A Preliminary Evaluation of Open-Source LLMs for Datalog-Based Semantic Parsing in the ASVIN Project

Mario Alviano, Matteo Capalbo, Georg Gottlob, Irfan Kareem, Fabrizio Lo Scudo and Sebastiano Piccolo

Department of Mathematics and Computer Science, University of Calabria, Rende (CS), Italy

Abstract

This paper presents a preliminary evaluation of open-source Large Language Models (LLMs) for semantic parsing in the context of the ASVIN (Assistente Virtuale di Negozio) project, which aims to develop a regulatory-compliant virtual assistant for the fashion retail sector. The core task involves translating natural language user queries into structured semantic representations using Datalog (or Answer Set Programming), enabling logical reasoning and personalized interaction in e-commerce applications. We construct a pilot dataset of fashion-domain queries annotated with Datalog ground truths and evaluate multiple LLMs using zero-shot and one-shot prompting strategies. The evaluation focuses on syntactic and semantic accuracy, using F1-score as the primary metric, and compares performance across a range of open models including Mistral, Gemma, Qwen, and DeepSeek. Our results show that smaller models such as Mistral-small3.1 can outperform larger counterparts when guided by well-structured prompts, highlighting the importance of prompt design and task framing. This study lays the groundwork for a full-stack integration of LLM-based reasoning in virtual retail assistants.

Keywords

Large Language Models (LLMs), Semantic Parsing, Datalog, Answer Set Programming (ASP), Virtual Assistants

1. Introduction

Recent advancements in Large Language Models (LLMs) have significantly improved the capabilities of conversational agents, enabling fluid and deep interactions across a wide range of domains. These systems have shown remarkable success in understanding and generating natural language, making them appealing candidates for applications in virtual assistance, including e-commerce and customer support scenarios [1, 2, 3, 4]. However, while LLMs excel at producing linguistically coherent outputs, they exhibit notable limitations in tasks requiring formal logical reasoning, consistency, or verifiability [5, 6, 7]. This shortcoming becomes especially evident in domains such as fashion retail, where users often seek actionable, context-aware, and personalized recommendations based on incomplete or underspecified input.

In this paper, we present the first systematic effort to leverage open-source LLMs for semantic parsing into Datalog within the ASVIN project (Assistente Virtuale di Negozio), a project about a regulatory-compliant virtual shopping assistant designed for the fashion retail sector. Our central objective is to bridge the gap between natural language understanding and symbolic reasoning, by transforming user queries into structured logical representations suitable for downstream reasoning tools such as Datalog or Answer Set Programming (ASP) [8, 9]. This step is critical for grounding the responses generated by the assistant in formal semantics, enabling explainable and verifiable behaviors in product recommendation, item compatibility checking, and style advice delivery.

Current chatbot systems built on LLMs often rely on retrieval augmented generation (RAG) or fine tuned models to provide answers based on existing product catalogs or textual descriptions [10, 11]. However, these systems struggle when user queries are vague, incomplete, or implicit as it often happens with natural language queries. For example, a user might ask, "What can I wear with a floral skirt for a wedding?" or "I need something trendy but comfortable for a summer party." These inputs lack explicit references to catalog items or well-formed constraints. A traditional LLM might respond with

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2025-12-03

generic style suggestions, missing the opportunity to reason over item compatibility, availability, or the preferences previously expressed by the user.

Example 1. *Consider the user query: "What can I wear with a floral skirt for a wedding?" When prompted with this input, a general-purpose LLM such as ChatGPT may produce a fluent but vague recommendation, for example:*

```
Classic & Elegant Look
Top: Silk or satin camisole or blouse in a neutral or complementary
     color (e.g., ivory, blush, soft grey).
Shoes: Nude or metallic (gold/silver) heels or strappy sandals.
Accessories: Dainty gold or pearl jewelry, clutch in a neutral or
     metallic tone.
```

While the language is stylistically appropriate, the suggestions are not grounded in the actual wardrobe of the user, catalog availability, or any regulatory or stylistic policies that might constrain recommendations. Moreover, they are not verifiable: the system cannot explain why a satin camisole is compatible with a floral skirt, or whether that combination aligns with brand-specific guidelines for wedding attire.

Reinforcing the prompt with additional context might improve the output, but even with additional prompt context, the assistant lacks the structured semantic representation needed to reason over inventory, compatibility, or preference constraints in a principled way.

Reinforcing the prompt with additional context might improve the output, but even with such enhancements, the assistant lacks a structured, verifiable semantic representation of the query. This limits its ability to reason over inventory, enforce stylistic constraints, or provide explainable recommendations/requirements that are essential in the ASVIN setting. ■

In ASVIN, we aim to go further; by extracting a structured semantic representation of the user's intent, we aim at performing formal reasoning over product attributes, compatibility rules, and catalog constraints. The core idea is to parse natural language inputs into Datalog predicates, which formalize user goals, constraints, and preferences. These predicates can then be used as input to a logic engine (e.g., Datalog or ASP solver), whose output can guide product recommendation or inform further dialogue. This approach allows us to benefit from both sides: the expressive power of LLMs and the precision of logical reasoning frameworks.

