Guiding Long-Horizon Task and Motion Planning with Vision Language Models

Zhutian Yang[1,2*] Caelan Garrett[2] Dieter Fox[2] Tomás Lozano-Pérez[1] Leslie Pack Kaelbling[1]
[1]Massachusetts Institute of Technology, [2]NVIDIA Research

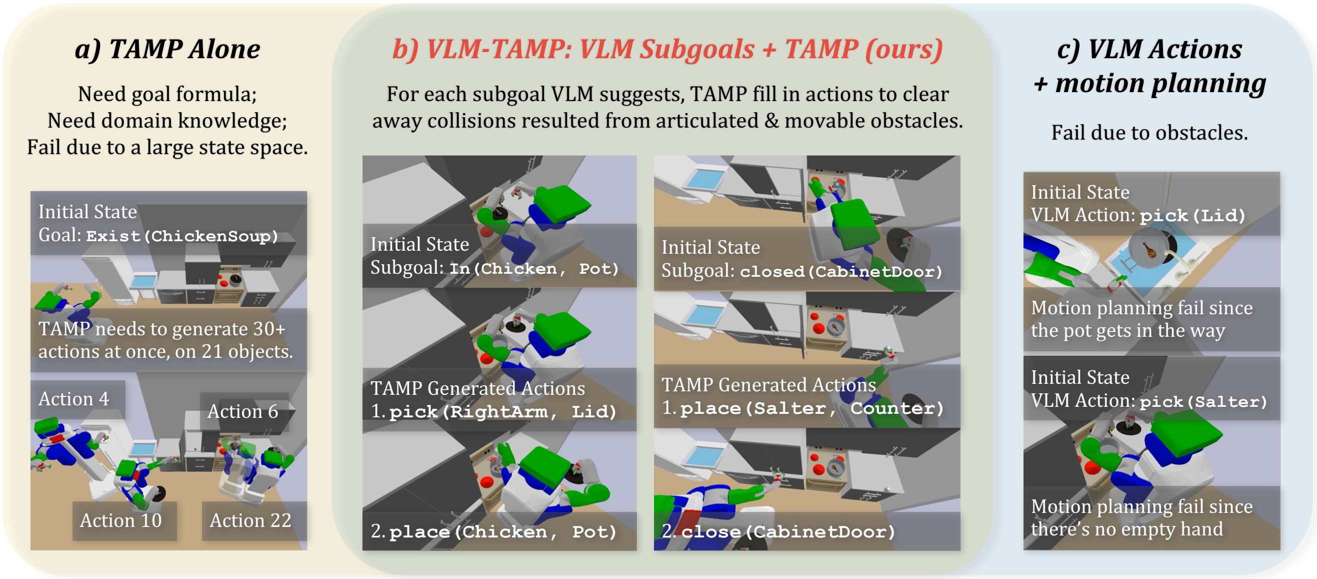

a) TAMP Alone
Need goal formula;
Need domain knowledge;
Fail due to a large state space.
Initial State
Goal: Exist(ChickenSoup)
TAMP needs to generate 30+ actions at once, on 21 objects.
Action 4
Action 6
Action 10
Action 22
b) VLM-TAMP: VLM Subgoals + TAMP (ours)
For each subgoal VLM suggests, TAMP fill in actions to clear away collisions resulted from articulated & movable obstacles.
Initial State
Subgoal: In(Chicken, Pot)
TAMP Generated Actions
1. pick(RightArm, Lid)
2. place(Chicken, Pot)
Initial State
Subgoal: closed(CabinetDoor)
TAMP Generated Actions
1. place(Salter, Counter)
2. close(CabinetDoor)
c) VLM Actions + motion planning
Fail due to obstacles.
Initial State
VLM Action: pick(Lid)
Motion planning fail since the pot gets in the way
Initial State
VLM Action: pick(Salter)
Motion planning fail since there's no empty hand

Fig. 1: Our approach VLM-TAMP overcomes the pitfalls of using TAMP alone and VLM task *then* motion planning when solving long-horizon robot manipulation problems. **a)** Pure TAMP fails when there are large state spaces and a long-horizon goals. **c)** VLMs fail at geometric reasoning by predicting actions that cannot be safely refined with motion planning. **b)** VLM-TAMP uses VLM to take in a natural language goal, generate subgoals, solve a sequence of smaller problems that respect all geometric constraints using TAMP.

