L2P: A Python Toolkit for Automated PDDL Model Generation with Large Language Models

Marcus Tantakoun[1], Christian Muise[2], Xiaodan Zhu[3]
School of Computing, Queen's University, Kingston, Canada

Mu Lab at Queen's University
INGENUITY LABS RESEARCH INSTITUTE at Queen's University
Queen's University

"Fake Reasoning"

LLMs Can't Plan Papers

"Deep learning for system 2 processing."

"Chain of thoughtlessness: An analysis of cot in planning"

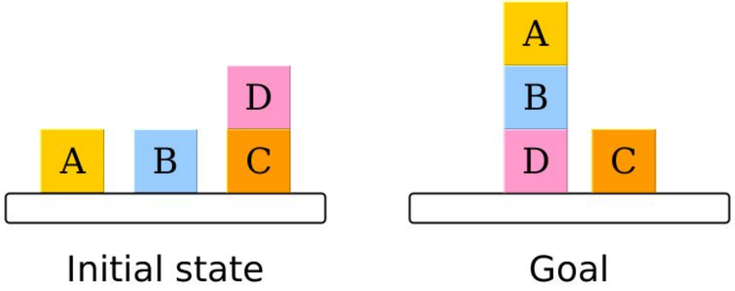
"Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench."

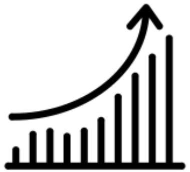
"Evaluating cognitive maps and planning in large language models with cogeval."

"Understanding the capabilities of large language models for automated planning."

And the list goes on …

I've seen this before!



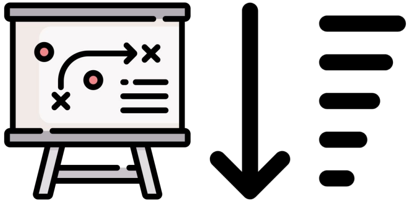


I'll try my best from what I've been trained on corpus!

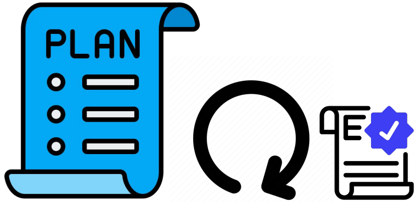

LLM
Excel in highly-structured generative solutions from natural language descriptions
AI Planning
Promising alternative to generate/validate robust, optimal plans through logical and computational methods

Broadly, LLM+AP can be categorized as:

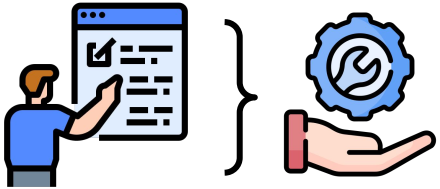
(i) ***LLMs-as-Heuristics***: enhance search efficiency by providing heuristic guidance (i.e. creating successor / goal function)



(ii) ***LLMs-as-Planners***: proposing plans that are later refined through post-hoc implementations (i.e. external validators)



(iii) ***LLMs-as-Modelers***: LLMs generate planning models → external symbolic solvers to produce plans (**OUR FOCUS**)



