Bridging Natural Language Understanding, Symbolic Planning, and Robotic Execution through Hybrid AI Systems

DE LA RECHERCHE À L'INDUSTRIE

COMPAI – 2025

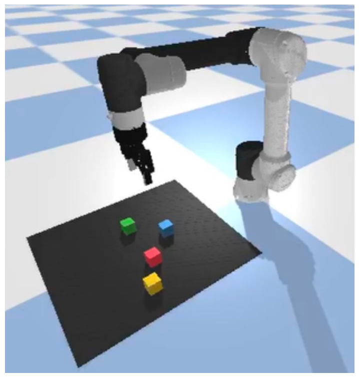
18 August 2025

Luis Palacios Medinacelli, Matteo Morelli, Gaël de Chalendar, Mykola Liashuha, Jaonary Rabarisoa, Lucas Labarussiat and Raphaël Lallement

Outline

Intro : RobotArm Scenario and Challenges

"Magicoders" : A project targeting multi-modal LLMs and Generative AI for Robotics Systems Lifecycle.



One of the challenges involves **planning and reasoning using LLMs**.

The scenario involves a **robot arm** and a set of **objects** (cubes) , where the **user** expresses his intention on what should be changed in the scene (**goal**), and this intention is down-streamed to the **actuators** on the robot.

Multidisciplinary challenge:

- Scene interpretation (aided by VLMs)
- Natural Language interpretation (LLMs)
- Planning (LLMs and/or PDDL) => *the article focuses on this stage*
- Assurances, guarantees, verification (Symbolic AI – PDDL)
- RobotArm execution (ROS and Behavior Trees + skills)

LLMs and VLMs

Generative AI (GenAI), has seen exceptional advancement and growth since ChatGPT was publicly released in 2022.

This technology, relies on the capabilities of ANN to learn and generalize from data samples, and enormous amount of data feed to these networks.

A notorious feature of this technology are the models, which **encode** (among others) **common sense knowledge**.

Although the majority of these models, known as **LLMs**, are known as being used for **text2text** generation, theoretically they can receive any type of input and provide any type of output data, like images, sound, commands, etc. This ”usage” ***beyond*** text is known as multimodal generative AI.

Some types of models form GenAI:

- **LLM** – **Large Language Model:** A type of AI model trained on massive text datasets to understand and generate human-like language (e.g., ChatGPT).
- **VLM** – **Vision-Language Model:** An AI model that combines visual (image/video) and textual data to understand and generate responses based on both modalities (e.g., interpreting images with text).
- **VLA : Vision-Language-Action model:** A class of multimodal foundation models that integrates vision, language and actions.

Planning Domain Definition Language (PDDL)

PDDL is a formal, logic-based language, designed to represent planning problems with well defined:

- Predicates
- Objects
- Actions
- Preconditions and Effects

Given a PDDL domain, intuitively, a **PDDL problem** describes an **initial state** and a desired **final state** (goal).

Then **solving the problem** consist on finding **a sequence of applicable actions** (defined in the PDDL domain) which transform **the initial state into the final state**.

If a solution is found, the sequence of actions is called a **PLAN**.

Considerations:

- There could be many plans.
- There exist very efficient solvers for PDDL and other planning formalisms.
- Once a plan is found there is a **guarantee** that the goal can be achieved.
- Encodes expert knowledge and a specific way to modify each state (defined by the actions and the effects).

PDDL - example

We have chosen a classic a PDDL domain from the 2nd International Planning Competition (2000)
The domain, called ”blocks world” includes the main actions and predicates for a pick-and-place scenario.

```
(define (domain blocksworld)
(:requirements :strips) ;; maybe not necessary
(:predicates (on ?x  ?y )
            (ontable ?x )
            (clear ?x)
            (handempty)
            (holding ?x)
            )
(:action pick-up
            :parameters (?x)
            :precondition (and (clear ?x) (ontable ?x)
(handempty))
            :effect
            (and (not (ontable ?x))
                 (not (clear ?x))
                 (not (handempty))
                 (holding ?x)))

  (:action put-down
            :parameters (?x)
            :precondition (holding ?x)
            :effect
            (and (not (holding ?x))
                 (clear ?x)
                 (handempty)
                 (ontable ?x)))
```

https://github.com/potassco/pddl-instances/blob/master/ipc-2000/domains/blocks-strips-typed/domain.pddl

