Integrating Answer Set Programming and Large Language Models for Enhanced Structured Representation of Complex Knowledge in Natural Language

Mario Alviano , **Lorenzo Grillo** , **Fabrizio Lo Scudo** and **Luis Angel Rodriguez Reiners**
DeMaCS, University of Calabria, 87036 Rende (CS), Italy
{mario.alviano, fabrizio.loscudo, luis.reiners}@unical.it

Abstract

Answer Set Programming (ASP) and Large Language Models (LLMs) have emerged as powerful tools in Artificial Intelligence, each offering unique capabilities in knowledge representation and natural language processing, respectively. In this paper, we combine the strengths of the two paradigms with the aim of improving the structured representation of complex knowledge encoded in natural language. In a nutshell, the structured representation is obtained by combining syntactic structures extracted by LLMs and semantic aspects encoded in the knowledge base. The interaction between ASP and LLMs is driven by a YAML file specifying prompt templates and domain-specific background knowledge. The proposed approach is evaluated using a set of benchmarks based on a dataset obtained from problems of ASP Competitions. The results of our experiment show that ASP can sensibly improve the F1-score, especially when relatively small models are used.

1 Introduction

Large Language Models (LLMs) and Answer Set Programming (ASP) represent two distinct but complementary paradigms in Artificial Intelligence (AI). LLMs, such as GPT [Brown and et al., 2020], PaLM [Chowdhery and et al., 2023], and Llama [Touvron and et al., 2023], have transformed natural language processing (NLP) by achieving unprecedented levels of fluency and capability in textual data. LLMs excel at various NLP tasks [Jin *et al.*, 2024; Zhang *et al.*, 2023], such as language generation, summarization, and sentiment analysis. In contrast, ASP [Marek and Truszczynski, 1999; Niemelä, 1999], a declarative programming paradigm extending Datalog under answer set semantics [Gelfond and Lifschitz, 1990], excels in knowledge representation and logical reasoning, making it fundamental for AI systems that require robust inference capabilities, including planning and scheduling [Cappanera *et al.*, 2023; Cardellini *et al.*, 2024], as well as diagnosis and configuration [Wotawa, 2020; Taupe *et al.*, 2021]. Capabilities that LLMs do not possess [Li *et al.*, 2024].

Example 1. *Let us consider a prompt asking to LLMs to address a nontrivial reasoning task such as the following:*

You are organizing a networking event with several attendees, and you want to **form the largest possible group where every person in the group knows every other person.**
The attendees have provided the following information about who they know:
Alice knows Bob, and Evan.
Bob knows Charlie, Diana, Evan, and Fiona.
Charlie knows Diana, Evan, Fiona, and George.
Diana knows Evan, Fiona, and George.
Evan knows George.
Fiona knows George, and Henry.
George knows Evan, and Henry.
Henry knows George.
Question: What is the largest group of attendees where everyone knows every other person in the group?

An answer provided by chatgpt.com (using GPT-4o from the web interface or gpt-4o-mini-2024-07-18 via API) is the following:

The largest group where everyone knows everyone else is the group consisting of **Bob, Charlie, Diana, Evan, and Fiona**.

The answer is wrong, as Evan does not know Fiona. (We obtained wrong answers also with Meta Llama 3.1:70b.) ■

Recognizing the complementary strengths of LLMs' linguistic abilities and ASP's reasoning capabilities, this paper proposes an approach that leverages the synergies between these two paradigms, inspired by recent works in the literature [Basu *et al.*, 2021; Zeng *et al.*, 2024]. Our objective is to develop a unified system that seamlessly integrates natural language processing with logical inference, allowing AI applications to adeptly handle the complex interplay between textual data and logical structures. In particular, we outline a method for encoding specific domain knowledge into input prompts using a YAML-based format, which allows LLMs to produce relational facts that are used by ASP for reasoning. An overview of the main pipeline addressed by our system is shown in Figure 1. The system receives input consisting of two YAML files: the *behavior file* provides general system configuration, and the *application file* details domain knowledge. Additionally, the input includes a database (i.e., relational facts) and a user request expressed in natural language. Behavior and application files are combined to obtain prompt

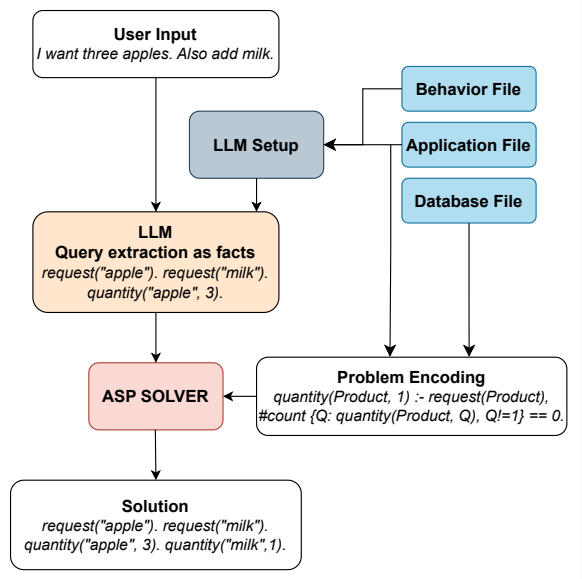

Figure 1: Graphical representation of the LLMASP pipeline. The pipeline begins with a user query formulated in natural language. The system also receives two YAML files: one specifying the system's behavior and another detailing the context of the specific application under development. The behavior YAML file provides initial prompts for the LLM, which are enriched with contextual details derived from the application YAML file and the user's query. These enriched prompts guide the LLM in extracting relevant information from the user input, transforming it into structured factual representations. These facts are subsequently integrated into an ASP framework, where they are combined with the existing knowledge base and database, and processed by the solver to compute an answer set.

templates for the LLM, which are completed by the user input. The complete prompts ask to extract data from the user input and represent them in the form of facts. Facts are then combined with a knowledge base (an ASP program) and processed by an ASP solver to obtain an answer set. (Optionally, and not discussed here, the answer set is then combined with the prompts obtained by the two YAML files to express the answer set in natural language.) One possible use case for our tool is implementing a chatbot for an e-commerce site. In this scenario, users could interact with the chatbot to place items in their shopping cart. The requests by the users are mapped to facts and combined with an ASP program. Actions are performed based on the computed answer set.