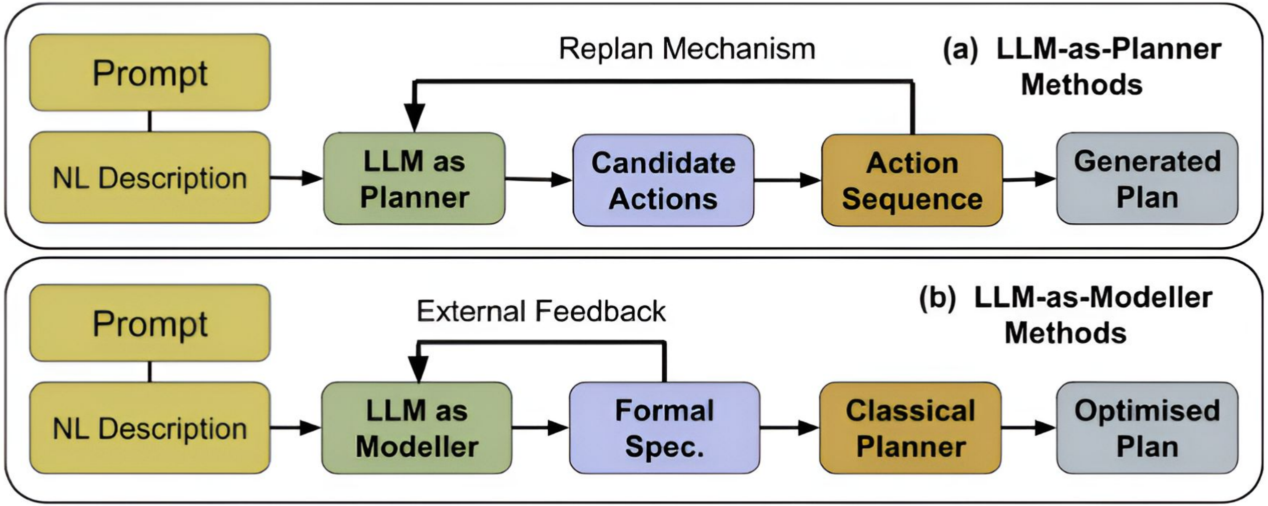
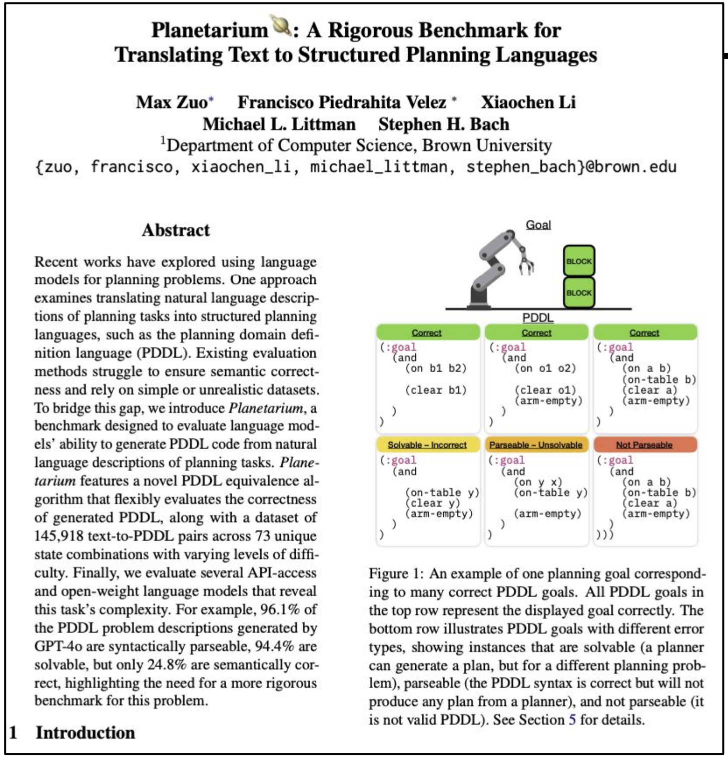


Figure 1: Distinction of planning using LLMs: (a) LLM-as-Planner uses LLMs for direct planning; (b) LLM-as-Modeller uses LLMs to generate AP specifications for existing task planning methods such as PDDL.

Emerging new field …



Planetarium: A Rigorous Benchmark for Translating Text to Structured Planning Languages

Max Zuo* Francisco Piedrahita Velez * Xiaochen Li
Michael L. Littman Stephen H. Bach
[1]Department of Computer Science, Brown University
{zuo, francisco, xiaochen_li, michael_littman, stephen_bach}@brown.edu

Abstract

Recent works have explored using language models for planning problems. One approach examines translating natural language descriptions of planning tasks into structured planning languages, such as the planning domain definition language (PDDL). Existing evaluation methods struggle to ensure semantic correctness and rely on simple or unrealistic datasets. To bridge this gap, we introduce *Planetarium*, a benchmark designed to evaluate language models' ability to generate PDDL code from natural language descriptions of planning tasks. *Planetarium* features a novel PDDL equivalence algorithm that flexibly evaluates the correctness of generated PDDL, along with a dataset of 145,918 text-to-PDDL pairs across 73 unique state combinations with varying levels of difficulty. Finally, we evaluate several API-access and open-weight language models that reveal this task's complexity. For example, 96.1% of the PDDL problem descriptions generated by GPT-4o are syntactically parseable, 94.4% are solvable, but only 24.8% are semantically correct, highlighting the need for a more rigorous benchmark for this problem.