Objects: The domain considers only 1 type of object: block

Predicates: It considers

- a binary predicate: (on ?x - block ?y - block),
- three unary prediactes: (ontable ?x - block), (clear?x - block), (holding ?x - block)
- a 0-arity predicate, to establish that the robot arm’s is free: (handempty)

Actions: Four actions, along with their preconditions and effects to manipulate the blocks:

- pick-up
- put-down
- stack

```
(define (problem instance-10)
(:domain blocksworld)
(:objects A B C D E F G )
(:INIT ( clear E ) ( ontable D ) ( on E G ) ( on G B ) ( on B A )
( on A F ) ( on F C ) ( on C D ) ( handempty ) )
(:goal (and ( on A G ) ( on G D ) ( on D B ) ( on B C ) ( on C F )
( on F E ) ) )
)
```

Hybrid Approach

LLMs

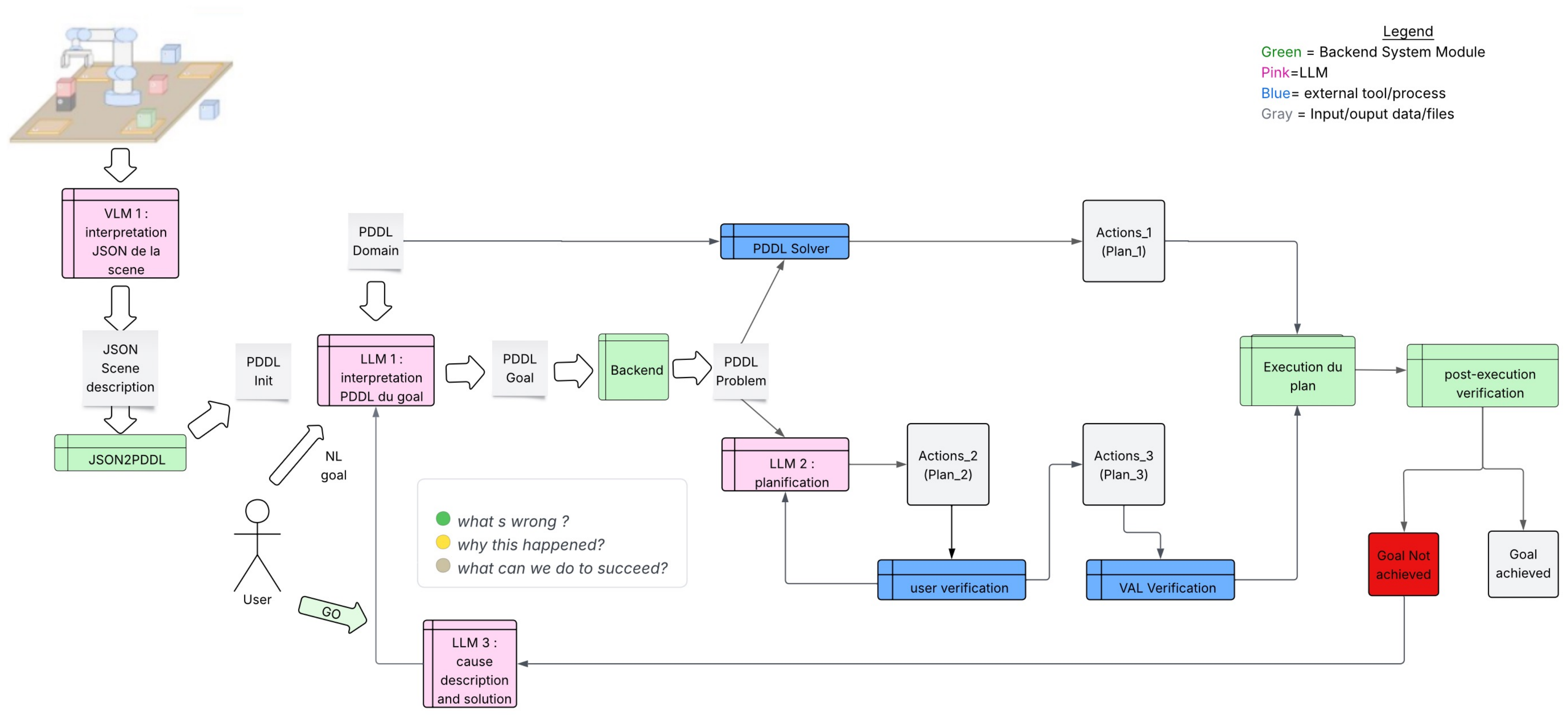
- Encodes commonsense knowledge.
- Interpretations and code generation.
- No guarantees, hallucinations, deviations, decontextualization.
- Lacks control : explainability, traceability.
- Reasoning "limits" and reach – a declination of this problem is planning.

