



**MARIO ALVIANO**  
DeMaCS@UNICAL



**UNIVERSITÀ  
DELLA  
CALABRIA**

il Campus per eccellenza

# ASP and LLMs interaction with YAML

Preliminary Report

Joint work with  
**Lorenzo Grillo**



**GitHub Repository**

<https://github.com/Xiro28/LLMASP>



**SERICS**  
SECURITY AND RIGHTS IN THE CYBERSPACE



**SISTEMA INTEGRATO  
DI LABORATORI  
PER L'AMBIENTE**



## Large Language Models

Computational models achieving **general-purpose language generation** and other natural language processing tasks.

## Large Language Models

Computational models achieving **general-purpose language generation** and other natural language processing tasks.

We have General AI here!



## Large Language Models

Computational models achieving **general-purpose language generation** and other natural language processing tasks.

We have General AI here!



False! It is good at NLP  
Logical reasoning is still our subject

## Answer Set Programming

KRR language, good for combinatorial search and optimization

KRR language, good for combinatorial search and optimization

```
% guess one color for each node  
{assign(X,C) : color(C)} = 1 :- node(X).  
  
% adjacent nodes must have different colors  
:- edge(X,Y), assign(X,C), assign(Y,C).  
  
% minimize the number of used colors  
:- assign(_,C). [1@1, C]
```

## Answer Set Programming

KRR language, good for combinatorial search and optimization

```
% guess one color for each node  
{assign(X,C) : color(C)} = 1 :- node(X).  
  
% adjacent nodes must have different colors  
:- edge(X,Y), assign(X,C), assign(Y,C).  
  
% minimize the number of used colors  
:- assign(_,C). [1@1, C]
```

Possibly (read it certainly)  
there is a (huge) barrier to entry

# Computer Programming





## Computer Programming



People chatting and telling around they are programming

ChatGPT

**Input**

Andrew is free from 11 am to 3 pm, Joanne is free from noon to 2 pm and then 3:30 pm to 5 pm. Hannah is available at noon for half an hour, and then 4 pm to 6 pm. What are some options for start times for a 30 minute meeting for Andrew, Hannah, and Joanne?

**Output**

Here is a list of their availability;

- Andrew: 11 am to 3 pm
- Joanne: noon to 2 pm, and 3:30 pm to 5 pm
- Hannah: noon to 12:30 pm, and 4 pm to 6 pm

Based on their availability, there is a 30-minute window where all three of them are available, which is from 4 pm to 4:30 pm. So, the meeting can be scheduled at 4 pm.

## Computer Programming



People chatting and telling around they are programming

ChatGPT

Input

Andrew is free from 11 am to 3 pm, Joanne is free from noon to 2 pm and then 3:30 pm to 5 pm. Hannah is available at noon for half an hour, and then 4 pm to 6 pm. What are some options for start times for a 30 minute meeting for Andrew, Hannah, and Joanne?

Output

Here is a list of their availability;

- Andrew: 11 am to 3 pm
- Joanne: noon to 2 pm, and 3:30 pm to 5 pm
- Hannah: noon to 12:30 pm, and 4 pm to 6 pm

Based on their availability, there is a 30-minute window where all three of them are available, which is from 4 pm to 4:30 pm. So, the meeting can be scheduled at 4 pm.

As we are not a church... **let's take the best of both worlds!**

## Idea Overview

Start with natural language

## Idea Overview

Start with natural language



Obtain a relational  
representation (facts)

## Idea Overview

Start with natural language



Obtain a relational  
representation (facts)



Combine facts with a KB  
(an ASP program)

## Idea Overview

Start with natural language



Obtain a relational  
representation (facts)



Combine facts with a KB  
(an ASP program)



Map the answer set to  
natural language

## Idea Overview

Start with natural language



Obtain a relational  
representation (facts)



Combine facts with a KB  
(an ASP program)



Map the answer set to  
natural language

**Users will not see ASP**  
(not that before we were showing them ASP)

## LLMs themselves

A simple PROMPT asking for facts

ChatGPT



Encode as Datalog facts the following sentences:  
Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.



