

Paper Presentation on

Open-World Task and Motion Planning via Vision-Language Model Inferred Constraints

Boxiang (William) Fu

16-832 (Spring 2026) Integrated Planning and Learning
Carnegie Mellon University

01/26/2026



Paper to Cover

- **Title:** Open-World Task and Motion Planning via Vision-Language Model Inferred Constraints
- **Authors:** Nishanth Kumar, William Shen, Fabio Ramos, Dieter Fox, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Caelan Reed Garrett
- **Affiliation:** MIT CSAIL and NVIDIA Research
- **Year:** 2025
- **Website:** <https://nishanthjkumar.com/owl-tamp/>
- **ArXiv:** <https://arxiv.org/abs/2411.08253>

TLDR Summary

- Describes a way to combine Vision-Language Models (VLMs) with classical Task and Motion Planning (TAMP) by having the VLM generate task-specific constraints from natural language prompt, which a TAMP system then plans and executes.
- Solves the problem of VLMs performing poorly in long-horizon manipulation tasks.

Motivation

- Two mainstream methods for manipulation:
- **Classical:** Task and Motion Planning (TAMP)
 - Method: Symbolic goal expression of planned tasks
 - Advantages: Capable of solving specific long horizon complex tasks
 - Problem: Cannot perform tasks for which their model is unspecified (e.g. open world scenarios)
- **New:** Vision-Language Models (VLMs)
 - Method: LLM or Neural-Network architecture using datasets
 - Advantages: Can adapt and solve a wide variety of short horizon tasks
 - Problem: Performs poorly over long horizon periods