Example 2 (Continuing Example 1)**.** *Our system uses prompts that instruct LLMs on extracting data rather than addressing reasoning tasks.*

You are a Natural Language to Datalog translator.
I will provide text in the format [INPUT]input[/INPUT], and instructions on how to map the INPUT to facts. The expected OUTPUT format will be given by [OUTPUT]predicate(terms).[/OUTPUT].
Here is some context that you MUST analyze and remember.
If there is a graph, represent it with the predicates node/1 and edge/2. For example, if A and B are connected, use node("A"). node("B"). edge("A","B").
[INPUT]You are organizing...[/INPUT] Process the INPUT according to the following instructions:
List all the edges from source (double quoted string) to target (double quoted string). The output format is [OUTPUT]edge(source, target)[/OUTPUT]
List all the nodes. Associate to every node its ID (double quoted string). The output format is [OUTPUT]node(id)[/OUTPUT]

Our system first obtains a relational representation of the graph in input:

```
node("Alice"). node("Bob"). ...
edge("Alice","Bob"). edge("Alice","Evan").
  ...
```

The above facts are then coupled with an ASP program to compute a maximal clique:

```
{in(X)} :- node(X).
:- in(X), in(Y), X < Y, not edge(X,Y).
:~ node(X), not in(X). [1@1, X]
```

Finally, our system represents the answer set in natural language:
The maximal clique consists of **Charlie, Diana, Evan, and George**.

(For this example, we coupled our system with Meta Llama 3.1:70b and the behavior file EIN; see Section 4.) ■

This paper also reports on the definition of a dataset focused on data extraction that we obtained from the domains used in several ASP Competitions, and which we used to evaluate some behavior and application files, as well as the use of ASP to improve the generation of facts. In particular, our dataset comprises test cases obtained from 15 different domains, which we approached with 10 behavior files characterized by combinations of specific features, including generic extraction examples, predicate and term format descriptions, repeated task instructions, and guidance for handling missing answers. Additionally, we analyzed two application file sets, providing either atom formats, or problem descriptions alongside atom formats, as defined by the ASP Competitions websites. Our analysis supports our conjecture that approaching the data extraction task in multiple stages and with the help of ASP is more effective than within a single request to LLMs. Specifically, the F1-score improves from 73.7% to 80.9% when Meta Llama 3.1:8b is used, and from 90.4% to 98.1% using Meta Llama 3.1:70b. Further improvements are obtained by instructing LLMs to extract explicit knowledge in the input, and then using ASP to generate implicit knowledge based on the specific domain of interest.

The remainder of this paper is structured as follows: Section 2 introduces background notions regarding LLMs, ASP and YAML. Section 3 defines the format of behavior and application files that are used in the extraction pipeline, which is also discussed in this section. Section 4 presents the LLMASP prototype, the definition of a dataset from domains of ASP Competitions, and the associated empirical evaluation. Related work is discussed in Section 5 and conclusions are given in Section 6.

2 Background

2.1 Large Language Models

Large Language Models (LLMs) are sophisticated AI systems designed to process and generate human-like text. These models are based on deep learning architectures, typically *transformers*, and are trained on vast amounts of text data to learn complex patterns and structures of language. In this article, LLMs are used as black box operators on text (functions that take text in input and produce text in output). At each interaction with an LLM, the generated text is influenced by all previously processed text (the *history* of the interaction), and the inherent stochastic nature of the model. The text in input is called *prompt*, and the text in output is called *generated text* or *response*.

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

> Encode as Datalog facts the following sentences:
> I want three apples. Also add milk.

A response produced by Meta Llama 3.1:70b comprises

```
want(I, "three_apples"). add("milk").
```

or

```
want(I, Apples). quantity(Apples, three).
add(Milk).
```

as a proposed alternative. Although it is a good starting point, the LLM should explicitly be instructed on the specific format required for encoding facts. ***Which predicates should be used? Should the quantity be represented as an integer? Should objects be represented as double quoted strings?*** *In this article, we aim at gaining more control on the output produced by the LLM.* ■

2.2 Answer Set Programming

We formally introduce the notion of fact and refer to the ASP-Core-2 format [Calimeri *et al.*, 2020] for other constructs. Let $\mathbf{P}$ be a fixed nonempty set of *predicate names*. Predicates are associated with an *arity*, a non-negative integer. Let $\mathbf{C}$ be the set of *constants*, that is, integers and (double-quoted) strings. A *fact* is of the form $p(\overline{c})$, where $p \in \mathbf{P}$, 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. 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 [Gelfond and Lifschitz, 1990]).

Example 4. *Let us consider the following program:*

```
request("apple"). request("milk").
quantity("apple", 3).
quantity(Product, 1) :- request(Product),
  #count{Q: quantity(Product,Q), Q!=1} == 0.
```

The first two lines above comprise three facts (e.g., the input database). After that, there is a rule defining 1 as the default quantity. In this case, there is one output database extending the facts with `quantity("milk", 1).` ■

2.3 YAML

YAML (YAML Ain't Markup Language; https://yaml.org/spec/1.2.2/) is a human-readable data serialization format commonly used for configuration files, data exchange, and representation of structured data. In this article, we focus on the following restricted fragment: A *scalar* is any number or string (possibly quoted). A *block sequence* is a sequence of entries, each one starting by a dash followed by a space. A *mapping* is a sequence of key-value pairs, each pair using a colon and space as separator, where keys and values are scalars. A scalar can be written in *block notation* using the prefix `|` (if not a key).