Sure! Here are the sentences encoded as Datalog facts:

datalog

Copia codice

```
% Facts about the desire to go eat pizza  
wants_to_eat_pizza(tonight, I).
```

```
% Facts about the people involved  
going_with(I, Marco, tonight).  
going_with(I, Alessio, tonight).
```

```
% Facts about pizza preferences  
likes_pizza_with_topping(Marco, onions).
```

Explanation of the facts:

- `wants_to_eat_pizza(tonight, I).` expresses that "I" want to eat pizza tonight.
- `going_with(I, Marco, tonight).` and `going_with(I, Alessio, tonight).` express that "I" will go with Marco and Alessio tonight.
- `likes_pizza_with_topping(Marco, onions).` expresses that Marco likes pizza with onions as toppings.





## LLMs themselves

A simple PROMPT asking for facts

We need **more control** on these facts!

ChatGPT



Encode as Datalog facts the following sentences:  
Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.



Sure! Here are the sentences encoded as Datalog facts:

datalog

Copia codice

```
% Facts about the desire to go eat pizza  
wants_to_eat_pizza(tonight, I).
```

```
% Facts about the people involved  
going_with(I, Marco, tonight).  
going_with(I, Alessio, tonight).
```

```
% Facts about pizza preferences  
likes_pizza_with_topping(Marco, onions).
```

Explanation of the facts:

- `wants_to_eat_pizza(tonight, I).` expresses that "I" want to eat pizza tonight.
- `going_with(I, Marco, tonight).` and `going_with(I, Alessio, tonight).` express that "I" will go with Marco and Alessio tonight.
- `likes_pizza_with_topping(Marco, onions).` expresses that Marco likes pizza with onions as toppings.



# LLMs themselves

A simple PROMPT asking for facts

We need **more control** on these facts!

And we don't need this garbage

ChatGPT



Encode as Datalog facts the following sentences:  
Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.



Sure! Here are the sentences encoded as Datalog facts:

datalog

Copia codice

```
% Facts about the desire to go eat pizza  
wants_to_eat_pizza(tonight, I).
```

```
% Facts about the people involved  
going_with(I, Marco, tonight).  
going_with(I, Alessio, tonight).
```

```
% Facts about pizza preferences  
likes_pizza_with_topping(Marco, onions).
```

Explanation of the facts:

- `wants_to_eat_pizza(tonight, I).` expresses that "I" want to eat pizza tonight.
- `going_with(I, Marco, tonight).` and `going_with(I, Alessio, tonight).` express that "I" will go with Marco and Alessio tonight.
- `likes_pizza_with_topping(Marco, onions).` expresses that Marco likes pizza with onions as toppings.



## Prompt Engineering

Chat with the LLM...  
give instructions  
on the format and  
task you want to achieve

ChatGPT ▾



You are a Natural Language to Datalog translator. To translate your input to Datalog, you will be asked a sequence of questions. The answers are inside the user input provided with [USER\_INPUT]input[/USER\_INPUT] and the format is provided with [ANSWER\_FORMAT]predicate(terms).[/ANSWER\_FORMAT]. Predicate is a lowercase string (possibly including underscores). Terms is a comma-separated list of either double quoted strings or integers. Be sure to control the number of terms in each answer! An answer MUST NOT be answered if it is not present in the user input. Remember these instructions and don't say anything!



Understood. Please provide the input for translation.

## Prompt Engineering

Chat with the LLM...  
give instructions  
on the format and  
task you want to achieve

We can start with general  
instructions for our task

ChatGPT ▾



You are a Natural Language to Datalog translator. To translate your input to Datalog, you will be asked a sequence of questions. The answers are inside the user input provided with [USER\_INPUT]input[/USER\_INPUT] and the format is provided with [ANSWER\_FORMAT]predicate(terms).[/ANSWER\_FORMAT]. Predicate is a lowercase string (possibly including underscores). Terms is a comma-separated list of either double quoted strings or integers. Be sure to control the number of terms in each answer! An answer MUST NOT be answered if it is not present in the user input. Remember these instructions and don't say anything!