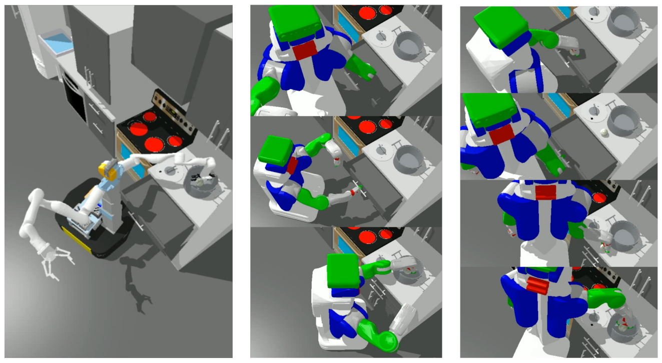
***Abstract*— Vision-Language Models (VLM) can generate plausible high-level plans when prompted with a goal, the context, an image of the scene, and any planning constraints. However, there is no guarantee that the predicted actions are geometrically and kinematically feasible for a particular robot embodiment. As a result, many prerequisite steps such as opening drawers to access objects are often omitted in their plans. Robot task and motion planners can generate motion trajectories that respect the geometric feasibility of actions and insert physically necessary actions, but do not scale to everyday problems that require common-sense knowledge and involve large state spaces comprised of many variables. We propose VLM-TAMP, a hierarchical planning algorithm that leverages a VLM to generate both semantically-meaningful and horizon-reducing intermediate subgoals that guide a task and motion planner. When a subgoal or action cannot be refined, the VLM is queried again for replanning. We evaluate VLM-TAMP on kitchen tasks where a robot must accomplish cooking goals that require performing 30-50 actions in sequence and interacting with up to 21 objects. VLM-TAMP substantially outperforms baselines that rigidly and independently execute VLM-generated action sequences, both in terms of success rates (50 to 100% versus 0%) and average task completion percentage (72 to 100% versus 15 to 45%). See project site https://zt-yang.github.io/vlm-tamp-robot/ for more information.**

*Research performed partly during an internship at NVIDIA.

I. Introduction

Large Language Models (LLMs) contain an enormous amount of common-sense and cultural knowledge through training on internet-scale datasets [1], [2]. They can suggest high-level courses of action for solving almost any problem, ranging from selling your house to making a meal. Vision Language Models (VLMs) extend the capabilities of LLMs, by conditioning on an input image, allowing problems to be described both textually and via one or more pictures.

However, neither method is capable of detailed geometric reasoning: they don't understand whether a particular robot with particular kinematics can reach something or whether it's possible to fit two particular pans into the oven simultaneously. Furthermore, because they are trained on human-generated text, which usually only contains the most important aspects of a plan but leaves unstated many steps that are obvious to a human (you have to open the fridge to get the eggs, it's a good idea to close it again, you should extract the egg from the shell before adding it to your cake, etc.) The plans they generate are error-prone and partial. Besides, the same plan may be feasible for some robot embodiment but not the others depending on the robot



(a) Dual-arm Rummy directly picks and places the cabbage as its arms are long enough to reach far. (b) Dual-arm PR2 first closes the drawer to make space for reaching the pot. (c) Single-arm PR2 first puts the cabbages aside before closing the drawer.

Fig. 2: Example trajectories of different robots achieving the same goal of having the cabbage in the pot, where the cabbage is placed in a drawer and the pot is hard to reach. While the VLM may not be able to generate feasible action plans based on text and image description of the scene, TAMP can find the shortest feasible task plans that move obstacles if necessary and respect the kinematic constraints of the robot and other articulated objects.

reachability, as shown in an example in Figure 2.

In the robotics research community, task and motion planning (TAMP) [3] methods can solve complex long-horizon manipulation problems. These planners are sound and (semi) complete: that is, given an accurate model of the domain, and a problem to be solved that satisfies some common assumptions, they are *guaranteed* to eventually produce a detailed plan, at the level of robot joint commands, that achieve the goal, if such a plan exists. Two weakness of TAMP approaches are how they 1) handle open-world *semantics*, they can only address the geometric and kinematic aspects of the problem (they can plan to put a chicken in a pot, but don't know what it means to make tasty soup), and 2) scale *computationally*, the solution time explodes with complexity of the problem (the number of objects that need to be manipulated and the length of the solution plan).

In this paper, we present VLM-TAMP, a system that combines the great strengths of VLMs and TAMP (Figure 1). It uses a VLM to suggest a sequence of *subgoals* such that achieving them in order could achieve a higher-level common-sense goal (like making tasty soup). These subgoals are then solved by a TAMP algorithm, which can fill in missing geometric details, insert new steps (e.g., opening a cupboard) if needed, and approximately detect infeasibility. The system executes the actions after solving each subgoal. In case one subgoal suggested by the VLM cannot be grounded in the planning domain or is geometrically infeasible, VLM-TAMP reprompts the VLM to generate a new subgoal sequence given the current state of the world and objects that are collided in simulation during the last failed planning process.

We evaluate VLM-TAMP on two robot embodiments, solving cooking problems in procedurally generated kitchen scenes. The problems consist of making chicken soup with different scene initial conditions, requiring task plans that range from 30 to 50 actions. We compare VLM-TAMP extensively to a more common strategy, in which the VLM is asked to produce a sequence of *actions*, which are then directly refined by sampling the necessary continuous parameters, and executed in the world. We find that the ability of VLM-TAMP to make up for deficiencies in the high-level plan makes it much superior. Our experiments show that, in a challenging set of very long-horizon problems, VLM-TAMP succeeds 50 to 100% of the time, where as the baseline, a naive VLM predicting actions + motion planning approach never succeeds. When we count task progress, the baseline completes on average only 15 to 45% of the subproblems before failing to refine an action the VLM suggested, even given chances to reprompt VLM with collision information. In comparison, ours, which asks VLM for subgoals, visibly benefit from reprompting, with task success increasing by 47 to 55% on the hardest problems with 21 planning objects in a layout that resembles real-world kitchens.