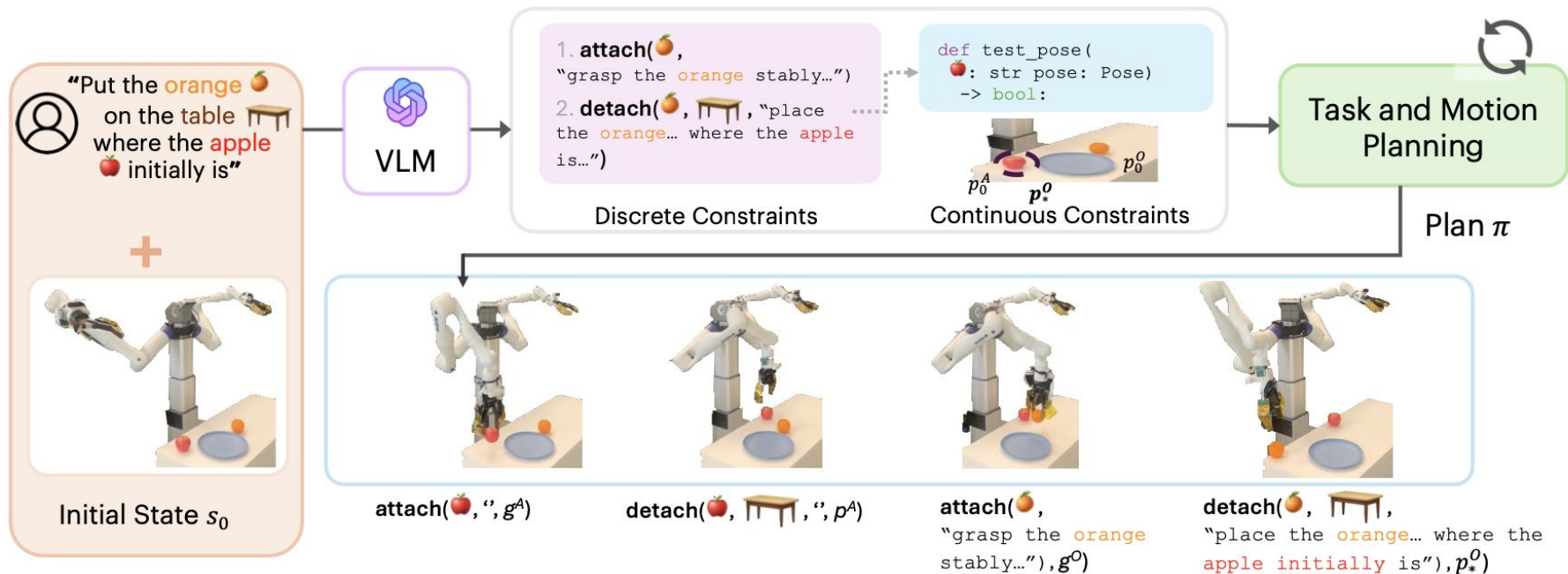
Related Work

- **Task and Motion Planning (TAMP)**: Symbolic goal expression of planned tasks. Classical approach for long-horizon robot manipulation tasks
- **LLMs/VLMs (Task Planning)**: Generate symbolic plans from natural language. Utilize low level executor to execute the generated symbolic plan
- **Code as Policy (CoP)**: Bypasses explicit symbolic planning. Generates executable programs that directly compose low-level skills into a policy
- **LLMs/VLMs (End-to-End)**: Directly generate motor commands from input natural language prompt. Currently very bad for long-horizon tasks

OWL-TAMP

- Fuse together the “best of both worlds”
- Integrate TAMP and VLMs through the use of “constraints”
- VLM infers from natural language and tells TAMP what plans or parameter values are allowed
- Provides logical structure of the long horizon plan
- It does not directly control the robot how to move – this is still controlled by the TAMP controller
- For example:
 - “Cook the strawberry by putting it in the pan, then serve it in the bowl”
 - detach(strawberry, pan) **must occur before** detach(strawberry, bowl)

OWL-TAMP



Assumptions

1. Tasks are specified by the user in unambiguous natural language
2. Scene is static with no movement other than the robot
3. An off-the-shelf VLM/LLM model that can understand sequential partial tasks and convert it to pseudocode with relatively good accuracy
4. An off-the-shelf TAMP model which constraints commonplace manipulation primitives applicable across a wide range of tasks

Methodology

1. Method for generating constraints on action sequences to specify partial plans with language descriptions
2. Method for generating constraints on continuous variables affected within the partial plan from (1)
3. Combining (1) and (2) within a TAMP system

Methodology – Step 1

- Prompt VLM to generate a partial plan of discrete constraints for the TAMP
- Problem: VLMs hallucinate and can provide impossible solutions
- Solution:
 - Ground the set of actions and literals (True/False states of the world) to only ones that are reachable
 - Prompt the VLM to only return solutions in this grounded set

Algorithm 1 VLM Task Reasoning

```
1: procedure VLM-TASK-REASONING( $s_0, \mathcal{A}, g$ )
2:    $A \leftarrow \text{GROUND-ACTIONS}(s_0, \mathcal{A})$ 
3:    $L \leftarrow s_0 \cup \{l \mid a \in A. l \in e.\text{eff}\}$ 
4:    $[a_1, \dots, a_n, l_1, \dots, l_m] \leftarrow \text{QUERY-VLM}(\text{"What partial plan using actions } \{A\} \text{ for goal literals } \{L\} \text{ achieves goal } \{g\}?\text{"})$ 
5:   for  $i \in [1, n - 1]$  do
6:      $a_i.\text{eff} \leftarrow a_i.\text{eff} \cup \{\text{Executed}(i)\}$ 
7:      $a_{i+1}.\text{pre} \leftarrow a_{i+1}.\text{pre} \cup \{\text{Executed}(i)\}$ 
8:    $a_n.\text{eff} \leftarrow a_n.\text{eff} \cup \{\text{Executed}(n)\}$ 
9:    $G \leftarrow \{l_1, \dots, l_m\}$ 
10:  return SOLVE-TAMP( $s_0, A, G \cup \{\text{Executed}(n)\}$ )
```

Legend:

- s : state
- A : action set
- L : literal set
- g : goal
- $.\text{eff}$: effect
- $.\text{pre}$: predicate
- G : partial plan

Methodology – Step 2

- Turn the partial plan generated from Step 1 into a full legal plan that TAMP can use
- Problem: Partial plan from Step 1 could be open-ended natural language

`detach(“place the mug stably on the table
ensuring it is upright and positioned
to receive the coffee”, mug, ...),`