Example 2. *From the following user question "What can I wear with a floral skirt for a wedding?" we want ASVIN to extract the following Datalog representation:*

```
request_type(fashion_advice).           companion_item(skirt).
style_of_companion_item(skirt, floral). event(wedding).
```

Such a representation can inform and guide the answers of the virtual assistant. If more information is needed, ASVIN can request it; otherwise, ASVIN can suggest items in the catalog that are appropriate for the specified style and occasion. Even if reasoning is not yet implemented in the current prototype, this Datalog representation is designed to serve as input to a logic-based module. Given catalog facts (e.g., available items and their attributes) and style compatibility rules encoded in ASP, the system could infer suitable combinations (for example, recommending pastel tops or neutral-toned shoes that match the floral pattern and formality of a wedding). ■

Mapping natural language to formal logic is challenging due to its ambiguity and context dependence, which traditional parsers cannot handle well. To overcome this, we use LLMs with in-context learning to extract Datalog facts, guiding them through carefully designed domain-specific predicates and task-specific prompts. Additionally, to address the lack of suitable datasets, we introduce a novel benchmark consisting of 50 hand-curated fashion-related user queries, each annotated with a corresponding set of ground truth Datalog predicates. The dataset spans a variety of scenarios, from stylistic queries to occasion-specific recommendations and compatibility checks.

Our evaluation focuses on a range of competitive open-source LLMs, including Mistral (3.1-small), Qwen, Gemma, and DeepSeek, assessed for their ability to extract semantically accurate predicates. In the zero-shot setting, where models are only provided with a list of valid predicates and a task description, performance is generally poor, with limited consistency and low F1-scores. In contrast, in the one-shot setting, where a single example is added to the prompt, performance are significantly improved. The best results are achieved by Mistral 3.1-small, with F1-score reaching 0.82, illustrating that even relatively small models can perform well when guided by carefully constructed prompts.

2. Background

We refer to the ASP-Core-2 format [12] for common constructs of Answer Set Programming (ASP). Here we consider constants being integers or strings starting by lowercase. A *fact* has the form $p(\overline{c})$, where p is a predicate and $\overline{c}$ is a possibly empty sequence of constants. A *database* is a possibly empty set of facts. A *program* (or *ASP Knowledge Base*) is a set of rules defining conditions to derive new facts from an input database, or to eliminate undesirable solutions. For the purposes of this paper, it is sufficient to see a program as a black-box associating one input database to zero or more output databases (according to the stable model semantics [13]).

Example 3. *Let us consider the following database:*

```
is_item(1, trousers).
is_item(2, shirt).
is_item(3, trousers).
is_appropriate(1, wedding).
is_appropriate(2, wedding).
```

Let us consider the following ASP rule which defines if a product is suitable for a certain type of event:

```
suitable(ID, Item, Event) :- asks(Item), is_item(ID, Item),
event_type(Event), is_appropriate(ID, Event).
```

Finally, let us consider the user query: "Show me the trousers that are appropriate for a wedding", from which ASVIN extracts the following facts:

```
asks(trousers).
event_type(wedding).
```

Giving in input the full program into an ASP solver, we get the following answer set $\{\text{suitable}(1, \text{trousers}, \text{wedding})\}$*. In fact, from the database we know that the product with id=2 is not a trousers, although it is appropriate for a wedding. The product with id=3, instead, is a trousers but it is not appropriate for a wedding.* ■

3. Problem Statement

The problem addressed in this work is the semantic parsing of natural language user queries into structured Datalog facts, with the goal of enabling formal reasoning in the context of a virtual assistant for fashion retail. Specifically, given a user request, we want to obtain a structured representation of it using a set of predefined domain-specific Datalog predicates. These predicates are designed to capture semantically relevant aspects of the query, such as the intent of the user (e.g., request for a recommendation), contextual constraints (e.g., event type, weather, color preferences), or compatibility conditions (e.g., which items should go together).

Given a query q in natural language and a dictionary of domain-specific predicates $\mathcal{P}$, we want to extract a set of Datalog facts $F = \{p_1(\overline{c_1}), p_2(\overline{c_2}), \dots\}$ where $p_i \in \mathcal{P}$ and each $\overline{c_j}$ is a possibly empty sequence of constants drawn from relevant domains (e.g., clothing items, occasions, styles). The resulting set of facts F expresses the semantic content of q in a form suitable for downstream reasoning using Datalog or ASP.

This paper reports on the initial attempt, in the context of the ASVIN project, to enable the translation $\mathcal{P}, q \mapsto F$. Such a translation is part of a broader pipeline, where the set F of facts is subsequently processed by an ASP solver to address formal reasoning, and enrich user queries expressed in natural language. Indeed, once a query is parsed into a set of facts, it can be combined with a domain-specific rule base to derive logical consequences that inform recommendation and retrieval tasks.

4. Representing User Requests via Domain-specific Predicates

In order to support formal reasoning over user requests in the ASVIN project, we developed a tailored set of domain-specific predicates designed to capture the essential semantic elements of fashion-related queries. These predicates serve as an intermediate representation between natural language input and logical inference, enabling the system to produce grounded, explainable, and context-aware recommendations.

ASVIN focuses primarily on enhancing the in-store shopping experience through a virtual assistant capable of interpreting vague or under-specified user queries. We identified two primary classes of user intent: item recommendations and style advice. In an item recommendation, the user explicitly requests a type of item; for example, "What shoes can I wear under blue navy trousers?", where the goal is to suggest suitable shoes. In contrast, style advice occurs when the user seeks guidance without explicitly specifying what they want; for example, "What can I wear with a blue shirt?", where the mentioned item is already possessed by the user, and the assistant must infer compatible additions.

To formally capture these variations, we designed a set of predicates that captures the type of request as well as contextual attributes that influence recommendation quality: item properties (such as color, material, and style), situational parameters (event type, formality level), and temporal context (time of day, season, or part of the year).