II. Related Work

LLMs and VLMs have been used to translate natural language task description to formal languages [4], [5], [6], which a model-based planner can consume and solve. They have been prompted to generate high-level action sequences that are then implemented by pre-trained skill policies [7], [8]. They have also been used to generate code for calling robot motion primitives [9], [10] and guiding trajectory generation [11]. These methods do not explicitly handle the problem of geometrically infeasible action sequences.

To correct the infeasible task plans generated by LLMs due to obstacles or partial observability, [12], [13], [14], and [15] provide replanning abilities by prompting the VLM to adapt robot actions when the initial plan fails to achieve the desired goal. These methods showed improved success rate on small-scale table-top rearrangement tasks. But using the VLM or LLM for geometric reasoning assumes that the infeasibility can be described with a given text template "some object is blocking the goal object", which does not really scale to larger environments, especially involving mobile manipulators. For example, when the goal is to pick up a pepper shaker from a small cabinet and the robot has only one arm, the actions to resolve the geometric infeasibility would involve putting down the object in the hand and opening both cabinet doors. While a VLM is unlikely to come up with all these actions, a TAMP planner can.

[16] prevents LLMs from proposing actions that violate geometric constraints by querying the LLMs to generate the constraints, such as that the robot is holding an object and thus unable to pick up another object. Designing such queries is equivalent to providing the planning domain knowledge for checking violations. To incorporate geometric feasibility reasoning directly into LLM and VLM planning, [17] proposed a shooting-based strategy, where the LLM proposes K task plans, then geometric feasibility planning is carried out to find continuous parameters, the sequences' success probability and predicted future states. Their method considers the geometric dependencies spanning the whole

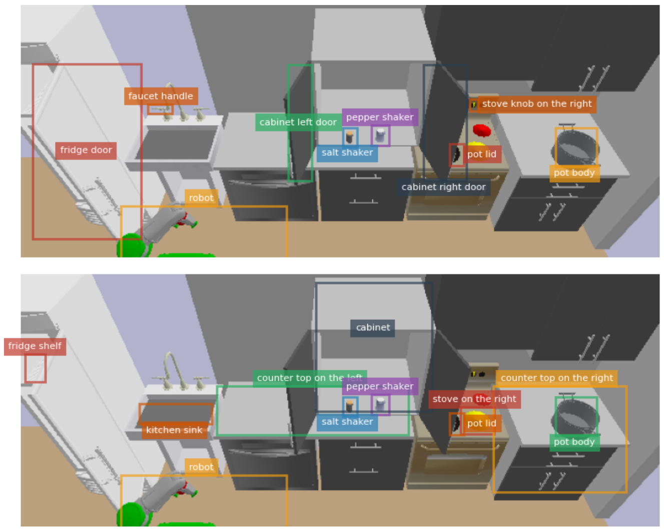


Fig. 3: An example input image to the VLM, which are annotated with object names and bounding boxes. The top image marks movable objects and articulated joints, while the bottom image marks movable objects and placement surfaces.

action sequence, but the planner depends on Q-functions that are specific to the robot embodiment and scene layout. Furthermore, the method doesn't enable local geometric (in)feasibility to guide which task plans to try next.

Our work differs from all these methods in that we use an LLM to generate intermediate goals instead of actions [18], and deploy TAMP planner to achieve them. This enables VLM-TAMP to substantially modify an LLM's suggested partial plan by adding steps and by solving continuous parameters that ensure geometric and kinematic feasibility.

III. Method

We present VLM-TAMP, a planning algorithm that uses a VLM to break down a long-horizon manipulation planning problem (defined in Section III-A) into a sequence of smaller ones (Section III-C), which a TAMP planner solves in sequence to satisfy geometric feasibility (Section III-D). VLM-TAMP also contains a replanning mechanism that deals with mistakes in VLM goal translation, infeasible task plans, and TAMP planning failures (Section III-E).

A. Problem Formulation

We assume that problems are represented as correct 3D geometric model of the world plus the robot and a natural language goal, e.g. "make chicken soup". Because the geometric model of the world cannot be fully communicated to a VLM with image renders of the scene alone, we provide the VLM a text description of the scene that lists the objects and the relations they satisfy. For example, Figure 3 is generated by rendering the scene in the PyBullet [19] simulator and labeling the observable objects using ground-truth semantic segmentation. This can also be generated with semantic segmentation models for real-world experiments.