Example 5. *Below is a YAML document:*

```
name: LLMASP
combined technologies:
- Answer Set Programming
- Large Language Models
description: |
  LLMASP combines ASP and LLMs...
  LLMs are used to extract data...
```

It encodes a mapping with keys `name`*,* `combined technologies` *and* `description`*. Key* `name` *is associated with the scalar* `LLMASP`*. Key* `combined technologies` *is associated with the list* `[Answer Set Programming, Large Language Models]`*. Key* `description` *is associated with a scalar in block notation.* ■

3 LLMASP Configuration and Pipeline

The architecture of LLMASP is shown in Figure 1 [Alviano and Grillo, 2024; Alviano *et al.*, 2024]. LLMASP takes in input two YAML files, namely the behavior file B and the application file A, together with a database file D (comprising facts) and a request text T (expressed by the user in natural language). A set F of facts is populated and a database is given in output. This section defines B and A, as well as the LLMASP pipeline.

Behavior File. The behavior file specifies global behavior settings for LLMASP, as tone, style and general instructions for the LLM. It comprises two keys, namely `preprocessing` and `postprocessing`. The preprocessing section contains the following properties: (i) `init`, whose value is used to provide general instructions to the LLM; (ii) `context`, whose value must include the string `{context}`, to be combined with contextual information regarding an application of interest; (iii) `mapping`, whose value must include the strings `{input}`, `{instructions}` and `{atom}`, to be combined with the instructions on how to extract specific atoms from the user input. The postprocessing section is similar.

Application File. The second YAML file contains domain-specific guidelines, as a description of the context and mappings between facts and their corresponding natural language translation. In details, it contains three sections, namely `preprocessing`, `knowledge base`, and `postprocessing`. The values associated with `preprocessing` and `postprocessing` are mappings where keys are either facts or the special value _ (used for

providing a context), and values are scalars. The value associated with `knowledge base` is a program.

Pipeline. The pipeline implemented by LLMASP to map natural langauge to ASP facts comprises four steps, namely **P1**–**P4**, reported below. We use the following notation to define such steps: For a, b, c being strings, let $a[b \mapsto c]$ denote the string obtained from a by replacing all occurrences of b with c. Given a behavior file B, let $pre_B(\alpha)$ be the value associated with α in the `preprocessing` mapping of B, where α is among `init`, `context` and `mapping`. For an application file A, let $pre_A(\alpha)$ be the value associated with α in the `preprocessing` mapping, where α is either an atom or _. Finally, let kb_A be the program in `background knowledge`.

P1. The history of the LLM is set to the prompt $pre_B(\texttt{init})$.

P2. If $pre_A(\_)$ is defined, the history of the LLM is extended with the prompt $pre_B(\texttt{context})[\{\texttt{context}\} \mapsto pre_A(\_)]$.

P3. For each defined $pre_A(\alpha)$, the LLM is invoked with the prompt $pre_B(\texttt{mapping})[\{\texttt{input}\} \mapsto T][\{\texttt{atom}\} \mapsto \alpha][\{\texttt{instructions}\} \mapsto pre_A(\alpha)]$. Facts in the response are collected in F. Everything else is ignored.

P4. An answer set of $kb_A \cup \{\alpha. \mid \alpha \in D \cup F\}$ is searched.

Example 6. *Let us consider a behavior file B with the following `preprocessing` section:*

```
init: |
  You are a Natural Language to Datalog translator.
  I will provide text in the format [INPUT]input[/INPUT],
  and instructions on how to map the INPUT to facts.
  The expected OUTPUT format will be given by
  [OUTPUT]predicate(terms).[/OUTPUT].
context: Here is some context that you MUST analyze
  and remember: {context}
mapping: |
  [INPUT]{input}[/INPUT] Process the INPUT according
  to the following instructions: {instructions}
  The output format is [OUTPUT]{atom}[/OUTPUT]
  If the answer is not in the INPUT, reply NONE.
```

As for the application file A, let us consider the following `preprocessing` section:

```
_: You have to extract data about the requested products.
  Ignore plural, always write product names in singular.
  Double quote names.
request("product"): List all the products in the request.
quantity("product", value): value is the quantity
  associated with "product" if explicitly given in the request.
```

*Files A and B are combined to extract the facts in Example 4 (plus the ignored atom `quantity("milk", NONE)`). In particular, the execution of step **P2** uses*

```
Here is some context that you MUST analyze and remember:
You have to extract data about the requested products. Ignore
plural, always write product names in singular. Double quote
names.
```

*As for **P3**, it is executed twice:*

```
[INPUT]I want three apples. Also add milk.[/INPUT]
Process the INPUT according to the following instructions:
List all the products in the request. The output format is
[OUTPUT]request("product")[/OUTPUT]
If the answer is not in the INPUT, reply NONE.
```

```
[INPUT]I want three apples. Also add milk.[/INPUT]
Process the INPUT according to the following instructions:
value is the quantity associated with "product" if ex-
plicitly given in the request. The output format is
[OUTPUT]quantity("product", value)[/OUTPUT]
If the answer is not in the INPUT, reply NONE.
```

The extracted facts are combined with the knowledge base (e.g., the rule in Example 4) to obtain an output database. ■