Figure 1: An example of one planning goal corresponding to many correct PDDL goals. All PDDL goals in the top row represent the displayed goal correctly. The bottom row illustrates PDDL goals with different error types, showing instances that are solvable (a planner can generate a plan, but for a different planning problem), parseable (the PDDL syntax is correct but will not produce any plan from a planner), and not parseable (it is not valid PDDL). See Section 5 for details.

1 Introduction

Currently only benchmark for evaluating ability of LLMs to translate natural language descriptions into structured formats (PDDL) suitable for planning tasks

→ Assumes ONLY task specification

What are the techniques for LLMs extracting these planning models?

PROBLEM

There is a fragmented landscape of NL-PDDL methods with each work possessing **different levels of assumptions:**

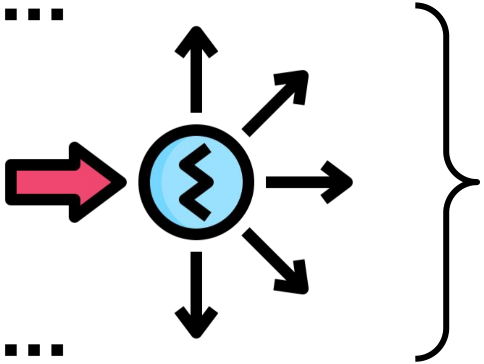
LLMs can be a bit … unpredictable

1. **Granularity** of natural language descriptions? — Explicit vs. Minimal Descriptions
2. What kind of **assumptions** are given? — Given info (levels of grounding)
3. What kind of **prompting styles** are used? — In-context, CoT, other styles…
4. What **generative techniques** are used? — Direct vs. Incremental Generation

MOTIVATION

Narrow down these techniques to provide actual insights to the limitations and advantages each of these works possess.

Huge list of existing pipelines / techniques / assumptions



Narrow it down for Pete's Sake!

The next steps towards applying LLMs in **real-world applications** is to establish a **standard, fair-comparison** of these frameworks – **What works? What doesn't work?**

Language-to-Plan (L2P)

With the proliferations of emerging NL-PDDL extraction techniques, we introduce **Language-to-Plan (L2P)**, an open-source Python library that **unifies NL-PDDL frameworks into to a single umbrella** which can then be tested on rigorous benchmarks.

L2P possesses capabilities of constructing core PDDL components that enables researchers to create their own NL-PDDL pipelines

Comprehensive Tool Suite: easily plug in various LLMs for streamlined extraction experiments with our extensive collection of PDDL extraction and refining tools.

Modular Design: facilitates flexible PDDL generation, allowing users to explore prompting and create customized pipelines.

Autonomous Capability: building block support for fully autonomous pipeline, reducing manual efforts of producing LLM-AP pipelines from scratch.

Language-to-Plan (L2P) – Examples

L2P can **recreate and encompass previous frameworks** for converting NL-PDDL, serving as a comprehensive foundation that integrates past approaches.

Some examples:

- ❏ LLM+P
- ❏ LLM-DM (example to the right)
- ❏ NL2Plan
- ❏ P+S
- ❏ PROC2PDDL

Shortened example of "Action-by-action" algorithm from (Guan et al. 2023)

```python
import os
from l2p import *

def run_aba_alg(model: LLM, action_model,
    domain_desc, hierarchy, prompt, max_iter: int=2
    ) -> tuple[list[Predicate], list[Action]]:
actions = list(action_model.keys())
pred_list = []

for _ in range(max_iter):
  action_list = []
  # iterate each action spec. + new predicates
  for _, action in enumerate(actions):
    if len(pred_list) == 0:
      prompt = prompt.replace('{predicates}',
      '\nNo predicate has been defined yet')
    else: res = ""
      for i, p in enumerate(pred_list):
        res += f'\n{i + 1}. {p["raw"]}'
        prompt = prompt.replace('{predicates}', res)
    # extract pddl action and predicates (L2P)
    pddl_action, new_preds, response = (
      builder.extract_pddl_action(
        model=model,
        domain_desc=domain_desc,
        prompt_template=prompt,
        action_name=action,
        action_desc=action_model[action]['desc'],
        action_list=action_list,
        predicates=pred_list,
        types=hierarchy["hierarchy"]
      )
    )
    new_preds = parse_new_predicates(response)
    pred_list.extend(new_preds)
    action_list.append(pddl_action)
  pred_list = prune_predicates(pred_list,action_list)
return pred_list, action_list
```