This structured representation serves two key purposes: *(i)* it enables symbolic reasoning via ASP to filter and generate suitable item combinations from the product catalog, and *(ii)* it provides a stable target for the semantic parsing task performed by the LLM. Rather than attempting to generate logic rules or full programs from scratch, we focus on extracting a constrained set of well-defined facts, which can be reliably interpreted and composed with existing background knowledge.

Table 1
Domain-specific predicates used to semantically structure user queries.

Predicate	Description
`type_of_request(request_type)`	Request type: recommendation or style advice.
`request_item(item_name)`	An item explicitly requested by the user.
`material_of_request_item(item, material)`	Material of the requested item.
`color_of_request_item(item, color)`	Color of the requested item.
`style_of_request_item(item, style)`	Style of the requested item.
`advice_item(item_name)`	The item for which style advice is being requested.
`material_of_advice_item(item, material)`	Material of the item in the advice request.
`color_of_advice_item(item, color)`	Color of the item in the advice request.
`style_of_advice_item(item, style)`	Style of the item in the advice request.
`companion_item(item_name)`	An item to be matched by the recommended one.
`material_of_companion_item(item, material)`	Material of the companion item.
`color_of_companion_item(item, color)`	Color of the companion item.
`style_of_companion_item(item, style)`	Style of the companion item.
`season(season_name)`	Season or part of the year relevant to the request.
`event(event_name)`	Type of event (e.g., wedding, business meeting).
`location_event(location_name)`	Location where the event will take place.
`formality(formality_level)`	Formality level of the occasion (e.g., casual, formal).
`time_of_event(time)`	Specific time information for the event.
`weather(weather_condition)`	Weather condition relevant to the request.

We report the set of predicates in Table 1. Besides the predicate `type_of_request/1`, the other predicates fall in one of the following groups: *1)*: predicates that describe the requested item, *2)* predicates that describe the companion item, *3)*: predicates that describe the item for which the user is requesting a style advice, and *4)* situational predicates that describe the event for which the user is asking the query.

To facilitate accurate semantic parsing by LLMs, we constrained all domain-specific predicates to have at most two arguments. This design choice reduces syntactic complexity, aligns well with common natural language structures, and lowers the likelihood of generation errors—particularly in zero- or few-shot prompting settings. While this results in a larger set of predicates, the benefits in terms of model reliability and ease of logical integration outweigh the cost in verbosity.

The set of predicates, although minimal, can capture even complex queries such as queries in which the user asks for more the one recommendation, or having more than one companion item, or even queries where the user combines both a recommendation and a style advice.

Example 4. *Let us consider the following user query:*

> *"I am looking for a jacket for a winter business trip. I usually wear gray wool trousers. What would go well with that?"*

This query contains both a recommendation request and a style advice. It can be structured as follows:

```
type_of_request(request).
request_item(jacket).
season(winter).
event(business_trip).

type_of_request(advice).
advice_item(trousers).
companion_item(trousers).
color_of_companion_item(trousers, gray).
material_of_companion_item(trousers, wool).
```

This example showcases the flexibility of the predicates we designed. Here, the trousers are both a companion item for the recommendation of the jacket, and an item to take into account for a style advice. ■

5. A Synthetic Dataset of Fashion-related Queries

In the fashion domain, user queries are typically expressed in natural language. To address our problem, there is a need for a dataset that captures these queries and translates them into structured representations, such as those based on Datalog facts.

To create the dataset, we manually generated 10 meta queries, each comprising *query* and *ground_truth* in JSON file format. These meta queries are divided into two types of requests: (i) item recommendation and (ii) style advice. From these meta-queries, we expanded the dataset to 50 examples by generating 5 examples from each of the 10 meta-queries. Each example comprises a *query*, a user request in natural language simulating a possible request in the fashion domain, and a corresponding *ground truth*, which represents the semantic meaning of the query as a set of predicates.

Example 5. *Below is an example entry in the produced dataset:*

```
{ "query": "Recommend for gold sequin skirt a strappy high heels,
             a satin clutch, and a cashmere wrap for a dinner.",
    "ground_truth": [
      "type_of_request(item_recommendation)",
      "request_item(heels)",
      "style_of_request_item(heels, strappy_high)",
      "request_item(clutch)",
      "material_of_request_item(clutch, satin)",
      "request_item(wrap)",
      "material_of_request_item(wrap, cashmere)",
      "companion_item(skirt)",
      "color_of_companion_item(skirt, gold_sequin)",
      "event(dinner)" ]}
```

The above entry illustrates an item recommendation query along with its ground truth, where the attributes are represented as structured facts. ■

6. Experiment

In this section, we present a preliminary evaluation of open-source LLMs for the task of translating natural language fashion-related queries into structured Datalog predicates. The goal of this experiment is to assess the ability of different models to perform accurate semantic parsing, providing the formal representations necessary for downstream logical reasoning in the ASVIN system. We conduct this evaluation using the benchmark dataset of fashion-related queries introduced in Section 5. The selected models are: Mistral-small 3.1 24B, Gemma 3 27B, Gemma 3 12B, Mistral 7B, Mixtral, Llama 3 70B, Llama 3 8B, Deepseek R1 70B, Qwen 3 32B, and Phi4 14B. We selected these models to cover a representative sample of contemporary open-source LLMs with competitive performance across general NLP benchmarks, despite none being specifically trained for semantic parsing into formal languages.