The system outputs a sequence of robot commands (joint angle trajectories) to satisfy each subproblem identified by

```
Plan a short sequence of [OUTPUT] that accomplishes
    ↪ the following {goal}.
[RESPOND_WITH] where <obj>, <surface>, <joint>, <
    ↪ button>, <handle> must be items from the
    ↪ following {planning_objects}.
Currently, you can see {object_relations}. The
    ↪ accompanying image ... {img}
{action_history} {failed_action} {collided_objects}
```

a) The initial question to the VLM. Text formatted using **[RED]** differs when asking for *subgoals* versus *actions*.

```
Open the fridge door to access the
    ↪ chicken leg.
Pick up the chicken leg from the fridge
    ↪ shelf ...
```

```
Translate the above intermediate goals into a formal
    ↪ language defined by the following subgoals.
subgoals = ['On(<movable>, <surface>)': the result of
    ↪ picking up <movable> then placing it on <
    ↪ surface>, ...] % 13 in total
Please answer with objects in the respective types: {
    ↪ objects_by_types}
```

b) The VLM response and our next question when asking for *subgoals*.

```
Opened(fridge door),
Picked(chicken leg), ...
```

```
Translate the each of the listed actions in English
    ↪ into a formal language defined by the
    ↪ following primitive actions. Each action in
    ↪ English may correspond to multiple actions:
actions =['pick(<obj>)': it contains one argument.
    ↪ The robot must have an empty hand to pick up
    ↪ an object. ...] % 12 in total
Please answer with objects in the respective types: {
    ↪ objects_by_types}
```

c) The VLM response and our next question when asking for *actions*.

```
open(fridge door),
pick(chicken leg), ...
```

Fig. 4: Conversation Template for querying VLMs and example responses used by VLM-TAMP (ab) and baseline VLM + Motion Planning (ac). The same templates are used during reprompting, with text formatted using **{Purple}** representing updated information.

the VLM and verified by the TAMP planner. The resulting state after achieving a subgoal is the initial state for planning and execution of the next subgoal.

B. Approach

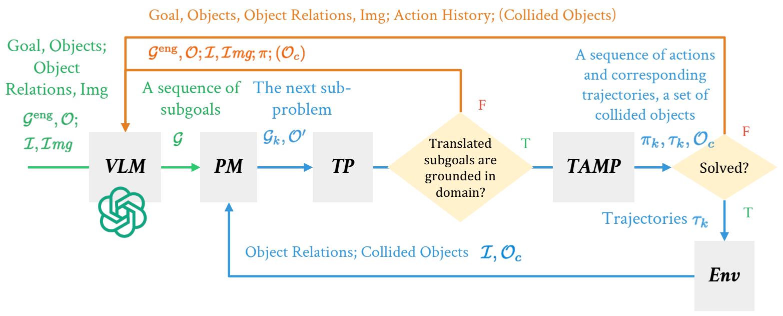
Figure 5 shows an overview of our approach, and Algorithm 1 shows the corresponding pseudocode. There are *two* distinct modes of operation of the system depending on what the VLM is prompted to produce:

- Predicting *subgoals*: the VLM produces subgoals in PDDL format, using on a list of provided predicates
- Predicting *actions*: the VLM produces a sequence of high-level actions, drawing from a set of legal actions.

After checking the semantic consistency of the subgoals or the actions, a problem is constructed for the TAMP planner. In the subgoal setting, the TAMP planner must construct a detailed motion plan that may involve moving multiple objects, opening cabinets, etc. In the action setting, the TAMP planner still needs to convert the high-level action to actual robot commands with continuous parameters for grasps, paths, etc. There may be failures at several points in this process which are addressed by reprompting the VLM.

Algorithm 1 VLM-TAMP

```
Input: $\mathcal{O}, \mathcal{I}, \mathcal{G}^{\text{eng}}, flag_{a(actions)}, N_{\text{reprompt}}, N_{\text{TAMP}}$
1:  $\pi \leftarrow []$
2:  $\mathcal{I}, \mathcal{I}mg \leftarrow$ OBSERVE-STATE-AND-IMAGE()
3:  VLM $\leftarrow$ QUERY-FN-GEN($\mathcal{O}, \mathcal{G}^{\text{eng}}, flag_a, N_{\text{reprompt}}$)
4:  $\mathcal{G} \leftarrow$ VLM-QUERY($\mathcal{I}, \mathcal{I}mg, \pi$)
5:  while LEN($\mathcal{G}$) $> 0$ do
6:      $\mathcal{G}_k \leftarrow \mathcal{G}.pop()$
7:      $\mathcal{O}', \mathcal{O}_c \leftarrow$ REDUCE-OBJECT($\mathcal{O}, \mathcal{I}, \mathcal{G}_k$)
8:      $r \leftarrow$ CHECK-SEMANTICS($\mathcal{G}_k$)
9:      if $r =$ SUCCCESS then
10:         $r, \pi_k, \tau_k, \mathcal{O}_c \leftarrow$ TAMP($\mathcal{O}', \mathcal{I}, \mathcal{G}_k, N_{\text{TAMP}}$)
11:         if $r =$ SUCCESS then
12:             $\pi.extend(\pi_k)$
13:             $\mathcal{I} \leftarrow$ EXECUTE($\tau_k$)
14:             $\mathcal{I}, \mathcal{I}mg \leftarrow$ OBSERVE-STATE-AND-IMAGE()
15:     if $r =$ FAILURE then
16:         $\mathcal{G} \leftarrow$ VLM($\mathcal{I}, \mathcal{I}mg, \pi, \mathcal{O}_c$)
```



