogicLM Architecture and Overview. To configure the system, a data engineer uses Logica rules (1) to specify an OLAP cube including *dimensions*, *measures*, and possible *filters*. The user enters a natural language query (2) in the browser-based UI which is appended to the cube information (3) to generate an integrated prompt. From this, the given LLM generates a query configuration (4) that the user can modify *if needed* (5, 6). The query configuration is translated into a Logica program that is executed via compilation to SQL queries running on the underlying database system (7). The results are visualized in the UI (8).

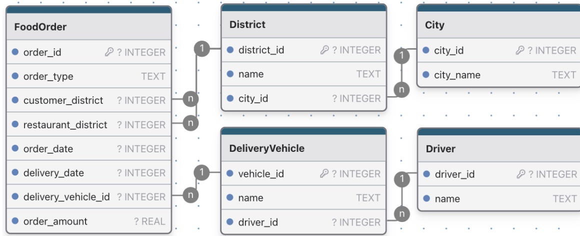


Figure 3: Schema of the synthetic dataset used in the demo of the system. FoodOrder is a fact table and the rest are dimension tables.

Using this functional predicate, we can leverage the `District` dimension table to map food order facts, which lack the district name, to the customer's city. To create the `CustomerCityName` dimension and associate an order with a city, we need to perform a join. In Logica, this can be accomplished by simply combining two predicates using a conjunction.

```
CustomerCityName(fact) = city_name :-
  District(district_id: fact.customer_district,
           city_id:city_id),
  City(city_id:, city_name:);
```

Measures represent quantitative data that can be aggregated for analysis. In LogicLM, they are represented as aggregating operators, which are functional predicates that utilize built-in aggregation functions. For instance, the measure `OrderCount`, which calculates the total number of orders, can be defined in Logica as below.

```
OrderCount(fact) = Count(fact.order_id);
```

Additionally, aggregations can be applied to expressions. For example, `DeliveryDuration` is a measure that calculates the sum of the differences between delivery and order dates:

```
DeliveryDuration(fact) =
  Sum(fact.delivery_date - fact.order_date);
```

Filters in LogicLM are standard predicates for defining specific conditions. For instance, the filter `OrdersToCities` selects orders from a set of cities in the `city_names` parameter. Because dimension `CustomerCityName(fact)` is defined as a functional predicate, it can be directly reused as in the following example.

```
OrdersToCities(fact, city_names:):-
  Constraint(CustomerCityName(fact) in city_names);
```

Metadata in LogicLM is defined using a dedicated predicate that specifies the fact table, measures, dimensions, and filters, and can configure UI elements like the header tagline and server port. It also includes a `suffix_lines:` field, which allows custom text to be appended to the LLM prompt. This enables users to refine the LLM's understanding by providing additional context or instructions. The final prompt, formed by combining the metadata and user request, guides the LLM in constructing the query. The metadata context significantly reduces the search space the LLM must consider when identifying the appropriate measures, dimensions, and filters. As a result, the LLM's performance is greatly enhanced, minimizing misinterpretations. For the full definition of metadata, refer to the `.l` file in the repository [10].

```
LogicLM(title: "Delivery Statistics",
        fact_tables:["FoodOrderDataSource"],
        dimensions:["CustomerDistrictName","CustomerCityName"],
        measures:["OrderCount", "DeliveryDuration"],
        filters:["OrdersToCities"],
        suffix_lines:[
```

Figure 4: The LogicLM UI displays: (a) a Query Input Panel for user requests; (b) an OLAP Config Panel showing the LLM-generated OLAP configurations, with options for interactive adjustments; and (c) a visualization of the query results.

```
7          "Try using linechart or barchart .",
8          "If unsure use table chart .",
9          "Good luck !"]);
```

4 DEMONSTRATION PLAN

The demonstration showcases LogicLM running with the configuration detailed in Section 3. Attendees can interact with the system by posing questions and observing the results. We highlight the system's feedback loop, emphasizing how users can verify the LLM's understanding through the OLAP Config Panel. Figure 4 provides one of the examples used in the demonstration showing a natural language question posed in LogicLM, its corresponding OLAP Config Panel, and its resulting chart.

In the example, the user asks *"How many deliveries were done to districts of Springfield."* The LLM, based on the LogicLM meta-configuration, interprets the natural-language query into: `OrderCount` as the measure, `CustomerDistrictName` as the dimension, and `OrderToCities` with the argument of `city_names:["Springfield"]` as the filter. This interpretation, formatted as a JSON object, is parsed and results in the following Logica program, which is then compiled to SQL[2]:

```
1 Report(
2   OrderCount? Aggr= OrderCount(fact),
3   CustomerDistrictName:CustomerDistrictName(fact)) distinct:-
4     FoodOrderDataSource(fact),
5     OrdersToCities(fact, city_names:["Springfield"]);
```

CONCLUSION

LogicLM provides a transparent approach to data analysis, converting natural language requests into readable Logica scripts representing OLAP queries. LogicLM allows users to verify the system's interpretation and enables engineers to extend the configuration, reusing its definitions for custom analyses. LogicLM supports diverse database backends such as SQLite and BigQuery, making it adaptable to various data sizes and analytical scenarios, from ad-hoc exploration to large-scale warehousing. It combines LLMs for natural language processing with logic programming for accuracy and accessibility in interactive data analysis.

[2]The SQL compilation is done via conventional programming. For more information about the compilation process, see the Logica open-source code [9, 12]

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] CopilotPowerBI. Overview of Copilot for Power BI. https://learn.microsoft.com/en-us/power-bi/create-reports/copilot-introduction. Accessed: 12/10/2024.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv, 2018. URL https://arxiv.org/abs/1810.04805.

[4] GeminiLooker. Introducing Gemini in Looker to bring intelligent AI-powered BI to everyone. https://cloud.google.com/blog/products/data-analytics/introducing-gemini-in-looker-at-next24. Accessed: 12/10/2024.

[5] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[6] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 3rd Edition.* J. Wiley & Sons, 2013.

[7] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep Contextualized Word Representations. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 2227–2237, June 2018.

[8] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. Querying large language models with SQL. In *International Conference on Extending Database Technology, EDBT*, pages 365–372, 2024. URL https://doi.org/10.48786/edbt.2024.32.

[9] Evgeny Skvortsov. Logica Source Repository, December 2023. URL https://github.com/EvgSkv/logica.

[10] Evgeny Skvortsov, Shayan Mirjafari, Kevin Prewitt, and Ojaswa Garg. LogicLM Source Repository, December 2024. URL https://github.com/google/LogicLM.

[11] Evgeny S. Skvortsov, Yilin Xia, Shawn Bowers, and Bertram Ludäscher. The Logica System: Elevating SQL Databases to Declarative Data Science Engines. In *International Workshop on the Resurgence of Datalog in Academia and Industry (Datalog-2.0)*, volume 3801 of *CEUR Workshop Proceedings*, pages 69–73, 2024. URL https://ceur-ws.org/Vol-3801/short5.pdf.

[12] Evgeny S. Skvortsov, Yilin Xia, and Bertram Ludäscher. Logica: Declarative Data Science for Mere Mortals. In *International Conference on Extending Database Technology (EDBT)*, pages 842–845, 2024.

[13] TableauAI. AI in Tableau Accelerate your data culture with AI-powered insights. https://www.tableau.com/products/artificial-intelligence. Accessed: 12/10/2024.

[14] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.