We evaluate the LLMs under two prompting strategies: *zero-shot* and *one-shot*. In the zero-shot setting, the model is given only the task description and the input query. In the one-shot setting, a single example of a successfully parsed query is appended to the prompt, providing a template for the expected output structure. The full prompt used in both settings is illustrated in Figure 1. The one-shot variant includes the example at the end, whereas the zero-shot version omits it.

The prompt frames the model as a specialized AI assistant tasked with translating natural language requests into structured Datalog facts. It imposes strict output requirements: no explanations or extra text, only the Datalog program. The core rules instruct the model to (i) identify the request type, (ii) annotate items and attributes conditionally based on whether the query is a recommendation or a style advice, and (iii) ensure semantic and syntactic correctness in predicate use. A predefined vocabulary of allowable predicates is provided, encompassing item attributes (e.g., material, color, style), contextual cues (e.g., event, season, location), and structural roles (e.g., request or advice items, companion items). This controlled prompt design ensures consistency in model outputs and facilitates precise evaluation of the ability of the model to semantically interpret domain-specific queries.

All experiments were executed on a high-performance server equipped with three AMD MI210 GPUs (64 GB each), running the models in parallel. To evaluate performance, we use the F1-score; which is defined as

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{and} \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Here, *TP* denotes the number of true positives, *FP* denotes the number of false positives, and *FN* denotes the number of false negatives. For each query in the test set, we compute precision, recall, and F1-score by comparing predicted facts by the model to the annotated ground truth. The overall F1-score for each LLM is then obtained as the macro-average across all 50 test queries. That is, we calculate the F1-score independently for each example and then take the unweighted mean. This approach ensures that each test case contributes equally, regardless of the number of predicted facts.

One-shot Prompt (Zero-shot: lines 1-33)

```
1  You are an AI assistant specialized in translating natural language queries into Datalog programs.
2    Your task is to convert the user's query into a series of Datalog facts.
3    You must adhere to the following rules and use only the predicates from the predefined list.
4  Output Requirement:
5    You must return only the Datalog program, with no explanations, no comments, and no additional text.
6  Core Rules:
7    1. Identify the main purpose of the query and use `type_of_request(request_type)`. The `request_type` can be
           'item_recommendation' or 'style_advice'.
8    2. Conditional Item Tagging:
9         – If `type_of_request` is `item_recommendation`, all items being recommended must be tagged using
             `request_item(item_name)`. Related attributes like material, style, or color for these items should use predicates
             like `material_of_request_item(item_name, material)`, `style_of_request_item(item_name, style)`, etc.
10        – If `type_of_request` is `style_advice`, all items for which advice is being given or combined must be tagged using
             `advice_item(item_name)`. Related attributes for these items should use predicates like
             `material_of_advice_item(item_name, material)`, `color_of_advice_item(item_name, color)`, etc.
11   3. Ensure that the arguments for each predicate are extracted correctly from the query.
12 Allowed Predicates:
13   * type_of_request(request_type)
14   * request_item(item_name) // Use only if type_of_request is 'item_recommendation'
15   * material_of_request_item(item_name, material)
16   * style_of_request_item(item_name, style)
17   * color_of_request_item(item_name, color)
18   * advice_item(item_name) // Use only if type_of_request is 'style_advice'
19   * material_of_advice_item(item_name, material)
20   * style_of_advice_item(item_name, style)
21   * color_of_advice_item(item_name, color)
22   * companion_item(item_name)
23   * material_of_companion_item(item_name, material)
24   * style_of_companion_item(item_name, style)
25   * color_of_companion_item(item_name, color)
26   * season(season_name)
27   * event(event_name)
28   * location_event(location_name)
29   * formality(formality_level)
30   * time_of_event(time)
31   * weather(weather_condition)
32 Translate the following natural language query into a Datalog program.
33 Only output the program, and nothing else.
34 Example 1 (Item Recommendation):
35   Query: "Recommend beige suede heels to go with a burgundy dress for an autumn cocktail party."
36   Datalog Output:
37   type_of_request(item_recommendation).
38   request_item(heels).
39   material_of_request_item(heels, suede).
40   color_of_request_item(heels, beige).
41   companion_item(dress).
42   color_of_companion_item(dress, burgundy).
43   season(autumn).
44   event(cocktail_party).
```

Figure 1: Prompts used in our experiment

6.1. Results

Zero-shot Prompt. Without examples, the best-performing models are gemma3 27b and gemma3 12b (F1 = 0.36), outperforming deepseek-r1 70b (0.32) and qwen3 32b (0.26), despite the latter being designed with reasoning tasks in mind. However, even the highest F1-score of 0.36 indicates poor overall performance in this setting. This suggests that while some models can partially generalize the task from instructions alone, semantic parsing to Datalog predicates remains challenging without guidance. As an archetypal example, llama3.3 70b, a very recent and large model, shows clear difficulty interpreting the task in zero-shot conditions with a disappointing F1-score of 0.11.