ID	*Problem Name*	*Facts*	*1-ary*	*2-ary*	*3-ary*
CMSL	Connected Max.-density Still Life	1	1	0	0
CM	Crossing Minimization	53	1	52	0
GG	Graceful Graphs	8	0	8	0
GC	Graph Colouring	109	54	55	0
IS	Incremental Scheduling	77	24	53	0
KT	Knight Tour	3	1	0	0
KTH	Knight Tour with Holes	9	1	8	0
L	Labyrinth	290	1	257	32
MCP	Maximal Clique Problem	19	11	8	0
NM	No Mystery	31	8	15	8
PU	Partner Units	24	9	15	0
PPM	Permutation Pattern Matching	17	1	16	0
RR	Ricochet Robots	22	9	0	13
S	Sokoban	44	32	4	8
SM	Stable Marriage	16	0	0	16
VLP	Valves Location Problem	28	12	8	8

Table 1: Average number of facts in the instances of each tested domain (overall, and per arity).

4 Implementation and Experiment

Our LLMASP implementation is powered by Ollama, OpenAI API and CLINGO [Gebser *et al.*, 2011]. Here, we focus on two models, Llama 3.1 with 8 and 70 billions parameters. In order to assess LLMASP empirically, we defined a dataset using domains from ASP Competitions [Gebser *et al.*, 2020]. Specifically, we use the description of the domain and format of the facts available online, and systematically generate text representations (of portions) of the instances. We therefore aim at extracting the facts from the generated text, and adopt F1-score as a measure of quality (precision and recall are additionally reported in the supplementary material).

The dataset consists of 32 test cases for each domain listed in Table 1. The table provides the average number of facts to be extracted for each problem instance, offering some insight into the complexity of each domain (which however also depends on other aspects such as its textual descriptions and the generated instances). In particular, the generation of text is obtained by randomly applying templates (among several alternatives) to randomly selected facts. The templates are structured to ensure variability in the generated natural language descriptions. For example, the application of some templates of *Incremental Scheduling* (IS) to the facts

```
job(1). job(3). job(10). job(17). job(19).
deadline(3,14). deadline(18,36).
job_len(1,14). importance(1,2).
precedes(13,14). precedes(10,11).
precedes(17,18). precedes(19,20).
device(1). job_device(1,1).
offline_instance(1,1).
```

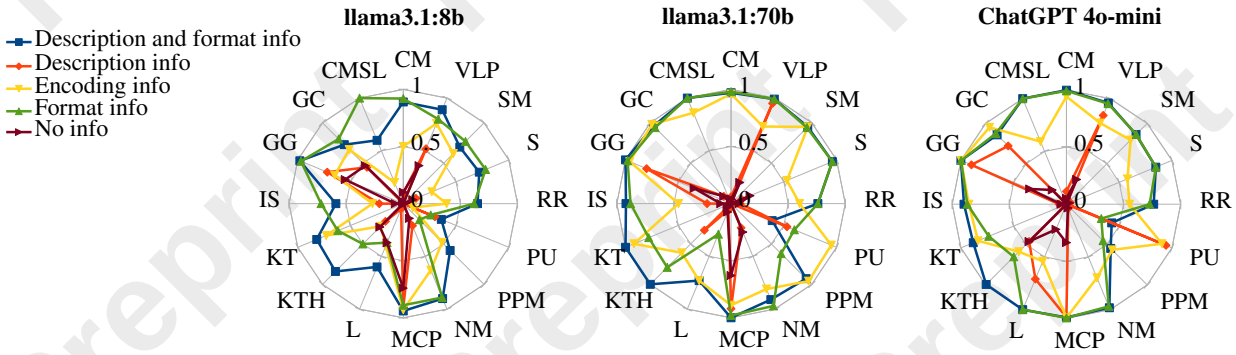

Figure 2: F1-score obtained by generating facts with Llama 3.1 models (8b and 70b) and ChatGPT (4o–mini), and prompts possibly providing the description of the domain, the format of atoms and the ASP program (as given in the ASP Competitions websites).

may result to the following text:

> The job 3 has a deadline 14. The completion time of job 13 must occur before the start time of 14. The job 1 has an importance of 2, must be executed by device 1, needs 14 timesteps to be performed. Job 19 should end before 20 starts. Device 1 has instance one offline. Job 17 must finish before the start time of 18. Job 10 must finish before the start time of 11. The job 18 has a deadline 36.

First of all, we try the prompt suggested by Open AI to parse unstructured data (https://platform.openai.com/docs/examples/default-parse-data), adapted to our format:

> You will be provided with unstructured data, and your task is to parse it into Datalog facts format.

As shown in Figure 2, with no information about the format of the facts to extract, both models exhibit a low F1-score (20.1% for 8b and 12.8% for 70b). A small improvement is obtained by providing the description of the domain (23.8% and 27.3%), and a more sensible improvement is obtained by providing the format of the facts to extract (69.4% and 84.1%). Including both descriptions and formats slightly improve the extraction process on average (73.7% and 90.4%). Alternatively, including in the prompt the official ASP program from the ASP Competitions results into relatively good F1-scores (49.1% and 80.6%), which however do not reach the previously obtained values. The figure also reports data regarding the ChatGPT 4o–mini, which are similar to those obtained for 70b: 12.3% when no information is provided; 36.5% when the domain is described; 82.9% when the format is detailed; 87.6% when both the domain and the format are given; 76.0% when the ASP program is included. Analyzing these first results, we conclude that a good description of the format of the facts to extract is paramount, while the description of the domain itself and the ASP program may help but are not fundamental. In the following, we employ LLMASP to test several prompts, by combining behavior and application files with different features.