- Solution:
 - Introduce a predicate (boolean) function that converts the natural language into code that can be tested

Methodology – Step 2

```
def test_poses(p) -> bool:
    ontop_table_bounds =
        modify_pose_bounds_to_be_ontop
        _of_object('mug', 'table')
    mug_on_table =
        position_within_bounds(mug.pose,
        ontop_table_bounds)
    upright_orientation = abs(mug.pose.roll)
        < 0.1 and abs(mug.pose.pitch) < 0.1
    return mug_on_table and
        upright_orientation
```

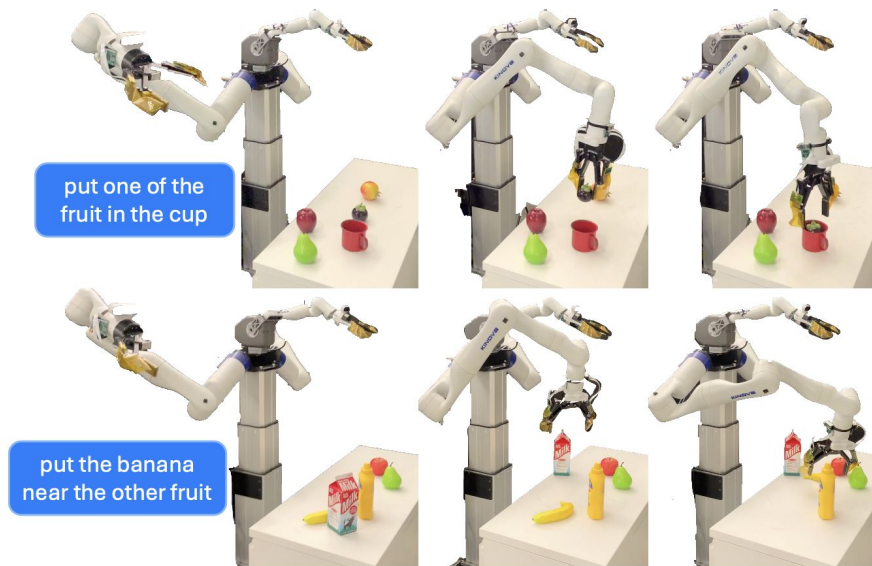
- Once the full plan is sent to the TAMP planner, it can check against this test to ensure the generated plan satisfies the original natural language partial plan
- Forces TAMP system to output solutions given this constraint

Methodology – Step 3

- Standard search-then-sample TAMP
- **Search Phase:** A* search over action sequences to generate symbolic task plan
- **Sample Phase:** Hand-crafted sampling of continuous environment and manipulator configurations to check if pose is valid or move is legal
- **Backtracking:** If sampling fails, backtrack to search phase to try a different skeleton plan
- **Result:** Terminates when the sampling phase passes all checks

Experiments

- **Tasks:** 10 of varying difficulty, all require multi-sequence planning
 - Example: *"I want to pour some coffee into the cup; can you set up the cup on the table so I can do this properly?"*
- **Environment:** Two Kinova Gen3 manipulator arms
- **Items:** YCB dataset
- **OD:** Grounding DINO
- **VLM:** GPT-4o (CoT)