Understood. Please provide the input for translation.



Here is some context that you MUST analyze and remember.

The user provides a request to obtain catering suggestions. The user can mention a day, other persons, and their cuisine preferences. Remember this context and don't say anything!

Something specific of the application we are working on



Understood. Please provide the user input for translation.



Here is some context that you MUST analyze and remember.

The user provides a request to obtain catering suggestions. The user can mention a day, other persons, and their cuisine preferences. Remember this context and don't say anything!

Something specific of the application we are working on



Understood. Please provide the user input for translation.

And we can start extracting facts!



[USER\_INPUT]Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.

[/USER\_INPUT]

List all the persons mentioned including me if indirectly included.

[ANSWER\_FORMAT]person("who").

[/ANSWER\_FORMAT]



person("I").

person("Marco").

person("Alessio").



[USER\_INPUT]Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.

[/USER\_INPUT]

For each person, list any restaurant preferences.

[ANSWER\_FORMAT]cuisine\_preferences("who", "country").[/ANSWER\_FORMAT]



cuisine\_preferences("Marco", "pizza").

cuisine\_preferences("Alessio", "pizza").

cuisine\_preferences("I", "pizza").



[USER\_INPUT]Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.

[/USER\_INPUT]

For each person, list any restaurant preferences.

[ANSWER\_FORMAT]cuisine\_preferences("who", "country").[/ANSWER\_FORMAT]



cuisine\_preferences("Marco", "pizza").

cuisine\_preferences("Alessio", "pizza").

cuisine\_preferences("I", "pizza").

[USER\_INPUT]Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.

[/USER\_INPUT]

For each person, list what they want to eat.

[ANSWER\_FORMAT]want\_food("who", "what").

[/ANSWER\_FORMAT]



want\_food("I", "pizza").

want\_food("Marco", "pizza with onions").

want\_food("Alessio", "pizza").





## Reasoning with ASP

```
person("I").  
person("Marco").  
person("Alessio").  
  
cuisine_preferences("Marco", "pizza").  
cuisine_preferences("Alessio", "pizza").  
cuisine_preferences("I", "pizza").  
  
want_food("I", "pizza").  
want_food("Marco", "pizza with onions").  
want_food("Alessio", "pizza").
```

Extracted facts

## Reasoning with ASP

```
person("I").
person("Marco").
person("Alessio").

cuisine_preferences("Marco", "pizza").
cuisine_preferences("Alessio", "pizza").
cuisine_preferences("I", "pizza").

want_food("I", "pizza").
want_food("Marco", "pizza with onions").
want_food("Alessio", "pizza").
```

Extracted facts

Combine facts  
with a KB

```
can_go_together(X,Y,Z) :-
    person(X), person(Y), X < Y,
    want_food(X,Z), want_food(Y,Z).

can_go_together(X,Y,Z) :-
    person(X), person(Y), X < Y,
    cuisine_preferences(X,Z), cuisine_preferences(Y,Z).

#show can_go_together/3.
```

## Reasoning with ASP

```
person("I").
person("Marco").
person("Alessio").

cuisine_preferences("Marco", "pizza").
cuisine_preferences("Alessio", "pizza").
cuisine_preferences("I", "pizza").

want_food("I", "pizza").
want_food("Marco", "pizza with onions").
want_food("Alessio", "pizza").
```

Extracted facts

Combine facts  
with a KB

```
can_go_together(X,Y,Z) :-
    person(X), person(Y), X < Y,
    want_food(X,Z), want_food(Y,Z).

can_go_together(X,Y,Z) :-
    person(X), person(Y), X < Y,
    cuisine_preferences(X,Z), cuisine_preferences(Y,Z).

#show can_go_together/3.
```

Obtain an  
answer set

```
can_go_together("Alessio", "Marco", "pizza").
can_go_together("I", "Marco", "pizza").
can_go_together("Alessio", "I", "pizza").
```

## Back to Natural Language

General instructions  
on the task

You are now a Datalog to Natural Language translator.  
You will be given relational facts and mapping instructions.  
Relational facts are given in the form  
[FACTS]atoms[/FACTS].  
Remember these instructions and don't say anything!