We consider 10 behavior files having (some of) the following features: a generic extraction example in the context (E); a description of predicate and terms format in the context (F); repeated instructions on the extraction task in the mapping (I); indication in the context to reply NONE if the answer is missing (N); repeated indication on NONE in the mapping (R). We denote behavior files by the concatenation of their features, e.g., EF for a behavior file having examples, and the format of predicates and terms in the context. We consider sets of application files whose context provide the following information (as given in the ASP Competitions websites): (A1) format of atoms; (A2) description of the problem and format of atoms. Results are shown in Figure 3. Regarding A1, we observe that LLMASP improves the average F1-score of the 8b model up to 80.9% when the EINR behavior file is used; the other behavior files perform similarly. For the 70b model, F1-score reaches a pick of 96.8% with the EIN behavior file, with the other behavior files performing anyhow better than the LLM without LLMASP (94.0% for F is the worst result in this set). Moving on A2, we observe no significant difference when the 8b model is used, and a minimal improvement for the 70b model (registering the best average F1-score, i.e., 98.1% for EIN). All in all, we conclude that F is not necessary and may even confuse the 8b model; N helps the LLMs to handle missing answers, while R does not; I is also beneficial, likely because it simplifies the attention mechanism of the LLMs; E tends to improve the F1-score. Therefore, we select EIN as the best behavior file in this experiment. We also acknowledge a small improvement in using A2 instead of A1, possibly thanks to some contextualized examples.

Our last benchmark involves domains for which part of the generation process can be addressed in ASP after some facts are extracted. We therefore encode such parts in the `knowledge base` section of the application files. In more details, in L we replace field/2 by size/1 and the ASP rule

```
field(X,Y) :- size(N), X = 1..N, Y = 1..N.
```

Similarly, in NM and S we replace step/1 by steps/1 and

```
step(I) :- steps(N), I = 1..N.
```

Finally, in RR we modify the extraction of dim/1, replace barrier/3 by barrier/5, and employ the following ASP rules:

```
dim(I-1) :- dim(I), I > 1.
barrier(X,Y,A,B,D) :- barrier(A,B,X,Y,D).
barrier(X,Y,D) :- barrier(X,Y,X-1,Y,D),
                  D == west.
barrier(X,Y,D) :- barrier(X,Y,X+1,Y,D),
```

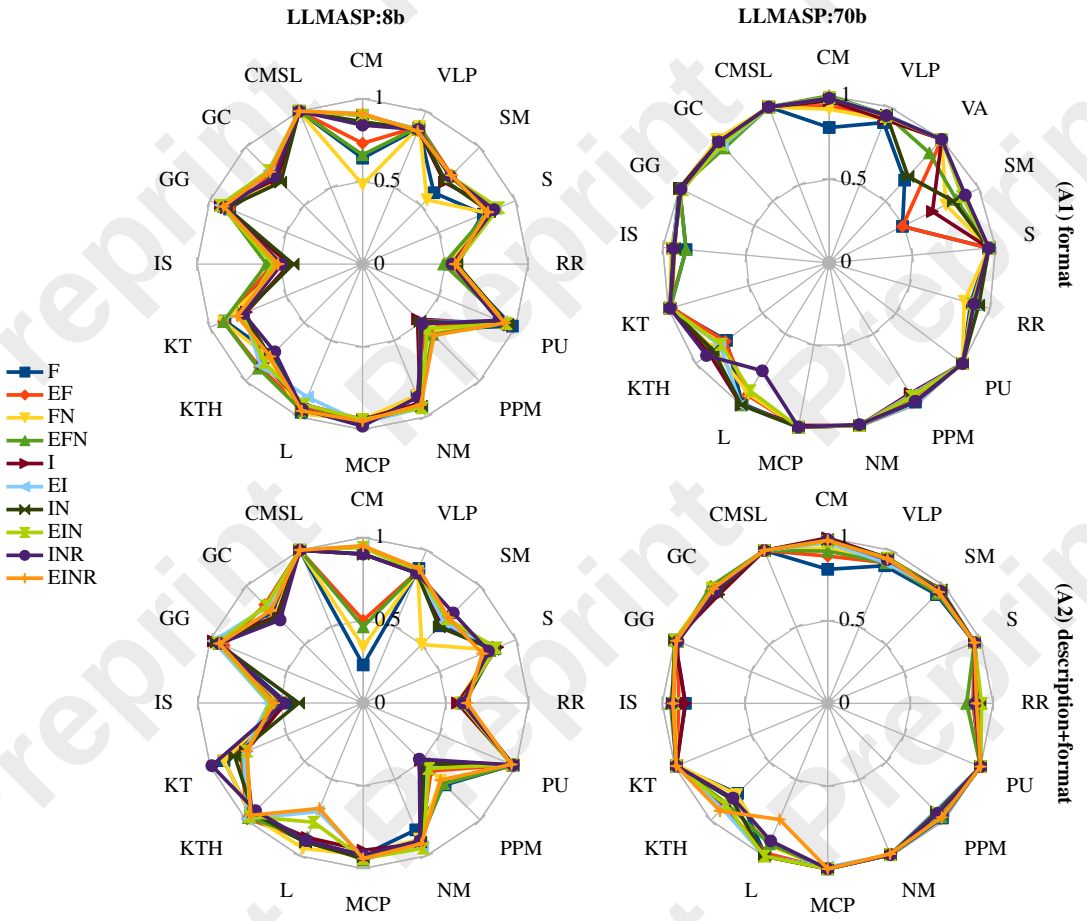

Figure 3: F1-score obtained by generating facts with LLMASP using Llama 3.1 models (8b and 70b), and application files providing the format of atoms and possibly the description of the domain.

```
                D == east.
barrier(X,Y,D) :- barrier(X,Y,X,Y-1,D),
                D == north.
barrier(X,Y,D) :- barrier(X,Y,X,Y+1,D),
                D == south.
```

In the program above, the definition of barrier/3 is of particular interest. In fact, we observed that often the tested LLMs have difficulties to associate a barrier between $(x-1, y)$ and (x, y) toward direction west with location $(x-1, y)$. We therefore expect better results by first extracting a more syntactic fact such as `barrier(x-1,y,x,y,west)` and then employing ASP to generate the correct fact `barrier(x-1,y,west)`. Results are shown in Table 2. We observe a sensible improvement for RR, especially for the 8b model. With a few exceptions, the use of ASP is beneficial for the generation process.