Goal, Objects, Object Relations, Img; Action History; (Collided Objects)
Goal, Objects; Object Relations, Img
$\mathcal{G}^{\text{eng}}, \mathcal{O}; \mathcal{I}, \mathcal{I}mg$
$\mathcal{G}^{\text{eng}}, \mathcal{O}; \mathcal{I}, \mathcal{I}mg; \pi; (\mathcal{O}_c)$
A sequence of subgoals
The next sub-problem
A sequence of actions and corresponding trajectories, a set of collided objects
VLM
$\mathcal{G}$
PM
$\mathcal{G}_k, \mathcal{O}'$
TP
Translated subgoals are grounded in domain?
F
T
TAMP
$\pi_k, \tau_k, \mathcal{O}_c$
Solved?
F
T
Trajectories τ_k
Env
Object Relations; Collided Objects $\mathcal{I}, \mathcal{O}_c$

Fig. 5: Our VLM-TAMP Algorithm. ***PM*** means Problem Manager, which formulates the next TAMP sub-problem to solve. ***TP*** means Task Planning, which checks the semantics of subgoals.

C. Using VLM for Subgoal or Action Sequencing

Each VLM query has two phases. First, the VLM takes in an English description of the goal $\mathcal{G}^{eng}$, all the objects in the scene $\mathcal{O}^{eng}$, and their spatial relations $\mathcal{I}^{eng}$. It outputs a sequence of intermediate subgoals $\{\mathcal{G}_i^{eng}\}_{i=1}^K$ or actions $\hat{\pi}^{eng}$ in English, as shown in Figure 4a. Next, it takes a description of the possible subgoals or actions to translate the English answer into. It outputs a sequence of intermediate subgoals $\{\mathcal{G}_i\}_{i=1}^K$ (Figure 4b) or actions $\hat{\pi}$ (Figure 4c) in PDDL format that the TAMP planner consumes.

a) Predicting Subgoals: After predicting subgoals in English $\{\mathcal{G}_i^{eng}\}_{i=1}^K$, the VLM is prompted to translate them into a sequence of goal tuples $\{\mathcal{G}_i\}_{i=1}^K$ (a single grounded predicate), given a pre-defined list of predicates $\mathcal{P}$ along with the types of objects allowed for each position[*]. We currently use a single goal tuple for each subgoal, but this could be extended to a conjunction of goal literals.

b) Predicting Actions: After predicting actions in English, the VLM translates the plan skeleton $\hat{\pi}^{eng}$ into a sequence of actions $\{\hat{a}_i\}_{i=1}^K$, given a pre-defined list of actions $\mathcal{A}$[†], along with English descriptions of the preconditions of applying the action. Figure 4c) lists an example precondition that a "hand should be empty before picking an object". These preconditions are used when prompting the VLM in order to improve reasoning accuracy.

The VLM may produce goal literals or actions with semantic errors, e.g. using objects with the wrong type or objects that don't exist in the world, or using the wrong number of arguments. Each entry in the VLM output is verified with pure symbolic planning in a simplified PDDL formulation that only specifies the discrete arguments for predicates and actions. When one entry is found to be impossible, an error message is fed back to VLM to generate subgoals or actions again. The algorithm returns failure when a maximum number of queries N_{reprompt} is reached.

[*]For example, `On⟨object, surface⟩`, `Sprinkled⟨object, object⟩`, `Opened⟨joint⟩`, and `TurnedOn⟨joint⟩`.

[†]For example, `pick⟨object⟩`, `place⟨object, region⟩`, `sprinkle⟨object, region⟩`, and `open⟨joint⟩`.

D. Using TAMP to Refine Subgoals or Action Sequences

Given a sequence of subgoal or actions, the TAMP system refines each one in order, generating a sequence of grounded action plans and corresponding motion trajectories.

a) TAMP problems: We represent TAMP problems using an extension of the Planning Domain Definition Language (PDDL) [20], a logic-based action language, that supports planning with continuous values [21]. We define a TAMP *domain* $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ by a set of *predicates* $\mathcal{P}$ and *actions* $\mathcal{A}$. *Predicates* and *actions* can be represented as tuples consisting of a name and a list of typed arguments. The arguments may be (1) discrete, such as object and part names, or (2) continuous, such as object poses, object grasps, robot configurations, object joint angles, and robot trajectories.

We define a TAMP *problem* $\langle \mathcal{O}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ using a set of *objects* $\mathcal{O}$ (constants specific to the problem), a set of *initial literals* $\mathcal{I}$, a conjunctive set of *goal literals* $\mathcal{G}$, and the planning domain $\mathcal{D}$. A *literal* is a predicate with an assignment of values to its arguments. The set of initial literals defines a state of the world. Each *grounded action* defines a deterministic transition of the world state.