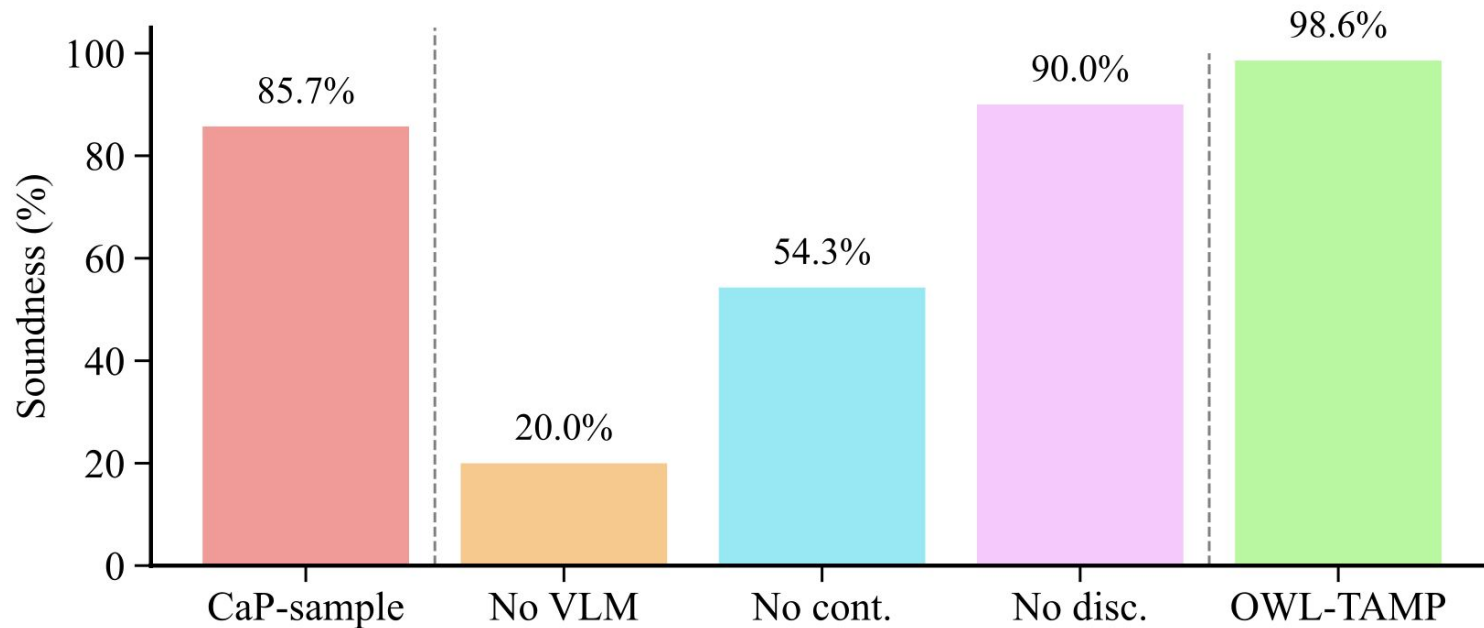


Results – Success Rate

Method	Tasks										Overall
	Berry1	Citrus	Berry2	BerryCook	FruitSort	Coffee	Mug1	Mug2	Mug3	SoupPour	
CaP	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
CaP-sample	100%	20%	20%	0%	0%	0%	0%	0%	0%	0%	14%
No VLM	100%	100%	100%	0%	0%	0%	0%	0%	20%	0%	32%
No sample	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
No disc.	100%	100%	100%	0%	0%	0%	0%	0%	20%	0%	32%
No cont.	100%	100%	100%	100%	10%	60%	70%	0%	0%	20%	56%
No back.	100%	90%	100%	60%	80%	100%	40%	30%	0%	0%	60%
OWL-TAMP	100%	100%	100%	60%	100%	100%	100%	100%	70%	90%	92%

- **CaP**: Code as Policy

Results – Soundness



- **Soundness:** Higher rates indicate lower false positives (i.e. algorithm saying natural language goal was completed but in reality it did not)