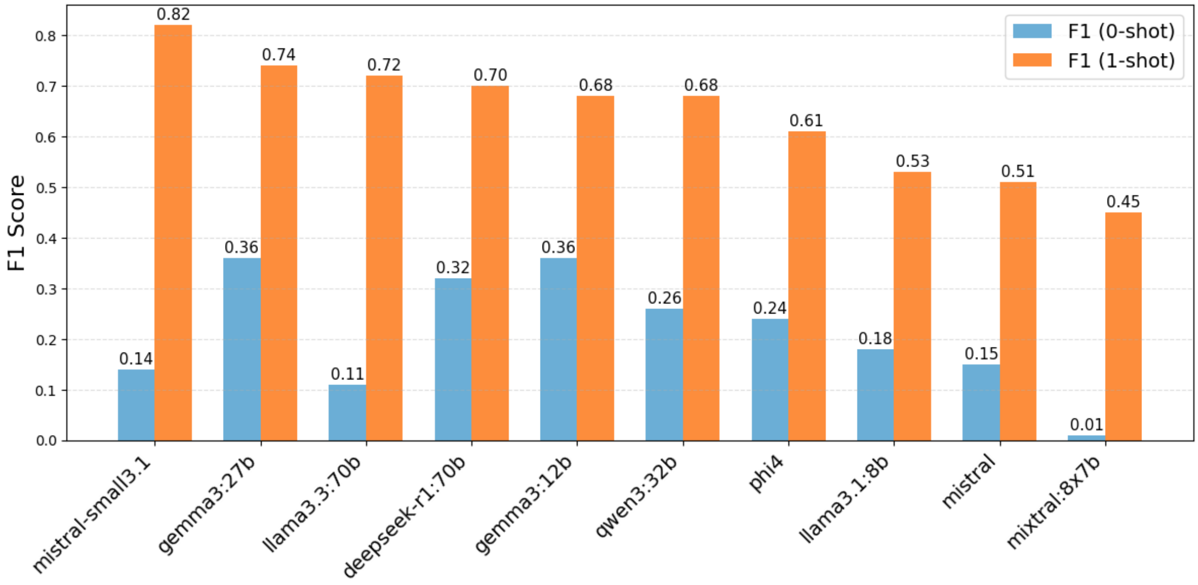


Figure 2: Performance comparison (F1-score) of LLMs in zero-shot and one-shot prompting settings

One-shot Prompt. With a single example, performance improves substantially. Mistral-small3.1 shows the greatest improvement, from 0.14 (zero-shot) to 0.82, achieving the best result overall (Figure 2). This is notable given its smaller size relative to larger models like deepseek-r1 70b and llama3.3 70b. Other strong performers include gemma3 27b (0.74), llama3.3 70b (0.72), and deepseek-r1 70b (0.70). While within the same family larger models tend to perform better (for instance, gemma3 27b performs better than gemma3 12b and llama3.3 7b performs better than llama3.1 8b), Figure 2 highlights that model size, particularly across models of different families, is not a good predictor of performance. In our case, mistrall-small3.1 and gemma3 27b outperform both llama3.3 70b and deepseek-r1 70b that have a larger number of parameters. This emphasizes that architecture and training quality matter as much as size. Interestingly, Gemma models maintain solid results in both zero-shot and one-shot settings, suggesting strong instruction-following capabilities likely supported by diverse training data. Overall, these findings confirm that LLMs benefit significantly from example-driven prompting, aligning with established observations that transformers generalize better in multi-task settings with explicit guidance.