A *solution* π is a finite sequence of grounded action instances that, when sequentially applied to the initial state $\mathcal{I}$, produces a terminal state where the goal literals $\mathcal{G}$ all hold. A *plan skeleton* $\hat{\pi}$ is a sequence of partially grounded actions, where the discrete parameters are bound but the continuous parameters are unbound.

b) Planning for Subgoals: For each goal literal $\mathcal{G}_k$, a *subproblem* $\langle \mathcal{O}_k, \mathcal{I}_k, \mathcal{G}_k \rangle$ is generated, which TAMP takes in then returns solution $\langle \pi_k, \tau_k, \mathcal{O}_c \rangle$ if successful and $\langle r = \text{FAILURE}, \mathcal{O}_c \rangle$. $\mathcal{I}_k$ is the current state and involves all objects. $\mathcal{O}_k$ is a small subset of objects to be considered for grounding predicates and actions during planning. Reducing the universe of objects reduces the size of the action space (and thus branching factor) of this subproblem. First, planning is attempted with only the objects that are mentioned in the goal or apart of the robot state (i.e. currently grasped). During planning, the system records $\mathcal{O}_c$, the movable or articulated objects that collide with the robot in some future state (e.g. while sampling object grasps, solving inverse kinematics, or planning motion). If planning with the goal-relevant objects fails, the often sparse subset of objects that contributed to collisions is added to the set of objects and the TAMP planner is called again. The subgoal is declared unreachable after N_{TAMP} unsuccessful calls to the TAMP planner. When planning is successful, the system execute the motion trajectories τ_k and extend grounded plan π_k to the whole history of actions executed. It then observe the environment and obtain updated $\mathcal{I}_{k+1}$.

In this formulation, omitting objects from $\mathcal{O}$ generally reduces the set of changes the robot can make in the domain but does not, for example, remove objects from the world that might cause collisions. Limiting the relevant objects is most effective when the goal predicates are designed to be the effects of actions in the domain. For example, `Heat`$\langle$`object`$\rangle$ is uninformative as it leaves out arguments like the heating surface and appliance handle; planning without those objects will fail. To avoid this, all predicates we ask the VLM to generate are on the spatial and motion level, which specify the directly relevant entities, such as `PlaceOn`$\langle$`object, surface`$\rangle$ and `TurnHandle`$\langle$`joint`$\rangle$.

c) Refining Actions: Given a partially grounded action a_k, which includes only discrete parameters such as objects and robot arms, the system first finds its symbolic effects and uses them as $\mathcal{G}_k$. Given a subproblem $\langle \mathcal{O}_k, \mathcal{I}_k, \mathcal{G}_k \rangle$, the TAMP planner is called in a manner where it is constrained to use the partial plan skeleton $\hat{\pi}_k = [a_k]$. This forces the planner to find a plan that uses that action; however, the planner still needs to refine the action by choosing values for its continuous parameters, and the planner may need to add additional actions, for example a base motion to reach the target action. This variant also uses reduced objects and is allowed multiple trials, as in general subgoal planning.

E. VLM Replanning after TAMP Failure

When the TAMP planner fails to solve a subgoal or action after N_{TAMP} trials, the VLM is prompted again as described in Algorithm 5 and Figure 5. It takes in an updated description of the scene, the sequence of actions already executed, and collision objects detected during failed runs. The VLM reprompting is carried out at most N_{reprompt} times before the system declares failure. Our approach assumes that there are no long-term low-level dependencies among the subgoals that can *only* be addressed by making one long detailed plan for the entire problem. In other words, as long as it is not possible to become irreversibly stuck after taking some actions, we can factor the whole problem into smaller ones, counting on plans for the later subgoals to resolve geometric difficulties caused by the previous ones.

IV. Experiments

We run experiments to answer the following questions:

1) Which mode of VLM sequencing gives better task completion performance: predicting subgoals or actions?
2) What's the extent to which reprompting improves performance? Does increasing the compute budgets increase the number of problems VLM-TAMP can solve?

A. Baselines and Ablations

We compare two approaches, each with three variants that allow VLM reprompting for $N_{\text{reprompt}} \in \{0, 1, 2\}$ times:

- *VLM Subgoal Sequencing + TAMP refinement (ours)* uses VLMs to generate a sequence of subgoals which a TAMP planner solves.
- *VLM Action Sequencing + limited TAMP refinement* uses VLMs to generate actions which a TAMP planner refines, but where no additional actions can be added other than moving the robot's base. This baseline is representative of [12], [13], [14], [15].

B. Task Suite and Robot Embodiment

We consider the task of making chicken soup in a kitchen with 5 movable objects (e.g. food and seasoning), 8 surfaces (e.g. counter, stove burners, sink, pot), 2 spaces (enclosed in doors or a drawer), and 6 articulated objects (e.g. doors, knobs). In the *Easy* case, all doors are initially open and the pot lid is on the counter. In the *More Obstacles* case, all doors are initially closed and the pot lid is covering the pot body. To ensure that generated problems are feasible, we set objects to fixed initial poses and doors to slightly different open positions. Note that the subsequent object poses and joint positions are randomly sampled during planning, so the induced subproblems vary drastically after the initial state.

The robot can change the pose of objects via pick and place actions as well as change the joint positions of articulated joints through pulling, pushing, and rotating its wrist. In the *Single-Arm* case, the robot is allowed to use only its left arm, while the *Dual-Arm* case allows it to use both arms. Note that this kitchen environment is quite challenging, e.g., the sink is small and the fridge door and faucet allow a very limited range for the base and arm to position the pot in the sink without collision.