Results – Wall Clock Time

Method	Tasks									
	Berry1	Citrus	Berry2	BerryCook	FruitSort	Coffee	Mug1	Mug2	Mug3	SoupPour
CaP	10.03	21.27	12.63	13.77	19.50	9.66	19.77	23.25	16.37	19.33
	±0.32 (47.38%)	±1.10 (25.02%)	±1.13 (42.76%)	±0.95 (45.88%)	±0.67 (26.14%)	±0.80 (47.61%)	±2.84 (46.16%)	±1.65 (30.42%)	±1.49 (38.14%)	±1.17 (42.88%)
CaP-sample	22.40	248.49	245.31	262.43	20.88	40.39	281.74	75.61	274.50	187.90
	±12.70 (21.22%)	±54.38 (2.14%)	±35.87 (2.20%)	±14.12 (2.41%)	±2.05 (24.41%)	±16.21 (11.39%)	±2.99 (3.24%)	±78.73 (9.35%)	±11.97 (2.27%)	±83.47 (4.41%)
No VLM	11.90	41.59	47.66	15.96	16.56	4.92	59.43	78.72	36.03	21.28
	±1.35 (8.85%)	±4.85 (4.53%)	±3.20 (1.60%)	±1.23 (29.77%)	±0.14 (24.60%)	±0.04 (10.25%)	±31.91 (4.11%)	±18.85 (6.36%)	±11.85 (8.04%)	±2.54 (20.83%)
No sample	44.00	75.17	59.47	54.93	81.19	39.95	55.07	80.00	46.22	91.46
	±4.03 (85.97%)	±5.26 (72.29%)	±4.69 (83.77%)	±4.68 (82.13%)	±10.40 (74.17%)	±4.28 (85.49%)	±3.43 (73.59%)	±9.63 (69.38%)	±2.57 (70.58%)	±29.02 (82.12%)
No disc.	44.92	91.62	80.87	44.50	36.10	18.92	118.42	120.94	81.85	96.95
	±4.07 (70.03%)	±6.22 (50.92%)	±6.83 (33.37%)	±4.38 (69.50%)	±2.01 (59.87%)	±2.96 (72.29%)	±16.71 (20.71%)	±9.99 (29.63%)	±10.58 (26.85%)	±6.37 (72.54%)
No cont.	18.15	54.53	36.91	24.30	57.03	25.73	52.26	136.88	102.53	91.00
	±1.84 (40.92%)	±13.09 (16.21%)	±5.61 (29.48%)	±1.91 (29.40%)	±9.88 (15.68%)	±4.13 (29.93%)	±16.08 (14.61%)	±5.72 (5.72%)	±22.20 (6.04%)	±20.21 (9.42%)
No back.	48.69	106.36	101.84	67.94	149.53	57.68	87.31	106.82	69.59	111.50
	±3.40 (71.92%)	±24.85 (60.35%)	±34.28 (72.59%)	±15.09 (75.11%)	±39.78 (66.52%)	±8.20 (65.30%)	±21.78 (58.23%)	±6.13 (54.49%)	±3.02 (52.24%)	±32.13 (67.36%)
OWL-TAMP	46.92	108.50	101.88	71.38	154.32	57.96	113.18	140.51	118.38	173.81
	±3.37 (74.64%)	±24.61 (59.15%)	±34.47 (72.56%)	±15.31 (71.49%)	±43.43 (64.46%)	±7.69 (64.98%)	±28.50 (44.92%)	±23.06 (41.43%)	±23.74 (30.71%)	±49.00 (43.21%)
Manual	13.53	44.42	24.84	19.78	49.72	22.15	45.18	79.91	37.64	81.38
	±2.36 (0.00%)	±5.34 (0.00%)	±2.78 (0.00%)	±1.34 (0.00%)	±3.59 (0.00%)	±6.36 (0.00%)	±16.74 (0.00%)	±33.04 (0.00%)	±10.40 (0.00%)	±41.85 (0.00%)

Limitations

- There is no provision for recovering from errors generated by the VLM (Step 1 and Step 2)
- There is no feedback loop between the VLM and TAMP. If the TAMP system declares a problem infeasible, a feedback loop would allow the VLM to replan a more feasible solution
- Hard to get a large dataset of robot manipulation tasks to finetune the VLM model
- If the algorithm returns successfully finished, it is not guaranteed that the actual result indeed satisfies the criteria specified in the original natural language goal
- No theoretical guarantees of solution quality
- Takes a very long time (buried deep in the appendix...)

Conclusion

- Authors propose OWL-TAMP, integrating VLM with TAMP by using language to generate discrete and continuous planning constraints
- Merges the best of both worlds, essentially assigning the “reasoning” to the VLM, and long-horizon planning to the TAMP
- Results shows OWL-TAMP can solve a wider range of multi-step problems compared to the benchmark
- Some limitations of the algorithm include no recovery procedures, lack of quality data, no theoretical guarantees and long runtime complexity

Thank You!