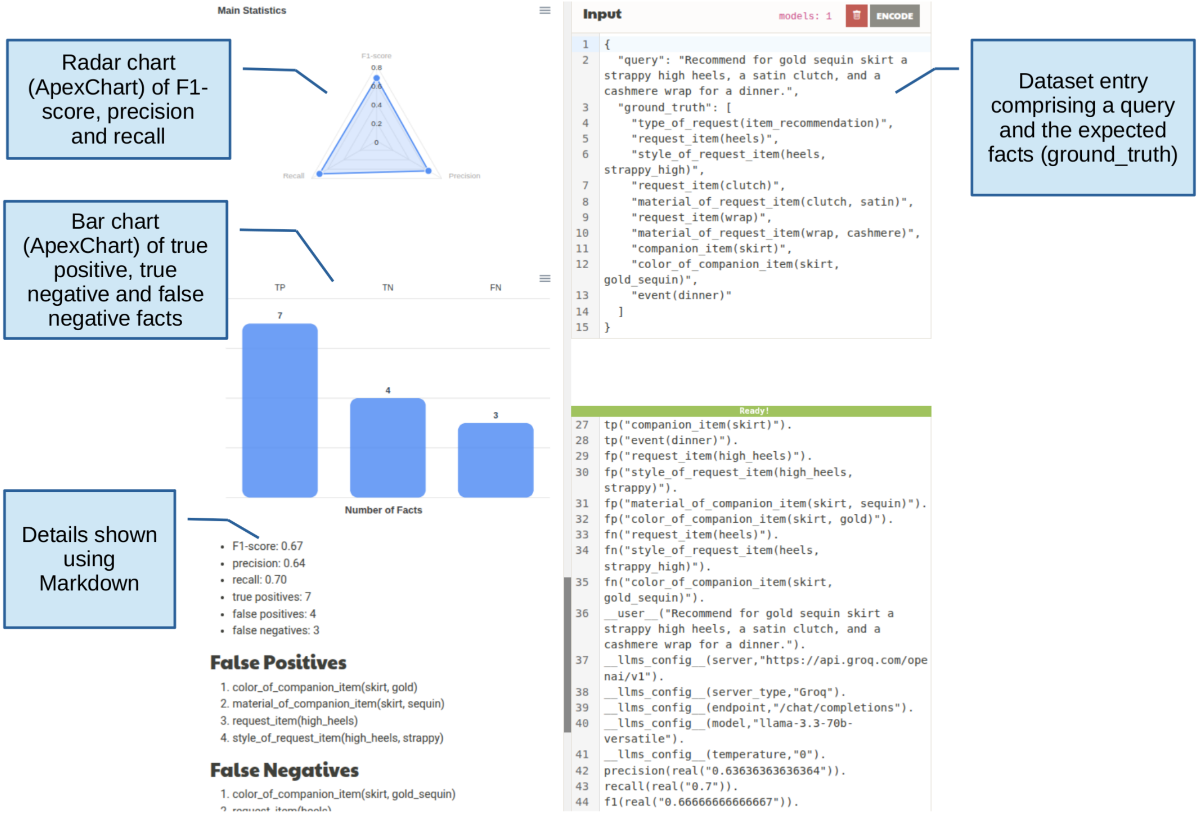
7. Interactive Evaluation via ASP Chef

To facilitate hands-on experimentation with our semantic parsing framework, we provide an executable ASP Chef recipe, i.e., a self-contained environment that enables users to test LLM-based Datalog generation and evaluate the results against the annotated ground truth. The recipe is hosted at https://asp-chef.alviano.net/s/ASVIN/ASPOCP25, and a screenshot focused on its input and output is shown in Figure 3.

7.1. Purpose and Setup

This recipe, titled “ASVIN First Report Playground”, allows users to explore how large language models translate natural language fashion queries into Datalog facts, following the schema and rules established in our dataset. It supports both zero-shot and one-shot prompting strategies and visualizes performance through automatic statistics calculation using ASP.

To run the recipe, users must:



Radar chart (ApexChart) of F1-score, precision and recall

Bar chart (ApexChart) of true positive, true negative and false negative facts

Details shown using Markdown

Dataset entry comprising a query and the expected facts (ground_truth)

```
{
  "query": "Recommend for gold sequin skirt a strappy high heels, a satin clutch, and a cashmere wrap for a dinner.",
  "ground_truth": [
    "type_of_request(item_recommendation)",
    "request_item(heels)",
    "style_of_request_item(heels, strappy_high)",
    "request_item(clutch)",
    "material_of_request_item(clutch, satin)",
    "request_item(wrap)",
    "material_of_request_item(wrap, cashmere)",
    "companion_item(skirt)",
    "color_of_companion_item(skirt, gold_sequin)",
    "event(dinner)"
  ]
}
```

```
tp("companion_item(skirt)").
tp("event(dinner)").
fp("request_item(high_heels)").
fp("style_of_request_item(high_heels, strappy)").
fp("material_of_companion_item(skirt, sequin)").
fp("color_of_companion_item(skirt, gold)").
fn("request_item(heels)").
fn("style_of_request_item(heels, strappy_high)").
fn("color_of_companion_item(skirt, gold_sequin)").
__user__("Recommend for gold sequin skirt a strappy high heels, a satin clutch, and a cashmere wrap for a dinner.").
__llms_config__(server,"https://api.groq.com/openai/v1").
__llms_config__(server_type,"Groq").
__llms_config__(endpoint,"/chat/completions").
__llms_config__(model,"llama-3.3-70b-versatile").
__llms_config__(temperature,"0").
precision(real("0.63636363636364")).
recall(real("0.7")).
f1(real("0.66666666666667")).
```

- F1-score: 0.67
- precision: 0.64
- recall: 0.70
- true positives: 7
- false positives: 4
- false negatives: 3

False Positives

1. color_of_companion_item(skirt, gold)
2. material_of_companion_item(skirt, sequin)
3. request_item(high_heels)
4. style_of_request_item(high_heels, strappy)

False Negatives

1. color_of_companion_item(skirt, gold_sequin)
2. request_item(heels)

Figure 3: Screenshot of the ASP Chef recipe used in the ASVIN project. The interface includes (top-right) the input panel showing a dataset entry with the natural language `query` and its expected Datalog `ground_truth`; (bottom-right) the extracted facts and configuration for LLM interaction; (left) the evaluation output comprising an F1-score radar chart, a bar chart of true/false positives and negatives, and a detailed Markdown summary of the system's performance.

- Register and configure a Groq API key, following the setup guide provided online (https://asp-chef.alviano.net/s/LLMs/getting-started).
- Optionally choose a different LLM model from the Groq model catalog by updating the model name in the *@LLMs/Config* ingredient. The recipe defaults to using the llama-3.3-70b-versatile model.
- Ensure that the temperature is fixed at 0 to maintain deterministic behavior.

7.2. Recipe Structure and Ingredients

The ASP Chef recipe consists of a sequence of ingredients (i.e., modular operations), each handling a key phase in the evaluation pipeline:

1. *Input and Prompt Encoding:* The recipe begins with an input JSON object representing a single dataset entry, consisting of a `query` and its associated `ground_truth`. A system message encodes the structured instructions to the model. If the user wishes to test the one-shot setting, a full example query with its Datalog translation is included as an additional system message. For zero-shot, this second message is omitted.
2. *User Message Extraction:* The natural language query is dynamically extracted from the input using JSONPath and used as the user message for the LLM interaction. This decouples prompt design from the data instance and enables automated testing across examples.

3. *LLM Interaction:* Once the prompt is assembled, the LLM is queried via the Groq API. The response is expected to be a pure list of Datalog facts, adhering to the rules specified in the prompt.

4. *Fact Extraction and Evaluation:* The predicted output is parsed and transformed into structured ASP facts. These are then compared against the `ground_truth` using ASP rules to compute true positives (TP), false positives (FP), and false negatives (FN).

5. *Metric Calculation:* Using ASP expressions, we calculate precision, recall, and F1-score, providing a clear view of how well the model has reproduced the intended semantics. The results are shown both as text and as radar/bar charts, offering a concise yet informative summary.

