Bootstrapping Object-level Planning with Large Language Models

David Paulius[1], **Alejandro Agostini**[2], **Benedict Quartey**[1], **George Konidaris**[1]

[1]Brown University, Providence, RI, USA
[2]University of Innsbruck, Innsbruck, Austria
{dpaulius,gdk}@cs.brown.edu, alejandro.agostini@uibk.ac.at, benedict_quartey@brown.edu

Abstract

We introduce a new method that extracts knowledge from a large language model (LLM) to produce *object-level plans*, which describe high-level changes to object state, and uses them to bootstrap task and motion planning (TAMP) in a hierarchical manner. Existing works use LLMs to either directly output task plans or to generate goals in representations like PDDL. However, these methods fall short because they either rely on the LLM to do the actual planning or output a hard-to-satisfy goal. Our approach instead extracts knowledge from a LLM in the form of plan schemas as an object-level representation called functional object-oriented networks (FOON), from which we automatically generate PDDL subgoals. Our experiments demonstrate how our method's performance markedly exceeds alternative planning strategies across several tasks in simulation.

Introduction

The advent of *large language models* (LLMs) has led to a plethora of work that exploits their capabilities for a variety of tasks, including planning for robotics (Ichter et al. 2023; Driess et al. 2023) and embodied agents (Huang et al. 2022; Raman et al. 2024) using LLMs. Language models encode domain knowledge about the world, which is useful to decision making. These approaches use LLMs as either: 1) a task planner (Ichter et al. 2023; Singh et al. 2023; Driess et al. 2023), or 2) a task goal generator (Liu et al. 2023; Xie et al. 2023; Liu et al. 2024). As a task planner, a LLM is informed of the task and scene and directly outputs a complete plan, forgoing the use of automated planning (Ghallab, Nau, and Traverso 2016) with off-the-shelf planners. Plan actions outputted by a LLM are then grounded to action policies or primitives. As a task goal generator, a LLM generates planning definitions in the form of representations like PDDL (McDermott et al. 1998) (short for *Planning Domain Definition Language*); this type of approach is often situated with task and motion planning (TAMP) (Garrett et al. 2021).

However, existing work in these categories fail to handle complex, goal-oriented tasks in several key aspects. On one hand, positing the LLM as a task planner deprives such methods of guarantees promised by classical planning (viz. optimality and completeness). Recent work has also called to question whether LLMs can effectively plan (Valmeekam et al. 2023). On the other hand, using the LLM as a task description generator will fail to generate plan specifications that are guaranteed to work due to the LLM's lack of embodiment. For instance, it may be difficult for the LLM to generate accurate PDDL definitions simply from a language description of the robot's environment.

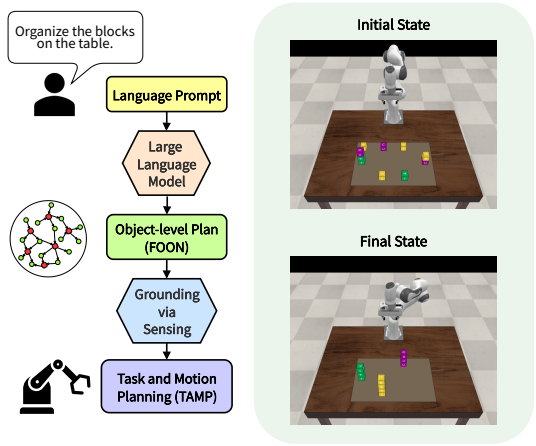


Figure 1: Our approach prompts a LLM for object-level information, from which we construct an object-level plan (as FOON). This plan schema *bootstraps* task- and motion-level planning (TAMP) with PDDL subgoals.