5 Related Work

LLMs (e.g., ChatGPT [Brown and et al., 2020], PaLM [Chowdhery and et al., 2023], and Llama [Touvron and et al., 2023]) excel in sequence-to-sequence (seq2seq) tasks, where a text input triggers a text output generated by the

	Beha. File	(A1) format				(A2) description+format			
		L	NM	RR	S	L	NM	RR	S
8b	F	97+2	86	54+37	79−1	91+7	83+5	61+32	86−2
	EF	94+6	91+1	52+43	83+9	89+10	91+2	58+38	86+5
	FN	98+1	86	53+38	81−2	95+3	89+1	57+36	84−1
	EFN	96+3	91	49+46	83+7	88+11	90+2	59+38	83+9
	I	92+7	91−2	56+36	86	88+11	92+2	57+37	88
	EI	87+13	94	55+41	87+3	71+29	94	60+38	85+5
	IN	96+3	93−2	57+36	83	91+8	91+4	61+33	85+2
	EIN	92+8	94	56+39	89−1	78+22	95	59+39	87−2
	INR	95+4	87+3	54+40	86+3	90+9	90	60+35	82+3
	EINR	96+4	92+1	56+40	82+7	69+31	92	63+33	78+12
70b	F	99+1	99	88+10	96	97+3	98+1	92+7	96
	EF	98+2	99	88+12	96	100	99	93+6	96
	FN	93+7	99	89+8	96	97+3	98+1	91+7	96
	EFN	92+8	99	84+16	96	100	99	93+6	96
	I	100	99−1	91+8	96	100	98	93+5	96
	EI	100	99	91+9	96	100	99	91+8	96
	IN	100	99	92+7	96	100	99	94+3	96
	EIN	100	99	93+6	96	100	99	92+7	96
	INR	90+10	99	89+10	96	97+3	99	94+5	96
	EINR	76+24	98.5+1	90+9	96	77+23	99	94+5	96

Table 2: F1-score difference due to the use of ASP in the generation of facts. Positive offsets are improvements (values in percentage).

model; seq2seq has several applications, among them machine translation [Dabre *et al.*, 2020], answering factual questions [Petroni *et al.*, 2019], executing basic arithmetic operations [Hoffmann *et al.*, 2022], text summarization [Zhang *et al.*, 2019], and chatbot functionalities [Liu *et al.*, 2022].

In this work, we employ LLMs in information extraction tasks, and in particular we focus on the extraction of relational facts from text with the aim of addressing subsequent reasoning tasks with formal logic systems. Our idea is motivated by the use of LLMs for similar natural language processing tasks, for example relation categories extraction [Cabot and Navigli, 2021], entity linking [Cao *et al.*, 2021] and semantic role labeling [Blloshmi *et al.*, 2021]. The closest work to our is [Yang *et al.*, 2023], whose focus is *question answering*: an LLM is used to transform a context and a question into atomic facts, which are processed by an ASP solver equipped with background knowledge encoded as ASP rules to derive an answer. The approach of [Yang *et al.*, 2023] uses *few-shot* examples rather than training datasets, which we also explored (in the behavior files whose name contains E). Our work differs from [Yang *et al.*, 2023] mainly in its more structured approach for the prompt engineering phase, distinguishing between portions of the prompts that are of general applicability (behavior file) and portions that are applicable to a specific domain of interest (application file).

Another argument supporting the use of LLMs for data extraction only rather than for reasoning is given by [Nye *et al.*, 2021], suggesting that LLMs are suitable for System-1 thinking: LLMs are designed to predict the next word in a sequence without deep comprehension of crucial reasoning concepts like causality, logic, and probability. As a matter of fact, merging LLMs with logical reasoning into a neurosymbolic framework is an active research field, to which LLMASP can contribute by easing the interface of LLMs and ASP. LLMASP makes no assumption on the origin of the knowledge base in application files. Indeed, we encoded simple knowledge regarding the generation of facts associated with instances in ASP Competitions, which simplified the extraction task for the tested LLMs. In general, we expect knowledge bases to be curated by domain experts, possibly with the help of automated tools.

Regarding the definition of knowledge bases, an interesting approach is given by the NL2ASP tool [Santana *et al.*, 2024], which uses a dual phase architecture to produce ASP programs from natural language descriptions. NL2ASP uses neural machine translation to convert natural language into Controlled Natural Language (CNL) statements, which are then translated into ASP code via the CNL2ASP tool [Caruso *et al.*, 2024]. It is important to highlight that our task is less complex than the one tackled by [Santana *et al.*, 2024] (and for this reason more likely to succeed in the short time). On the other hand, both approaches can benefit from each other: LLMASP can take advantage of NL2ASP to help domain experts in defining background knowledge bases, while NL2ASP can ignore the generation of facts and focus on more complex rules. Yet another possibility is to equip a tool like NL2ASP with rule templates, and instantiate them based on facts extracted with LLMASP, which we will explore in our future work.