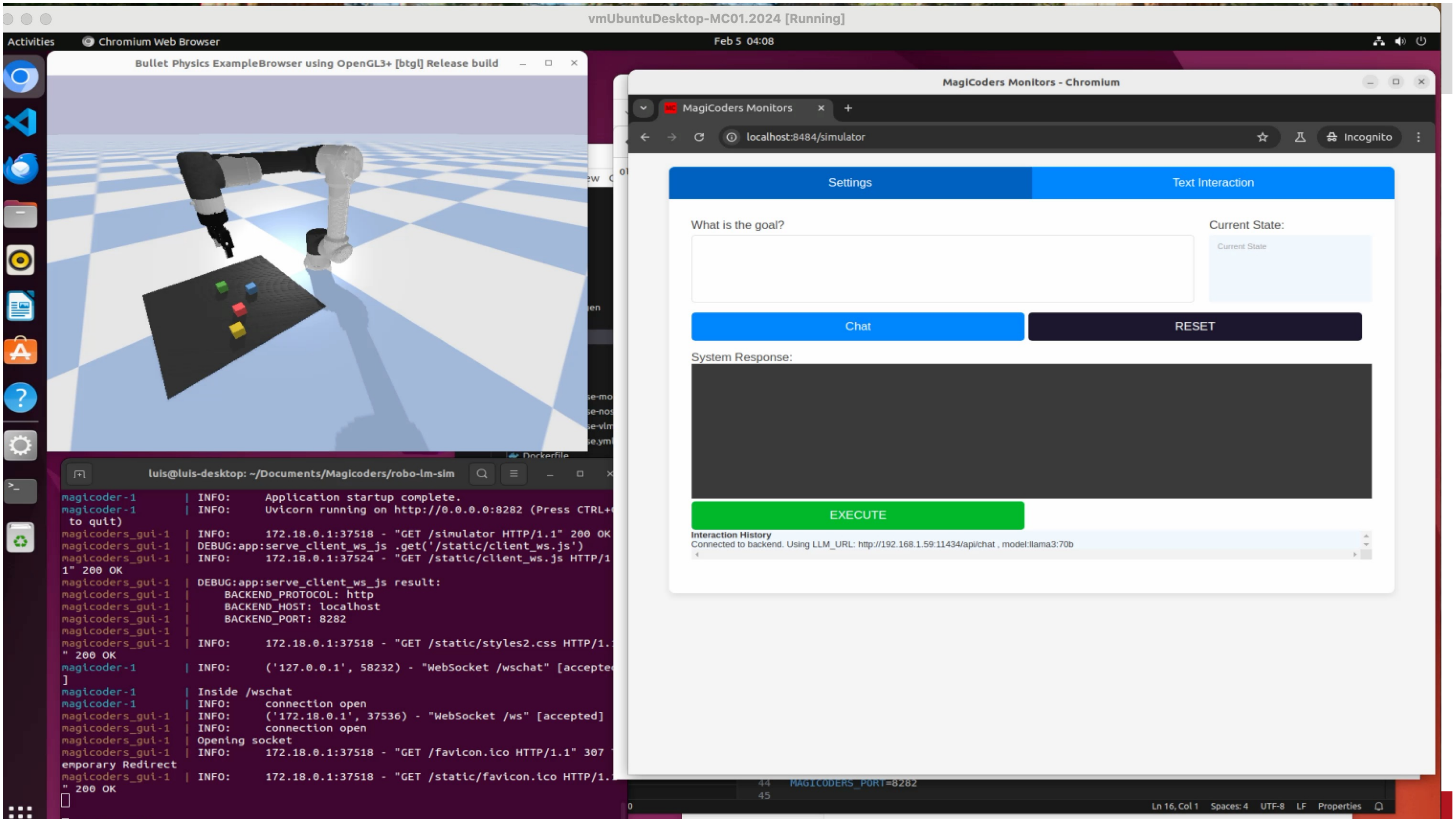
	Feature	GenAI	Symbolic AI
+	Interpretation	★	
+	Translation	★	
+	Code Generation	★	
−	Hallucinations	★	
−	Strictness, Rigidness		★
+	Proovability		★
+	Explainability, Traceability		★
−	Complex, Requires High Expertise		★