It is natural to explore language models for planning, as they contain domain knowledge useful to planning while often output useful steps. Similarly, LLMs are useful as goal generators because one can still exploit off-the-shelf planners. We propose an approach where the LLM is used to generate *partial goal schemas at the object level*, which can then be used to generate PDDL subgoals. Such an approach inherits the desirable commonsense planning knowledge of the LLM while still supporting sound and complete task-level planning. The object (as opposed to task) level is the level at which natural language is most appropriate, and at which most knowledge is contained and expressed (Kroemer, Niekum, and Konidaris 2021; Paulius 2022). While task-level planning focuses on task (action and motion) constraints for task execution, *object-level planning* focuses on

how objects are combined to produce new objects without committing to *how* these effects will be resolved until runtime.

We propose a modular approach that distills domain knowledge from a LLM to generate *object-level plans* (Paulius 2022), which can then be used to bootstrap hierarchical planning. We situate object-level planning as an interface between human language and TAMP and exploit an object-level representation (OLR) called the *functional object-oriented network* (FOON) (Paulius et al. 2016). Recent work has shown how object-level knowledge in FOON can automatically generate PDDL subgoals (Paulius*, Agostini*, and Lee 2023); however, this assumes that partial plan specifications already exist as a FOON. In this work, we exploit the capabilities of LLMs for generating FOONs which, when coupled with LLMs, overcomes the inability of LLMs to directly output feasible task plans while exploiting the higher, object-level nature of LLM output and language on a whole.

The contributions of our work are as follows: 1) we introduce a modular planning approach (Figure 1) that interfaces with a LLM to generate natural language instructions, from which we transform into an OLR (e.g., FOON) for hierarchical planning; 2) we show how object-level information can be distilled directly from an LLM and then used to generate planning definitions as PDDL, improving the feasibility of generated plans; and 3) we showcase markedly better performance than alternative LLM-based methods.

Background

Large Language Models: A large language model (LLM) is a complex neural network model trained via self-supervised learning and self-attention (the latter revolutionized by the transformer architecture (Vaswani et al. 2017)). LLMs have shown remarkable performance in natural language processing (NLP) and generation tasks. LLM variants such as GPT (Brown et al. 2020) and LLaMA (Touvron et al. 2023) are trained on large corpora of text gathered from the internet and fine-tuned using RLHF (reinforcement learning from human feedback). For this reason, a LLM can be thought of as a "compressed" representation of domain knowledge from the web (Chiang 2023), which is why we aim to exploit these models to inform planning. This work uses OpenAI's Chat-GPT (OpenAI 2024).

Task and Motion Planning: The aim of task and motion planning (TAMP) is to integrate higher-level symbolic *task planning* with lower-level *motion planning* for enabling robots to solve complex long-horizon tasks (Garrett et al. 2021). At the lower level, *motion planning* finds collision-free robot motions or trajectories that will resolve a task. However, typical robot tasks are too complex to be solved via motion-level planning alone due to the high-dimensional nature of the robot's task and configuration space. For this reason, *task planning* is necessary as an added layer to reason over an abstraction of a robot's actions and its environment for simplifying motion planning. Task planning assumes a state description $\mathcal{S}$ using logical predicates, which are `true` or `false` depending on whether or not the robot observes them. Starting from an initial state $s_0 \in \mathcal{S}$, task planning finds an action sequence $a \in \mathcal{A}$ that achieves a goal g as a task plan $\mathcal{P} = \{a_1, ..., a_n\}$. An action a refers to a robot-executable skill or policy; our work assumes access to a repertoire of skills, which we denote by $\mathcal{A}$, that are defined as planning operators in PDDL (McDermott et al. 1998).

Object-level Planning with Language Models

There exists a disconnect between language and TAMP, which makes it unsuitable for generalization across tasks and settings. Yet, existing works use LLMs either as planners or task description generators for task execution; these approaches fall short due to their overestimation of a LLM's ability to reason about task- and motion-level constraints. It is impractical to provide the entire context of a task setting to a LLM and expect it to handle all the reasoning about a robot's embodiment (e.g., where objects are located, in what poses they are, what type of gripper the robot has, etc.) in order to generate adequate planning definitions or feasible task plans.