Understood. Please provide the relational facts and mapping instructions.

## Back to Natural Language

General instructions  
on the task

You are now a Datalog to Natural Language translator.  
You will be given relational facts and mapping instructions.  
Relational facts are given in the form  
[FACTS]atoms[/FACTS].  
Remember these instructions and don't say anything!



Understood. Please provide the relational facts and mapping instructions.

Something specific of the  
application we are working  
on

Here is some context that you **MUST** analyze and remember.  
Explain the facts in a human readable way, as a paragraph.  
Remember this context and don't say anything!



Understood. Please provide the relational facts for translation.

Provide facts and mapping instructions

```
[FACTS]can_go_together("me", "marco", "pizza").  
can_go_together("me", "alessio", "pizza").  
can_go_together("marco", "alessio", "pizza"). [/FACTS]  
Each fact matching can_go_together("person 1",  
"person 2", "cuisine preference") must be  
interpreted as follows: Say that "person 1" can go  
with "person 2" to eat "cuisine preference".
```

Collect answers



I can go with Marco to eat pizza. I can go with Alessio to eat pizza. Marco can go with Alessio to eat pizza.

Ask to summarize the collected responses



Summarize the following responses:  
I can go with Marco to eat pizza. I can go with Alessio to eat pizza. Marco can go with Alessio to eat pizza.



The responses indicate that the speaker (presumably "me") can accompany Marco and Alessio to eat pizza, and Marco can also accompany Alessio to eat pizza.

This is the output shown to the user

## **Let's Automate IT!**

YAML (yet another markup language)

Like JSON... but more readable



## Let's Automate IT!

YAML (yet another markup language)

Like JSON... but more readable

```
name: Lorenzo
degrees:
- Bachelor
short bio: |
  I'm Lorenzo...
  I'm a student at UNICAL...
```

Here is an object with properties  
**name**, **degrees** and **short bio**

**degrees** is a list (of values)

**short bio** is a string (block notation)

## Let's Automate IT!

YAML (yet another markup language)

Like JSON... but more readable

```
name: Lorenzo
degrees:
- Bachelor
short bio: |
  I'm Lorenzo...
  I'm a student at UNICAL...
```

Here is an object with properties  
**name**, **degrees** and **short bio**

**degrees** is a list (of values)

**short bio** is a string (block notation)

YAML is used in many applications,  
among them Kubernetes

We use YAML to define an application

### preprocessing:

- \_: The user provides a request to obtain catering suggestions.  
The user can mention a day, other persons, and their cuisine preferences.
- person("who"): List all the persons mentioned including me if indirectly included.
- cuisine\_preferences("who", "country"): For each person, list any restaurant preferences.
- want\_food("who", "what"): For each person, list what they want to eat.

### knowledge base: |

```
can_go_together(X,Y,Z) :-  
    person(X), person(Y), X < Y,  
    want_food(X,Z), want_food(Y,Z).
```

```
can_go_together(X,Y,Z) :-  
    person(X), person(Y), X < Y,  
    cuisine_preferences(X,Z), cuisine_preferences(Y,Z).
```

```
#show can_go_together/3.
```

### postprocessing:

- \_: Explain the facts in a human readable way, as a paragraph.
- can\_go\_together("person 1", "person 2", "cuisine preference"): |  
Say that "person 1" can go with "person 2" to eat "cuisine preference".