PDDL

- Encodes formal knowledge.
- Transparent, guarantees, provability, highly optimized.
- Is complex and requires expertise.
- Guaranteed plans.
- Unambiguity and Explainability: The description of the domain, the scene, the objects and their interactions are well understood and traceable.
- Reusability: Symbolic knowledge, domains, plans, etc. are easily transferable to other systems concerned by the same or similar domains.

Approach Workflow



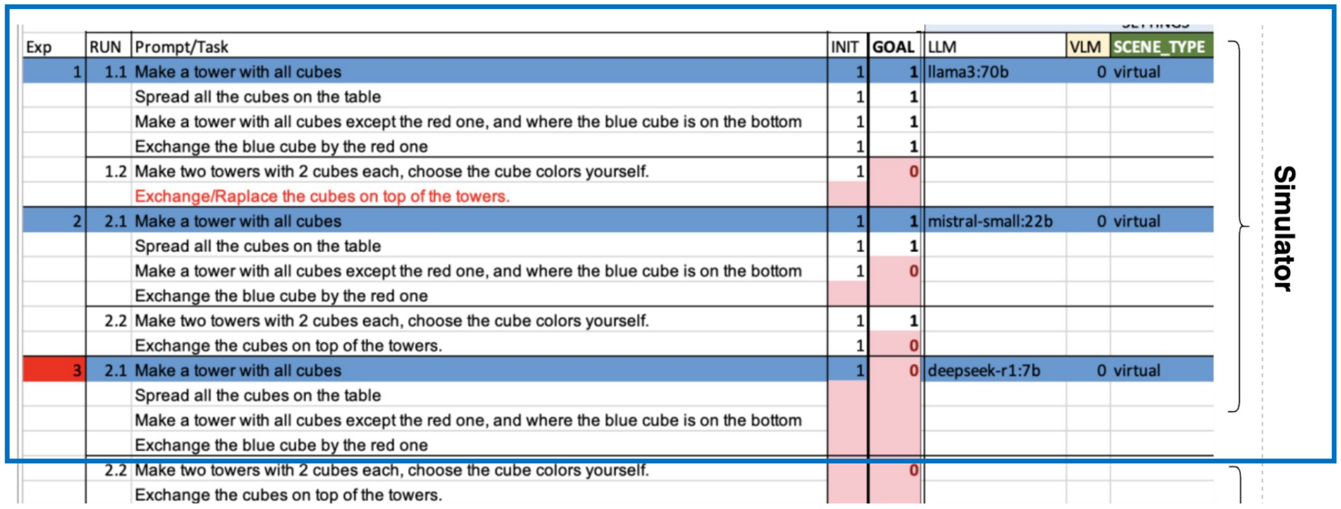


Interaction History
Connected to backend. Using LLM_URL: http://192.168.1.59:11434/api/chat , model:llama3:70b

```
magicoder-1         | INFO:     Application startup complete.
magicoder-1         | INFO:     Uvicorn running on http://0.0.0.0:8282 (Press CTRL+
 to quit)
magicoders_gui-1    | INFO:     172.18.0.1:37518 - "GET /simulator HTTP/1.1" 200 OK
magicoders_gui-1    | DEBUG:app:serve_client_ws_js .get('/static/client_ws.js')
magicoders_gui-1    | INFO:     172.18.0.1:37524 - "GET /static/client_ws.js HTTP/1
1" 200 OK
magicoders_gui-1    | DEBUG:app:serve_client_ws_js result:
magicoders_gui-1    |     BACKEND_PROTOCOL: http
magicoders_gui-1    |     BACKEND_HOST: localhost
magicoders_gui-1    |     BACKEND_PORT: 8282
magicoders_gui-1    |
magicoders_gui-1    | INFO:     172.18.0.1:37518 - "GET /static/styles2.css HTTP/1.
" 200 OK
magicoder-1         | INFO:     ('127.0.0.1', 58232) - "WebSocket /wschat" [accepte
]
magicoder-1         | Inside /wschat
magicoder-1         | INFO:     connection open
magicoders_gui-1    | INFO:     ('172.18.0.1', 37536) - "WebSocket /ws" [accepted]
magicoders_gui-1    | INFO:     connection open
magicoders_gui-1    | Opening socket
magicoders_gui-1    | INFO:     172.18.0.1:37518 - "GET /favicon.ico HTTP/1.1" 307
emporary Redirect
magicoders_gui-1    | INFO:     172.18.0.1:37518 - "GET /static/favicon.ico HTTP/1.
" 200 OK
```

```
44    MAGICODERS_PORT=8282
45
```

Experiments



Exp	RUN	Prompt/Task	INIT	GOAL	LLM	VLM	SCENE_TYPE
1	1.1	Make a tower with all cubes	1	1	llama3:70b	0	virtual
		Spread all the cubes on the table	1	1			
		Make a tower with all cubes except the red one, and where the blue cube is on the bottom	1	1			
		Exchange the blue cube by the red one	1	1			