7.3. Exploration and Insight

This interactive setup serves as both a debugging interface and an educational tool. It allows users to (i) test prompt variations (e.g., 0-shot vs. 1-shot), (ii) switch between different LLMs, (iii) directly observe the syntactic correctness and semantic completeness of generated programs, and (iv) quantitatively assess model output through logic-based validation.

By combining prompt-based generation, ASP reasoning, and visual analytics, this recipe exemplifies a principled and interpretable way to study LLM behavior in structured parsing tasks. It also represents a prototype for future integrations with broader systems like LLMASP, where such evaluation modules could be reused in a domain-agnostic and scalable fashion.

8. Related work

The use of natural language interfaces in data-centric systems has received sustained attention over the past decades. In the database domain, numerous efforts have sought to enable users to express data queries in natural language, bridging the gap between lay users and structured query languages such as SQL [14]. Despite advances in natural language processing (NLP), substantial challenges persist in terms of semantic interpretation and accurate query generation, particularly when dealing with complex user intents and domain-specific knowledge.

One of the most studied approaches is text-to-SQL parsing, which aims to convert natural language queries into executable SQL statements [15]. However, the expressiveness of SQL is tightly bound to the underlying database schema and lacks access to implicit or contextual knowledge. For instance, a query that implicitly relies on common-sense knowledge (e.g., identifying products appropriate for Christmas) cannot be resolved purely through schema-bound SQL translation. Some researchers have proposed augmenting databases with knowledge graphs to address these gaps, but such solutions often suffer from scalability and maintenance limitations. A recent survey by Hong et al. [16] offers a comprehensive update on these evolving techniques and their limitations. In the ASVIN project, we adopt a different approach: rather than generating SQL queries, we extract semantic facts in the form of Datalog predicates. This has several advantages. First, it decouples query understanding from rigid schemas, enabling the assistant to operate on logical abstractions that are stable even when product catalogs evolve. Second, it supports symbolic reasoning, allowing ASVIN to combine user intent with regulatory constraints and stylistic compatibility rules in a principled, explainable manner. Finally, as shown in our experiments, open-source LLMs can generate these facts reliably using prompt engineering, even in zero- or one-shot settings, without requiring access to large SQL-based training corpora.

Closely related to our objective, Shaw et al. [17] investigated compositional semantic parsing, highlighting how complex queries can be decomposed into smaller semantic units to improve interpretation and system robustness. Their results underscore the importance of generalization in parsing models, particularly in handling linguistic variability and nested semantic structures. This line of work aligns with our own focus on breaking down fashion-related queries into structured Datalog facts using predefined predicate templates.

With the advent of LLMs, new opportunities have emerged for improving semantic parsing. Brown et al. [18] demonstrated that LLMs can significantly outperform earlier rule-based and template-based approaches, especially for complex SQL features like multi-table joins and nested queries. Notably, their integration of user feedback loops allows for dynamic refinement of generated queries, improving both interpretability and accuracy. More recently, Schneider et al. [19] evaluated LLMs on generating structured queries (e.g., graph queries) from dialogue inputs, finding that few-shot prompting and fine-tuning substantially enhance performance, particularly for smaller models with weak zero-shot capabilities.

In parallel with these developments, our previous work on LLMASP [20, 21] introduced a general-purpose framework for connecting LLMs with symbolic reasoning via configurable prompting strategies. LLMASP supports domain-independent applications through two configurable layers: application files, which define the target logic task, and behavior files, which provide meta-prompts and formatting instructions. This modularity enables LLMASP to operate across diverse problem domains (including planning, diagnosis, and classification) without changing its internal structure.

The present work differs from LLMASP in several key respects. First, it is domain-specific, targeting semantic parsing within the fashion retail space as part of the ASVIN project. Second, we adopt a static prompt design, hardcoding a highly specialized template for Datalog generation, rather than relying on dynamically assembled meta-prompts. Third, while LLMASP was evaluated exclusively on LLaMA 3 models (8B and 70B), our analysis spans a broader range of open-source LLMs, including Mistral, Gemma, Qwen, Phi, and DeepSeek, allowing for a more comprehensive comparison of model behaviors under constrained logical tasks.

Despite these differences, we consider LLMASP a natural next step for future extensions of our work. Integrating the models and task defined here into the LLMASP pipeline would allow us to investigate whether the best-performing models in our hard-coded prompt setting (notably, mistrall-small3.1 and gemma3:27b) retain their advantage under dynamic prompting strategies. Such an integration could also offer insights into the trade-offs between fixed-task optimization and prompt flexibility, as well as the impact of meta-prompt design on semantic fidelity and generalization.

In summary, our study builds on a growing body of research at the intersection of LLM-based semantic parsing and structured query generation, contributing a focused evaluation in a practical, regulated domain while pointing toward broader generalization through systems like LLMASP.

9. Conclusions

This paper presented a preliminary evaluation of open-source LLMs for semantic parsing in the fashion retail domain, a core component of the ASVIN project. The goal is to translate natural language queries into Datalog facts to enable explainable, regulation-compliant virtual assistance. Fashion-related queries present unique challenges due to their reliance on style, context, and implicit semantics. Our results show that zero-shot prompting yields limited accuracy, highlighting the difficulty of the task. However, the substantial improvement in one-shot settings demonstrates that the problem is feasible when supported by well-crafted prompts, validating the importance of prompt engineering.