Another work aiming at generating ASP programs is presented by [Ishay *et al.*, 2023a]. The main element in their work is a prompt engineering strategy to transform natural language descriptions into ASP incrementally. The proposed pipeline initially identifies the relevant objects and their categories. Subsequently, it forms a predicate that delineates the relationships between objects from various categories. Using these derived data, the pipeline proceeds to build an ASP program following the Generate-Define-Test paradigm.

Finally, we acknowledge an increasing interest in the integration of LLMs and ASP to create powerful AI systems capable of both processing complex natural language inputs and performing sophisticated logical inference [Ishay *et al.*, 2023b; Rajasekharan *et al.*, 2023a; Wang *et al.*, 2024; Yang *et al.*, 2024; Kaur *et al.*, 2025]. Among the applications of such a powerful combination there is the generation of explanations for reasoning tasks [Rajasekharan *et al.*, 2023b; Kalyanpur *et al.*, 2024; Lin *et al.*, 2024]. Within this respect, the idea is to generate justification trees and produce their natural language representations. LLMASP can be used for this second phase with its postprocessing pipeline, which is not discussed in this work.

6 Conclusion

In this paper, we have introduced an approach for combining LLMs and ASP to harness their complementary strengths in natural language processing and logical reasoning. Our prototype system (https://github.com/lewashby/llmasp) is written in Python, and is powered by Ollama, OpenAI API and CLINGO. By providing predefined prompts and enriching specifications with domain-specific knowledge, our approach enables users to tailor the system to diverse problem domains and applications, enhancing its adaptability and versatility. The predefined prompts can be easily modified as they are stored in a separate YAML file (the behavior file). We tested several behavior and application files on a dataset obtained from domains of the ASP Competitions, identifying some key features to drive the generation of facts. Moreover, we showed that ASP can be employed to improve the generation process by representing semantic aspects in the knowledge base, and leaving more syntactic facts to be extracted by LLMs. Future lines of research include exploring different strategies to improve the preprocessing phase of LLMASP (e.g., by introducing calls to LLMs to drive subsequent calls), the evaluation of the postprocessing phase of LLMASP (i.e., the generation of natural language from facts), the use of LLMASP in the context of Fashion Retail, and the integration of LLMASP in LangChain.

Ethical Statement

The use of LLMs inherently exposes to risks related to transparency and accountability. Users of LLMASP must be clearly informed about such risks.

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).

References

[Alviano and Grillo, 2024] Mario Alviano and Lorenzo Grillo. Answer set programming and large language models interaction with YAML: preliminary report. In Emanuele De Angelis and Maurizio Proietti, editors, *CILC 2024, Rome, Italy, June 26-28, 2024*, volume 3733 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024.

[Alviano *et al.*, 2024] Mario Alviano, Fabrizio Lo Scudo, Lorenzo Grillo, and Luis Angel Rodriguez Reiners. Answer set programming and large language models interaction with YAML: second report. In Lucía Gómez Álvarez et al., editor, *KoDis+CAKR+SYNERGY@KR 2024, Hanoi, Vietnam, November 2-8, 2024*, volume 3876 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024.

[Basu *et al.*, 2021] Kinjal Basu, Sarat Chandra Varanasi, Farhad Shakerin, Joaquín Arias, and Gopal Gupta. Knowledge-driven natural language understanding of english text and its applications. In *AAAI 2021, February 2-9, 2021*, pages 12554–12563. AAAI Press, 2021.

[Blloshmi *et al.*, 2021] Rexhina Blloshmi, Simone Conia, Rocco Tripodi, Roberto Navigli, et al. Generating senses and roles: An end-to-end model for dependency-and span-based semantic role labeling. In *IJCAI 2021*, pages 3786–3793, 2021.

[Brown and et al., 2020] Tom B. Brown and et al. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[Cabot and Navigli, 2021] Pere-Lluís Huguet Cabot and Roberto Navigli. Rebel: Relation extraction by end-to-end language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2370–2381, 2021.

[Calimeri *et al.*, 2020] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. *TPLP*, 20(2):294–309, 2020.

[Cao *et al.*, 2021] Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. In *ICLR*. OpenReview.net, 2021.

[Cappanera *et al.*, 2023] Paola Cappanera, Marco Gavanelli, Maddalena Nonato, and Marco Roma. Logic-based benders decomposition in answer set programming for chronic outpatients scheduling. *TPLP*, 23(4):848–864, 2023.

[Cardellini *et al.*, 2024] Matteo Cardellini, Paolo De Nardi, Carmine Dodaro, Giuseppe Galatà, Anna Giardini, Marco Maratea, and Ivan Porro. Solving rehabilitation scheduling problems via a two-phase ASP approach. *TPLP*, 24(2):344–367, 2024.

[Caruso *et al.*, 2024] Simone Caruso, Carmine Dodaro, Marco Maratea, Marco Mochi, and Francesco Riccio. CNL2ASP: converting controlled natural language sentences into ASP. *TPLP*, 24(2):196–226, 2024.

[Chowdhery and et al., 2023] Aakanksha Chowdhery and et al. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023.

[Dabre *et al.*, 2020] Raj Dabre, Chenhui Chu, and Anoop Kunchukuttan. A survey of multilingual neural machine translation. *ACM Computing Surveys (CSUR)*, 53(5):1–38, 2020.

[Gebser *et al.*, 2011] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. *TPLP*, 11(4-5):821–839, 2011.

[Gebser *et al.*, 2020] Martin Gebser, Marco Maratea, and Francesco Ricca. The seventh answer set programming competition: Design and results. *TPLP*, 20(2):176–204, 2020.

[Gelfond and Lifschitz, 1990] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In D. Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh International Conference*, pages 579–597, 1990.

