



Fusing Planning Capabilities with LLM Agents

Status Quo: Agents are everywhere

- Frameworks: Transformer Agents (Huggingface), Langchain Agents etc.
- Usage: AutoGPT, BabyAGI
- Inner workings:
 - Prompt with different Tools + Descriptions
 - LLM has to decide which Tools to use when





Problem: LLMs are bad at planning

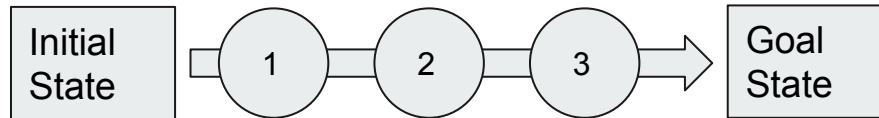
- Common Errors:
 - LLM hallucinates Tools
 - LLM uses Tools in wrong order
 - LLM gets stuck
- LLM-based planning is **expensive**

Domain	Method	Instances correct			
		GPT-4	GPT-3.5	Instruct-GPT3	GPT-3
Blocksworld (BW)	One-shot	206/600 (34.3%)	37/600 (6.1%)	41/600 (6.8%)	6/600 (1%)
	Zero-shot	210/600 (34.6%)	8/600 (1.3%)	-	-
	COT	214/600 (35.6%)	-	-	-
Logistics Domain	One-shot	28/200 (14%)	1/200 (0.5%)	3/200 (1.5%)	-
	Zero-shot	15/200 (7.5%)	1/200 (0.5%)	-	-
Mystery BW (Deceptive)	One-shot	16/600 (2.6%)	0/600 (0%)	7/600 (1.1%)	0/600 (0%)
	Zero-shot	1/600 (0.16%)	0/600 (0%)	-	-
	COT	53/600 (8.8%)	-	-	-
Mystery BW (Randomized)	One-shot	11/600 (1.8%)	0/600 (0%)	5/600 (0.8%)	1/600 (0.1%)
	Zero-shot	0/600 (0%)	0/600 (0%)	-	-

[\[2305.15771\] On the Planning Abilities of Large Language Models -- A Critical Investigation](#)

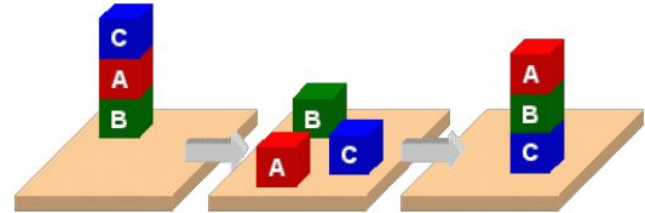
Back to the Roots: AI Planning

- 1970s-2000s: Different AI Planning Languages
 - Description of **initial state** of the world
 - Description of **actions**
 - Description of **goal state** to reach
- PDDL (1998): Planning Domain Definition Language
 - Became the standard
 - Used mainly in Robotics today



“Hello World” in PDDL

- BlocksWorld
- PDDL consist of two files:
 - “Domain” -> Description of Actions the Agent can take
 - similar to compiled code
 - “Problem” -> Description of Initial State + Goal
 - **based on Domain**
 - similar to User Inputs





Blocksworld: Domain

```
(define (domain BLOCKS)
  (:requirements :strips)
  (: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)))
```

Blocksworld: Domain

```
(define (domain BLOCKS)
  (:requirements :strips)
  (: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))))
```

“Imports”

“Descriptions” (can be true or false)

Blocksworld: Domain

```
(define (domain BLOCKS)
  (:requirements :strips)
  (: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)))
```

Input

Action/Function

Blocksworld: Domain

```
(define (domain BLOCKS)
  (:requirements :strips)
  (: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)))
```

State(s) of World
needed

Effect on World

Blocksworld: Problem

```
(define (problem BLOCKS-10-0)
  (:domain BLOCKS)
  (:objects A B C)
  (:INIT |
    (CLEAR C) (CLEAR B) (CLEAR A)
    (ONTABLE C) (ONTABLE A) (ONTABLE B)
    (HANDEEMPTY))
  (:goal
    (AND (ON C B) (ON B A)))
)
```

Defintion of
Objects / Blocks

Initial State of the
World

Goal State of the World
which we want



Blocksworld: Plan

- Domain + Problem are given into a Planner
- Planner searches for a solution for the given problem

Found Plan (output)

(pick-up b)