Altogether, we compare six methods on four variations of the task. Each method is run for 30 random trials. For each task, we ask the VLM for five plans and we sample randomly from those for each of the 30 repetitions. We measure the following performance metrics:

- *Task Success*. The algorithm successfully refines all subgoals or actions generated by the VLM into collision-free motion trajectories.
- *Task Completion Percentage*. The number of subproblems solved out of all subproblems generated by the VLM and the Problem Manager, including completed problems and unfinished problems from the last query.

C. Implementation Details

We use `gpt-4o-mini` as our VLM [2], with temperature = 0.2. For TAMP, we use the diverse planning mode of PDDLStream (as in [22] but without the task plan feasibility predictor) with a maximum number of considered plan skeletons equal to 12. For action refinement, we use the same planner but constrain it to include only the predicted action and moving the base in the plan skeleton. We use $N_{\text{TAMP}} = 3$ as the number of planning runs allowed for each subproblem, with increasing number of world objects

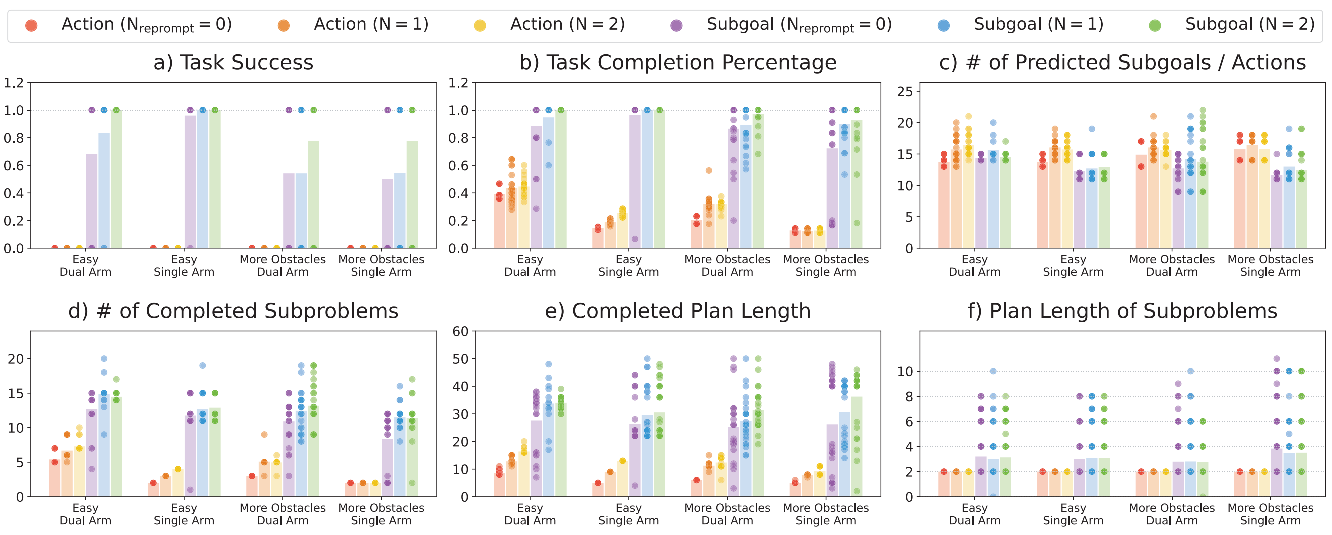


Fig. 6: Our experimental results show that that 1) predicting subgoals (VLM-TAMP) outperforms predicting actions, 2) reprompting helps when subgoals (VLM-TAMP) as number of reprompt tries increases but not when predicting actions. All six methods are run for 30 random trials on four problem difficulties, with increasing numbers controllable robot arms and manipulable obstacles.

included in planning. In other words, if the planner fails in the first two runs due to a sampling failure or not including a sufficient set of world objects but succeeds the last time, we still count the subproblem as a success.

D. Results

a) Predicting Subgoals Significantly Outperforms Predicting Actions: Compared to to using the VLM for predicting actions that has zero success rate across all variations of the problems, VLM-TAMP succeeds 50 to 100% of the time as shown in Figure 6a). The full TAMP planner successfully fills in the actions to resolve geometric infeasibility by moving articulated or movable obstacles (using on average 1 to 2 actions as shown in Figure 6f). When comparing task completion percentage, it also significantly outperforms baselines (72 to 100% versus 15 to 45%). More subproblems are solved (Figure 6d) out of similar length of subproblem sequences proposed (Figure 6c) and the resulting plans have more actions (Figure 6e). See the supplementary video for example execution traces generated by VLM-TAMP.

VLM-TAMP visibly benefit from VLM reprompting, with task success increasing by 47 to 55% on the harder problems, as shown in Figure 6a. As the number of reprompting runs increases, the performance of the subgoal variants increases as it gives TAMP more tries to solve the geometrically difficult subproblems and it may inject intermediate subgoals that make it easier to achieve the failed subgoal. As seen from individual run statistics, reprompting reduced the variance in the completion percentage. In comparison, reprompting didn't help the variant that generates actions, even though the prompt includes action history, collision summary, and description of which robot arms are holding which objects. This shows that the VLM cannot be relied upon to consider long-horizon history, complex world description, and geometric infeasibility when predicting actions.