These findings open several avenues for future work. Tools like LLMASP have shown that predicate-focused prompts can improve control and accuracy; adapting our task to this framework is a natural next step. Moreover, grammar-constrained generation techniques (though not explored here) represent a promising direction for ensuring syntactic and semantic correctness. A current limitation is the lack of domain-specific datasets. Our pilot dataset is a first step, and we plan to extend it with greater variety and real-world data from fashion companies. Additionally, we envision developing fashion knowledge bases that encode concepts such as styles and outfit logic. These could guide ASVIN in interactive sessions, helping the assistant request missing information and reason over product catalogs more effectively.

Overall, our results demonstrate that semantic parsing in this domain is both challenging and achievable, laying the groundwork for intelligent assistants that blend language, logic, and fashion expertise.

Acknowledgments

This work was supported by the Italian Ministry of University and Research (MUR) under PRIN project PRODE “Probabilistic declarative process mining”, CUP H53D23003420006, under PNRR project FAIR “Future AI Research”, CUP H23C22000860006, under PNRR project Tech4You “Technologies for climate change adaptation and quality of life improvement”, CUP H23C22000370006, and under PNRR project SERICS “SEcurity and RIghts in the CyberSpace”, CUP H73C22000880001; by the Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by the Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); under PN RIC project ASVIN “Assistente Virtuale Intelligente di Negozio” (CUP B29J24000200005); and by the LAIA lab (part of the SILA labs). Mario Alviano is member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-4o for grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

[1] S. Zhang, B. Peng, X. Zhao, B. Hu, Y. Zhu, Y. Zeng, X. Hu, Llasa: Large language and e-commerce shopping assistant, arXiv preprint arXiv:2408.02006 (2024).

[2] Y. Guan, D. Wang, Z. Chu, S. Wang, F. Ni, R. Song, L. Li, J. Gu, C. Zhuang, Intelligent virtual assistants with llm-based process automation, arXiv preprint arXiv:2312.06677 (2023).

[3] K. I. Roumeliotis, N. D. Tselikas, D. K. Nasiopoulos, LLMs in e-commerce: a comparative analysis of GPT and LLaMA models in product review evaluation, Natural Language Processing Journal 6 (2024) 100056.

[4] N. Vedula, O. Rokhlenko, S. Malmasi, Question suggestion for conversational shopping assistants using product metadata, in: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2024, pp. 2960–2964.

[5] F. Cheng, H. Li, F. Liu, R. van Rooij, K. Zhang, Z. Lin, Empowering llms with logical reasoning: A comprehensive survey, arXiv preprint arXiv:2502.15652 (2025).

[6] F. Alotaibi, A. Kulkarni, D. Zhou, Graph of Logic: Enhancing LLM Reasoning with Graphs and Symbolic Logic, in: 2024 IEEE International Conference on Big Data (BigData), IEEE, 2024, pp. 5926–5935.

[7] Y. Wan, W. Wang, Y. Yang, Y. Yuan, J.-t. Huang, P. He, W. Jiao, M. R. Lyu, LogicAsker: Evaluating and improving the logical reasoning ability of large language models, arXiv preprint arXiv:2401.00757 (2024).

[8] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-year Perspective, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[9] I. Niemelä, Logic programming with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.

[10] M. Kulkarni, P. Tangarajan, K. Kim, A. Trivedi, Reinforcement learning for optimizing rag for domain chatbots, arXiv preprint arXiv:2401.06800 (2024).

[11] K. Olawore, M. McTear, Y. Bi, Development and Evaluation of a University Chatbot Using Deep Learning: A RAG-Based Approach, in: International Symposium on Chatbots and Human-Centered AI, Springer, 2024, pp. 96–111.

[12] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, ASP-Core-2 Input Language Format, TPLP 20 (2020) 294–309. URL: https://doi.org/10.1017/S1471068419000450. doi:10.1017/S1471068419000450.

[13] M. Gelfond, V. Lifschitz, Logic programs with classical negation, in: D. Warren, P. Szeredi (Eds.), Logic Programming: Proc. of the Seventh International Conference, 1990, pp. 579–597.

[14] C. Ma, B. Molnár, Ontology Learning from Relational Database: Opportunities for Semantic Information Integration, Vietnam Journal of Computer Science 09 (2022) 31–57. doi:10.1142/S219688882150024X.

[15] G. Katsogiannis-Meimarakis, G. Koutrika, A survey on deep learning approaches for text-to-SQL, The VLDB Journal 32 (2023) 905–936.

[16] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, X. Huang, Next-generation database interfaces: A survey of llm-based text-to-sql, arXiv preprint arXiv:2406.08426 (2024).

[17] P. Shaw, M.-W. Chang, P. Pasupat, K. Toutanova, Compositional generalization and natural language variation: Can a semantic parsing approach handle both?, arXiv preprint arXiv:2010.12725 (2020).

[18] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[19] P. Schneider, M. Klettner, K. Jokinen, E. Simperl, F. Matthes, Evaluating large language models in semantic parsing for conversational question answering over knowledge graphs, arXiv preprint arXiv:2401.01711 (2024).

[20] M. Alviano, L. Grillo, Answer Set Programming and Large Language Models interaction with YAML: Preliminary Report, in: CILC, CEUR Workshop Proceedings, CEUR-WS.org, 2024.

[21] M. Alviano, L. Grillo, F. L. Scudo, L. A. Rodriguez Reiners, ntegrating Answer Set Programming and Large Language Models for Enhanced Structured Representation of Complex Knowledge in Natural Language, in: IJCAI 2025, Montreal, Canada, August 16-22, 2025, ijcai.org, 2025.