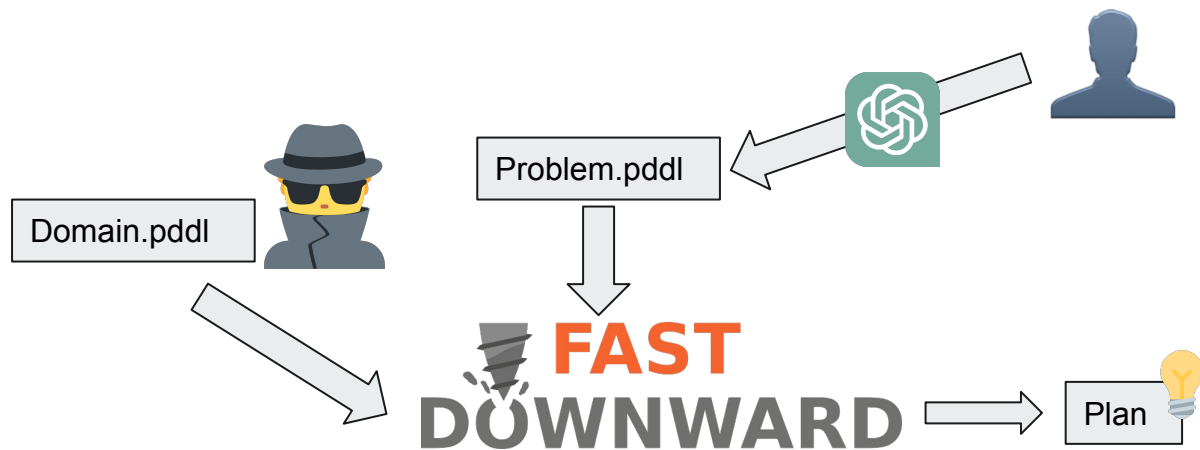
(stack b a)

(pick-up c)

(stack c b)

```
(:action pick-up
:parameters (b)
:precondition
  (and
    (clear b)
    (ontable b)
    (handempty)
  )
:effect
  (and
    (not
      (ontable b)
    )
    (not
      (clear b)
    )
    (not
      (handempty)
    )
    (holding b)
  )
)
```

Architecture: Fusing LLMs with PDDL



Practical Example

Model: GPT-4

S4

Write a PDDL domain.

The domain describes an agent which has is able to research topics and format the research topics.

The following actions are needed:

- document_search: This search covers topics about **films**. After the search is done, the agent has gained knowledge about the topic. To call this, the topic has to be film related.
- web_search: This search covers topics about **actors**. After the search is done, the agent has gained knowledge about the topic. To call this, the topic has to be related to an actor.
- format_answer: This formats the presentation of a topic. To call this, the agent has to have knowledge about the topic first.

Similar to Agent
Tools





Domain.pddl

```
(define (domain research_agent)
  (:requirements :strips :typing :negative-preconditions)
  (:types
    topic
  )
  (:predicates
    (film_topic ?t - topic)
    (actor_topic ?t - topic)
    (knowledge_about ?t - topic)
    (formatted ?t - topic)
  )

  (:action document_search
    :parameters (?t - topic)
    :precondition (and (film_topic ?t))
    :effect (and (knowledge_about ?t))
  )
)
```

```
(:action web_search
  :parameters (?t - topic)
  :precondition (and (actor_topic ?t))
  :effect (and (knowledge_about ?t))
)

(:action format_answer
  :parameters (?t - topic)
  :precondition (and (knowledge_about ?t)
                    (not (formatted ?t)))
  :effect (and (formatted ?t))
)
```



Problem Prompt

S4

I have this PDDL domain file:

```
`` `pddl
(define (domain research_agent)
  (:requirements :strips :typing :negative-preconditions)
  <trunc't>
)
```

Now write the PDDL problem for it. I have no knowledge about anything and I want to have a formatted answer about the James Bond Movie.



Problem.pddl

```
(define (problem research_problem)
  (:domain research_agent)
  (:objects
    James_Bond_Movie - topic
  )
  (:init
    (film_topic James_Bond_Movie)
  )
  (:goal
    (and (knowledge_about James_Bond_Movie)
          (formatted James_Bond_Movie))
  )
)
```



Resulting Plan

Found Plan (output)

```
(document_search james_bond_movie)
```

```
(format_answer james_bond_movie)
```

```
(:action document_search
:parameters (james_bond_movie)
:precondition
  (and
    (film_topic james_bond_movie)
  )
:effect
  (and
    (knowledge_about james_bond_movie)
  )
)
```



Pros & Cons

- You have to know your goal beforehand
- PDDL is pretty academic
 - bad software support
 - language has many “dialects” with different features
- + PDDL is fast and cheap (for these problems)
- + Open Source LLMs can convert NLP -> PDDL pretty good



Additional Resources

- Planning Wiki: <https://planning.wiki/>
- Online PDDL Editor + Planner: <http://editor.planning.domains/>
- Planner: FastDownward (<https://www.fast-downward.org/>) [GPL3]
- Even more resources: <http://education.planning.domains/>
- LLM as Domain.pddl Generator: <https://arxiv.org/abs/2305.14909>
- SayCan: <https://say-can.github.io/>
- Improving Reasoning with RL:
<https://openai.com/research/improving-mathematical-reasoning-with-process-supervision>