We use YAML to define an application

#### preprocessing:

- \_: The user provides a request to obtain catering suggestions.  
The user can mention a day, other persons, and their cuisine preferences.
- person("who"): List all the persons mentioned including me if indirectly included.
- cuisine\_preferences("who", "country"): For each person, list any restaurant preferences.
- want\_food("who", "what"): For each person, list what they want to eat.

#### knowledge base: |

```
can_go_together(X,Y,Z) :-  
    person(X), person(Y), X < Y,  
    want_food(X,Z), want_food(Y,Z).
```

```
can_go_together(X,Y,Z) :-  
    person(X), person(Y), X < Y,  
    cuisine_preferences(X,Z), cuisine_preferences(Y,Z).
```

```
#show can_go_together/3.
```

#### postprocessing:

- \_: Explain the facts in a human readable way, as a paragraph.
- can\_go\_together("person 1", "person 2", "cuisine preference"): |  
Say that "person 1" can go with "person 2" to eat "cuisine preference".

Preprocessing includes the context and the instructions on the atoms to extract from the input

We use YAML to define an application

#### preprocessing:

- \_: The user provides a request to obtain catering suggestions.  
The user can mention a day, other persons, and their cuisine preferences.
- person("who"): List all the persons mentioned including me if indirectly included.
- cuisine\_preferences("who", "country"): For each person, list any restaurant preferences.
- want\_food("who", "what"): For each person, list what they want to eat.

#### knowledge base: |

```
can_go_together(X,Y,Z) :-  
    person(X), person(Y), X < Y,  
    want_food(X,Z), want_food(Y,Z).
```

```
can_go_together(X,Y,Z) :-  
    person(X), person(Y), X < Y,  
    cuisine_preferences(X,Z), cuisine_preferences(Y,Z).
```

```
#show can_go_together/3.
```

#### postprocessing:

- \_: Explain the facts in a human readable way, as a paragraph.
- can\_go\_together("person 1", "person 2", "cuisine preference"): |  
Say that "person 1" can go with "person 2" to eat "cuisine preference".

Preprocessing includes the context and the instructions on the atoms to extract from the input

KB is simply an ASP program

We use YAML to define an application

#### preprocessing:

- \_: The user provides a request to obtain catering suggestions.  
The user can mention a day, other persons, and their cuisine preferences.
- person("who"): List all the persons mentioned including me if indirectly included.
- cuisine\_preferences("who", "country"): For each person, list any restaurant preferences.
- want\_food("who", "what"): For each person, list what they want to eat.

#### knowledge base: |

```
can_go_together(X,Y,Z) :-  
  person(X), person(Y), X < Y,  
  want_food(X,Z), want_food(Y,Z).
```

```
can_go_together(X,Y,Z) :-  
  person(X), person(Y), X < Y,  
  cuisine_preferences(X,Z), cuisine_preferences(Y,Z).
```

```
#show can_go_together/3.
```