Other Concrete Examples

```python
import os
from l2p import *

domain_builder = DomainBuilder() # initialize Domain Builder class

# REPLACE WITH OWN API KEY
api_key = os.environ.get('OPENAI_API_KEY')
llm = OPENAI(model="gpt-4o-mini", api_key=api_key)

# retrieve prompt information
base_path='tests/usage/prompts/domain/'
domain_desc = load_file(f'{base_path}blocksworld_domain.txt')
extract_predicates_prompt = load_file(f'{base_path}extract_predicates.txt')
types = load_file(f'{base_path}types.json')
action = load_file(f'{base_path}action.json')

# extract predicates via LLM
predicates, llm_output = domain_builder.extract_predicates(
    model=llm,
    domain_desc=domain_desc,
    prompt_template=extract_predicates_prompt,
    types=types,
    nl_actions={action['action_name']: action['action_desc']}
    )

# format key info into PDDL strings
predicate_str = "\n".join([pred["clean"].replace(":", " ; ") for pred in predicates])

print(f"PDDL domain predicates:\n{predicate_str}")

-------------------------------------------------------------------------------------

### OUTPUT
(holding ?a - arm ?b - block) ;  true if the arm ?a is holding the block ?b
(on_top ?b1 - block ?b2 - block) ;  true if the block ?b1 is on top of the block ?b2
(clear ?b - block) ;  true if the block ?b is clear (no block on top of it)
(on_table ?b - block) ;  true if the block ?b is on the table
(empty ?a - arm) ;  true if the arm ?a is empty (not holding any block)
```

Figure 4: L2P usage - generating simple PDDL predicates

```python
import os
from l2p import *

task_builder = TaskBuilder() # initialize Task Builder class
api_key = os.environ.get('OPENAI_API_KEY')
llm = OPENAI(model="gpt-4o-mini", api_key=api_key)

# load in assumptions
problem_desc = load_file(r'tests/usage/prompts/problem/blocksworld_problem.txt')
extract_task_prompt = load_file(r'tests/usage/prompts/problem/extract_task.txt')
types = load_file(r'tests/usage/prompts/domain/types.json')
predicates_json = load_file(r'tests/usage/prompts/domain/predicates.json')
predicates: List[Predicate] = [Predicate(**item) for item in predicates_json]

# extract PDDL task specifications via LLM
objects, initial_states, goal_states, llm_response = task_builder.extract_task(
    model=llm,
    problem_desc=problem_desc,
    prompt_template=extract_task_prompt,
    types=types,
    predicates=predicates
    )

# format key info into PDDL strings
objects_str = task_builder.format_objects(objects)
initial_str = task_builder.format_initial(initial_states)
goal_str = task_builder.format_goal(goal_states)

# generate task file
pddl_problem = task_builder.generate_task(
    domain="blocksworld",
    problem="blocksworld_problem",
    objects=objects_str,
    initial=initial_str,
    goal=goal_str)

print(f"### LLM OUTPUT:\n {pddl_problem}")
---------------------------------------------------------------------------------
### LLM OUTPUT
(define
   (problem blocksworld_problem)
   (:domain blocksworld)
   (:objects
      blue_block - object
      red_block - object
      yellow_block - object
      green_block - object
   )
   (:init
      (on_top blue_block red_block)
      (on_top red_block yellow_block)
      (on_table yellow_block)
      (on_table green_block)
      (clear blue_block) (clear yellow_block) (clear green_block)
   )
   (:goal
      (and (on_top red_block green_block))
   )
)
```