	1.2	Make two towers with 2 cubes each, choose the cube colors yourself.	1	0			
		Exchange/Raplace the cubes on top of the towers.					
2	2.1	Make a tower with all cubes	1	1	mistral-small:22b	0	virtual
		Spread all the cubes on the table	1	1			
		Make a tower with all cubes except the red one, and where the blue cube is on the bottom	1	0			
		Exchange the blue cube by the red one					
	2.2	Make two towers with 2 cubes each, choose the cube colors yourself.	1	1			
		Exchange the cubes on top of the towers.	1	0			
3	2.1	Make a tower with all cubes	1	0	deepseek-r1:7b	0	virtual
		Spread all the cubes on the table					
		Make a tower with all cubes except the red one, and where the blue cube is on the bottom					
		Exchange the blue cube by the red one					
	2.2	Make two towers with 2 cubes each, choose the cube colors yourself.		0			
		Exchange the cubes on top of the towers.					

Simulator

Success or failure in our case is given by

1) the appropriate interpretation of the scene and the **goal**, using generative AI, and by

2) the **correctness**, **feasibility** and **executability** of the produced plan.

Notes:

- ➢ The results take a zero-shot, out-of-the-box setting.
- ➢ There can be improvements by prompt engineering and finetuning, but this remains further work.
- ➢ The approach takes a "plug & play" concept.
- ➢ The models were selected by their popularity and free availability.
- ➢ The experiments evidence the feasibility, apparent difficulties/limits and steer the directions for further work.

Current and Further Work

Current work:

- ✓ Verification, Recovery and Mitigation Strategies:
    - ✓ Automatically verify the generated plan (if made by an LLM).
    - ✓ Verify the execution of the plan (success/failure):
        - ✓ What is wrong?
        - ✓ What happened? (explanation)
        - ✓ What can be done to succeed?
    - ✓ This opens complex planning, as a sequence of simpler goals.
- ✓ Testing highly ranked code generation models.
- ✓ Complexification of the scene.

Further work:

- ➢ Modification and enrichment of the PDDL domain (LLM assisted).
    - The execution layer has to be in sync with the actions defined in the PDDL domain to ensure the coherence and executability of the results.
- ➢ Training and Finetuning.

Thanks

luis.palacios@cea.fr