#### postprocessing:

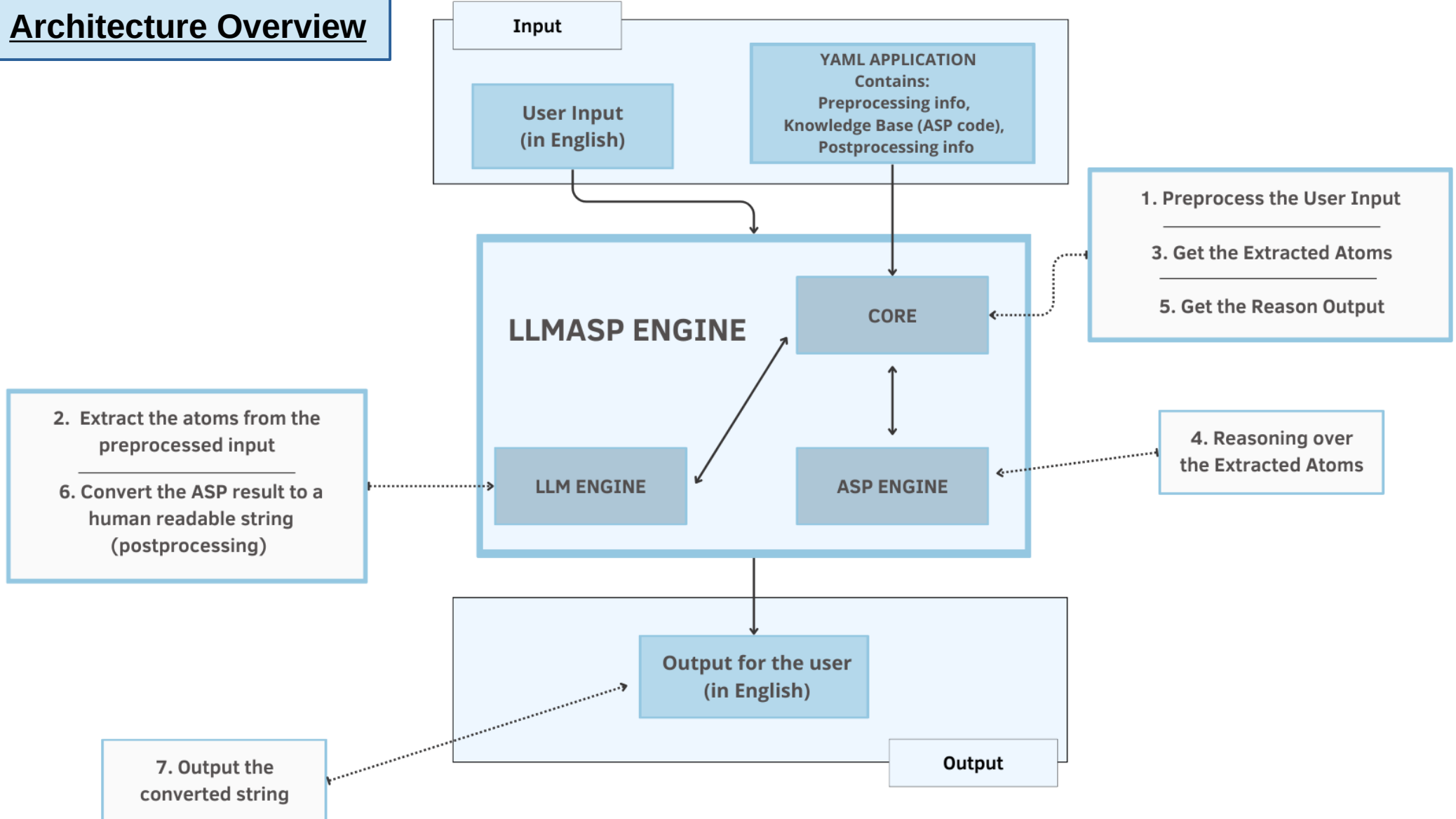
- \_: Explain the facts in a human readable way, as a paragraph.
- can\_go\_together("person 1", "person 2", "cuisine preference"): |  
Say that "person 1" can go with "person 2" to eat "cuisine preference".

Preprocessing includes the context and the instructions on the atoms to extract from the input

KB is simply an ASP program

Postprocessing includes the context and the mapping instructions

# Architecture Overview



## Summing UP

LLMASP combines  
NLP capabilities of LLMs  
with the deep reasoning of ASP



## Summing UP

LLMASP combines  
NLP capabilities of LLMs  
with the deep reasoning of ASP

We automated the interaction  
with LLM and ASP engines

## Summing UP

LLMASP combines  
NLP capabilities of LLMs  
with the deep reasoning of ASP

We automated the interaction  
with LLM and ASP engines

### Still a lot to do!

**Why those prompts and not something different?**

**We have to build a dataset.**

**We have to try different prompts and measure their quality.**

**We can start with the preprocessing and then evaluate the postprocessing by extracting facts from the produced output.**

# Questions