[Hoffmann *et al.*, 2022] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[Ishay *et al.*, 2023a] Adam Ishay, Zhun Yang, and Joohyung Lee. Leveraging large language models to generate answer set programs. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 374–383, 2023.

[Ishay *et al.*, 2023b] Adam Ishay, Zhun Yang, and Joohyung Lee. Leveraging Large Language Models to Generate Answer Set Programs. In *KR 2023*, pages 374–383, 8 2023.

[Jin *et al.*, 2024] Hanlei Jin, Yang Zhang, Dan Meng, Jun Wang, and Jinghua Tan. A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods. *CoRR*, abs/2403.02901, 2024.

[Kalyanpur *et al.*, 2024] Aditya Kalyanpur, Kailash Saravanakumar, Victor Barres, Jennifer Chu-Carroll, David Melville, and David A. Ferrucci. LLM-ARC: enhancing llms with an automated reasoning critic. *CoRR*, abs/2406.17663, 2024.

[Kaur *et al.*, 2025] Navdeep Kaur, Lachlan McPheat, Alessandra Russo, Anthony G Cohn, and Pranava Madhyastha. An empirical study of conformal prediction in llm with asp scaffolds for robust reasoning, 2025.

[Li *et al.*, 2024] Zhiming Li, Yushi Cao, Xiufeng Xu, Junzhe Jiang, Xu Liu, Yon Shin Teo, Shang-Wei Lin, and Yang Liu. Llms for relational reasoning: How far are we? In *LLM4CODE@ICSE*, pages 119–126, 2024.

[Lin *et al.*, 2024] Xinrui Lin, Yangfan Wu, Huanyu Yang, Yu Zhang, Yanyong Zhang, and Jianmin Ji. CLMASP: coupling large language models with answer set programming for robotic task planning. *CoRR*, abs/2406.03367, 2024.

[Liu *et al.*, 2022] Zihan Liu, Mostofa Patwary, Ryan Prenger, Shrimai Prabhumoye, Wei Ping, Mohammad Shoeybi, and Bryan Catanzaro. Multi-stage prompting for knowledgeable dialogue generation. In *ACL 2022*, pages 1317–1337, 2022.

[Marek and Truszczynski, 1999] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, Artificial Intelligence, pages 375–398. Springer, 1999.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.

[Nye *et al.*, 2021] Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M Lake. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34:25192–25204, 2021.

[Petroni *et al.*, 2019] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

[Rajasekharan *et al.*, 2023a] Abhiramon Rajasekharan, Yankai Zeng, and Gopal Gupta. Argument analysis using answer set programming and semantics-guided large language models. In *ICLP Workshops*, 2023.

[Rajasekharan *et al.*, 2023b] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar, and Gopal Gupta. Reliable natural language understanding with large language models and answer set programming. In *ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023*, volume 385 of *EPTCS*, pages 274–287, 2023.

[Santana *et al.*, 2024] Manuel A. Borroto Santana, Irfan Kareem, and Francesco Ricca. Towards automatic composition of ASP programs from natural language specifications. In *IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 6198–6206. ijcai.org, 2024.

[Taupe *et al.*, 2021] Richard Taupe, Gerhard Friedrich, Konstantin Schekotihin, and Antonius Weinzierl. Solving configuration problems with ASP and declarative domain specific heuristics. In Michel Aldanondo, Andreas A. Falkner, Alexander Felfernig, and Martin Stettinger, editors, *Proceedings of the 23rd International Configuration Workshop (CWS/ConfWS 2021), Vienna, Austria, 16-17 September, 2021*, volume 2945 of *CEUR Workshop Proceedings*, pages 13–20. CEUR-WS.org, 2021.

[Touvron and et al., 2023] Hugo Touvron and et al. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[Wang *et al.*, 2024] Zhongsheng Wang, Jiamou Liu, Qiming Bao, Hongfei Rong, and Jingfeng Zhang. Chatlogic: Integrating logic programming with large language models for multi-step reasoning. In *International Joint Conference on Neural Networks, IJCNN 2024, Yokohama, Japan, June 30 - July 5, 2024*, pages 1–8. IEEE, 2024.

[Wotawa, 2020] Franz Wotawa. On the use of answer set programming for model-based diagnosis. In Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki, editors, *IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings*, volume 12144 of *LNCS*, pages 518–529. Springer, 2020.

[Yang *et al.*, 2023] Zhun Yang, Adam Ishay, and Joohyung Lee. Coupling large language models with logic programming for robust and general reasoning from text. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5186–5219, 2023.

[Yang *et al.*, 2024] Xiaocheng Yang, Bingsen Chen, and Yik-Cheung Tam. Arithmetic reasoning with LLM: prolog generation & permutation. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Short Papers, NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 699–710. Association for Computational Linguistics, 2024.

[Zeng *et al.*, 2024] Yankai Zeng, Abhiramon Rajasekharan, Parth Padalkar, Kinjal Basu, Joaquín Arias, and Gopal Gupta. Automated interactive domain-specific conversational agents that understand human dialogs. In Martin Gebser and Ilya Sergey, editors, *PADL 2024, London, UK, January 15-16, 2024, Proceedings*, volume 14512 of *LNCS*, pages 204–222. Springer, 2024.

[Zhang *et al.*, 2019] Haoyu Zhang, Jianjun Xu, and Ji Wang. Pretraining-based natural language generation for text summarization. *arXiv preprint arXiv:1902.09243*, 2019.

[Zhang *et al.*, 2023] Wenxuan Zhang, Xin Li, Yang Deng, Lidong Bing, and Wai Lam. A survey on aspect-based sentiment analysis: Tasks, methods, and challenges. *IEEE Trans. Knowl. Data Eng.*, 35(11):11019–11038, 2023.