V. Failure Analysis

We overview several system failure modes. In particular, VLMs are fairly experimental and routinely make inaccurate predictions, although we expect them to improve over time.

A. VLM Failures

a) Subgoal/Action translation to PDDL: The second phase of VLM query process asks the VLM to translate English answers to formal PDDL language of goal or action tuples. The VLM may 1) miss certain subgoals or actions when one sense contains multiple subgoals or actions, and 2) generate a tuple with the wrong number of arguments or misspell the object name.

b) Infeasible actions: The subproblem may not be solvable because there exist movable or articulated objects (e.g., robot is asked to turn on the stove but the pot is placed in a location that blocks any path to grasp the knob handle), or because the precondition is not met (e.g. robot asked to close a door but the robot's only arm is holding an object)[‡].

B. TAMP Failures

a) Didn't find a feasible plan skeleton: The planner is allowed to try 12 plan skeletons in order of increasing task length. For a subproblem that involves six planning objects and two obstacles to clear out, the correct plan skeleton may not be ranked in the first 12 allowed.

b) Found feasible plan skeletons but failed to refine them within the compute budget: Each low-level samplers (e.g., pose sampler, grasp sampler, inverse-kinematic solver)

[‡]Although the VLM is given a rule in the prompt that says "`You must have at least one empty hand before you can pick up an object or open or close a joint`", it still sometimes generates an English goal "`Close the cabinet left door with the hand that was holding the salt shaker`" and then translates it into "`closed-door(cabinet left door)`" which is infeasible.

is allowed a limited number of attempts. During refinement, they are attempted in sequence and need all be successful in order to refine the plan skeleton.

VI. Discussion

We present VLM-TAMP, a system for solving long-horizon manipulation planning problems by marrying the strengths of VLM's language understanding and commonsense reasoning abilities, to the strengths of TAMP's ability to find feasible task skeletons and generate collision-free trajectories that respect all geometric constraints. The combined system effectively overcomes the shortcomings of 1) VLM's lack of geometric and long-horizon reasoning abilities by letting TAMP fill in required actions and parameter values and 2) TAMP's explosive computational complexity by leveraging a VLM to break down the problem along with allowing it to correct for failures on a short horizon via reprompting.

Future work involves training dexterous dual-arm manipulation policies to achieve subgoals given visual inputs.

Acknowledgment

We gratefully acknowledge support from AI Singapore AISG2-RP-2020-016; from NSF grant 2214177; from AFOSR grant FA9550-22-1-0249; from ONR MURI grant N00014-22-1-2740; from ARO grant W911NF-23-1-0034; from the MIT Quest for Intelligence; and from the Artificial Intelligence Institute. This work was partially conducted during an internship at NVIDIA Research.

References

[1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated Task and Motion Planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, 2021.

[4] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "Llm+ p: Empowering large language models with optimal planning proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[5] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Task and motion planning with large language models for object rearrangement," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 2086–2092.

[6] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, "Autotamp: Autoregressive task and motion planning with llms as translators and checkers," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 6695–6702.

[7] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[8] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman *et al.*, "Grounded decoding: Guiding text generation with grounded models for robot control," *arXiv preprint arXiv:2303.00855*, 2023.

[9] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.

[10] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[11] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *arXiv preprint arXiv:2307.05973*, 2023.

[12] X. Zhang, Y. Ding, S. Amiri, H. Yang, A. Kaminski, C. Esselink, and S. Zhang, "Grounding classical task planners via vision-language models," *arXiv preprint arXiv:2304.08587*, 2023.

[13] M. Skreta, Z. Zhou, J. L. Yuan, K. Darvish, A. Aspuru-Guzik, and A. Garg, "Replan: Robotic replanning with perception and language models," *arXiv preprint arXiv:2401.04157*, 2024.

[14] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu, "Llmˆ 3: Large language model-based task and motion planning with motion failure reasoning," *arXiv preprint arXiv:2403.11552*, 2024.

[15] F. Joublin, A. Ceravola, P. Smirnov, F. Ocker, J. Deigmoeller, A. Belardinelli, C. Wang, S. Hasler, D. Tanneberg, and M. Gienger, "Copal: Corrective planning of robot actions with large language models," *arXiv preprint arXiv:2310.07263*, 2023.

[16] Y. Guo, Y.-J. Wang, L. Zha, Z. Jiang, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," *arXiv preprint arXiv:2307.00329*, 2023.

[17] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," *Autonomous Robots*, vol. 47, no. 8, pp. 1345–1365, 2023.

[18] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, "Translating natural language to planning goals with large-language models," *arXiv preprint arXiv:2302.05128*, 2023.

[19] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016. [Online]. Available: http://pybullet.org

[20] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL: The Planning Domain Definition Language," Yale Center for Computational Vision and Control, Tech. Rep., 1998.

[21] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers," in *ICAPS*, 2020.

[22] Z. Yang, C. R. Garrett, T. Lozano-Perez, L. Kaelbling, and D. Fox, "Sequence-Based Plan Feasibility Prediction for Efficient Task and Motion Planning," in *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.