Figure 5: L2P usage - generating simple PDDL task specification

~~~~python
import os
from l2p import *

feedback_builder = FeedbackBuilder()
api_key = os.environ.get('OPENAI_API_KEY')
llm = OPENAI(model="gpt-4o-mini", api_key=api_key)

problem_desc = load_file(r'tests/usage/prompts/problem/blocksworld_problem.txt')
types = load_file(r'tests/usage/prompts/domain/types.json')
feedback_template = load_file(r'tests/usage/prompts/problem/feedback.txt')
predicates_json = load_file(r'tests/usage/prompts/domain/predicates.json')
predicates: List[Predicate] = [Predicate(**item) for item in predicates_json]
llm_response = load_file(r'tests/usage/prompts/domain/llm_output_task.txt')

objects, initial, goal, feedback_response = feedback_builder.task_feedback(
    model=llm,
    problem_desc=problem_desc,
    feedback_template=feedback_template,
    feedback_type="llm",
    predicates=predicates,
    types=types,
    llm_response=llm_response)

print("FEEDBACK:\n", feedback_response)
---------------------------------------------------------------------------------
### LLM OUTPUT
FEEDBACK:
1. Are any necessary objects missing?
   All necessary objects are included based on the problem description. So: No.
2. Are any unnecessary objects included?
   All objects included are relevant to the problem. Hence: No.
3. Are any objects defined with the wrong type?
   All objects are correctly defined as "object". Therefore: No.
4. Are any unnecessary or incorrect predicates declared?
   All predicates used in the initial state are relevant and correctly applied. Thus: No.
5. Are any needed or expected predicates missing from the initial state?
   The initial state is missing the predicate for the red block being clear. Since the red block is covered by
   the blue block, it should not be clear. Therefore: Yes.
6. Is anything missing from the goal state?
   The goal state accurately reflects the desired outcome of having the red block on top of the green block.
        So: No.
7. Is anything unnecessary included in the goal description?
   The goal description is concise and only includes what is necessary. Therefore: No.
8. Should any predicate be used in a symmetrical manner?
   The predicates used do not require symmetry as they are directional in nature. So: No.

My concrete suggestions are the following:
- Add the predicate to indicate that the red block is not clear:
  - (clear red_block) should be removed from the initial state since the red block is covered by the blue
      block.

Final output should reflect this change in the initial state:
```
(clear blue_block): blue block is clear
(clear yellow_block): yellow block is clear
(clear green_block): green block is clear
```

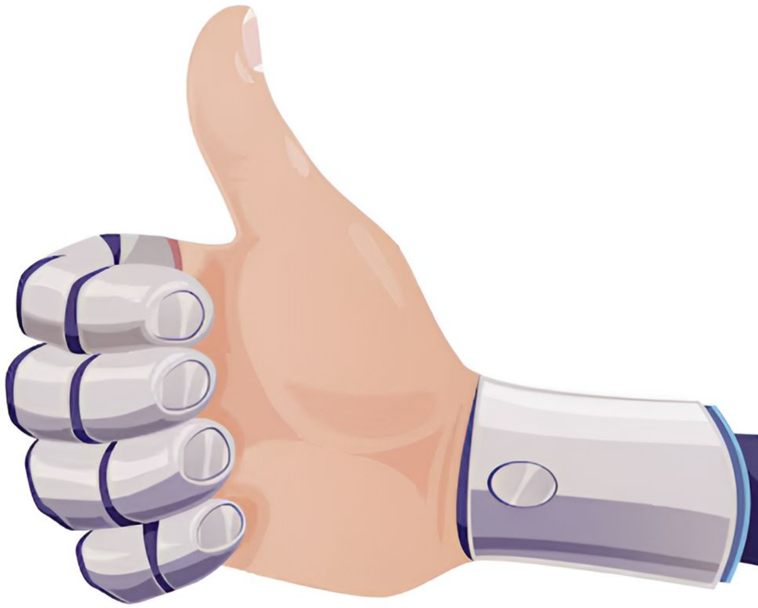
Overall, the feedback is: Yes, the initial state needs to reflect that the red block is not clear.
~~~~

Figure 6: L2P usage - generating LLM-feedback on task specification

Conclusion

In summary, **L2P** is a toolkit that consolidates various NL-PDDL approaches under one roof; this unification is easier for researchers to compare techniques and build upon existing works.

Our library contains a list of current NL-PDDL works and **we invite the community to adopt this kit into their projects**, building a repertoire of standardized benchmarks and methodologies, enabling fair comparison and drive of high-impact research in this field.



Any Questions?

https://github.com/AI-Planning/l2p