Instead, the strong suit of language models lies in its ability to provide plan sketches or subgoals that are useful to decision making at both task and motion levels. This is because language models can express task-relevant knowledge in a generic yet informative way using natural language. Imagine your typical cooking recipe, for instance: a recipe provides a sketch of object interactions agnostic to the state of your kitchen or the recipe writer's kitchen. It also does not provide details on how actions should be executed (e.g., which hand should you use, how should you grasp an object, etc.). What a recipe expressed in natural language may provide however is an idea of the types of actions and object-object interactions that are necessary to complete a task. Rather, the exact details of task- and motion-level execution are resolved at run time.

It is for this reason why we adopt an *object-level planning* approach to bootstrap task and motion planning. We generate object-level plan sketches, which provide task-level subgoals that naturally interface language and decision making, using a LLM. Briefly, given a prompt for a task to be executed by a robot, our approach (Figure 2) uses a LLM to generate a sequence of natural language instructions, which is then transformed into an *object-level plan* (OLP) represented as a FOON. It is then through task planning where properties relevant to the robot, e.g., robot's end-effector and object poses) are used to find a task-aware plan which we then execute via TAMP. We perform task-level planning by transforming each OLP action into PDDL definitions to find task plan segments.

Object-level Planning

In this work, we add another level of planning layered above TAMP called object-level planning, which reasons solely over object interactions (Paulius 2022). We situate object-level planning with the functional object-oriented network (FOON): a knowledge graph representation describing object-action relations (Paulius et al. 2016). Formally, a FOON $\mathcal{G} = \{\mathcal{O}, \mathcal{M}, \mathcal{E}\}$ is a bipartite graph with

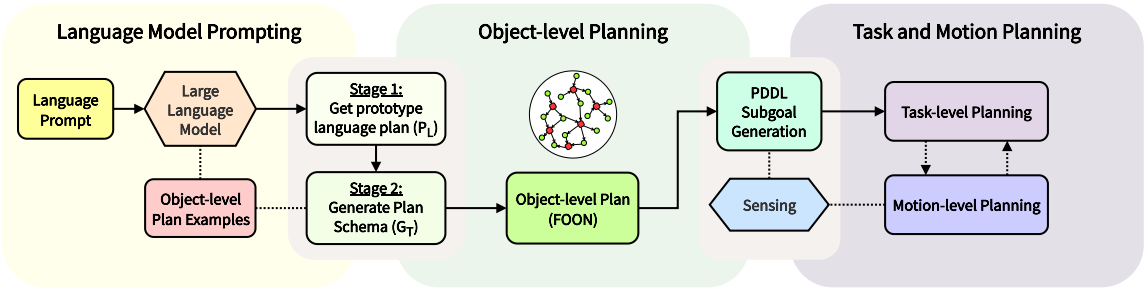


Figure 2: Our approach interfaces with a language model to generate object-level plans (as FOON graphs) for bootstrapping task and motion planning. We generate task-level subgoals as PDDL subgoal definitions by grounding object-level subgoals to a robot's environment; with these task-level definitions, we perform task planning to obtain task plan segments per object-level action, which we execute using motion-level planning, improving upon prior work (Paulius*, Agostini*, and Lee 2023).

object nodes ($o \in \mathcal{O}$) and motion nodes ($m \in \mathcal{M}$) connected via directed edges ($e \in \mathcal{E}$), which reflect the change of an object's state as it is manipulated via a corresponding action. An object $o = (o_t, o_s, o_{\mathcal{I}})$ is defined as a tuple with the following attributes: its object type or name (o_t), its state (o_s), and, if applicable, its object composition ($o_{\mathcal{I}} = \{o_{t_1}, o_{t_2}, ..., o_{t_n}\}$, where $n = |\mathcal{I}|$). A motion node $m = (m_t)$ is defined by an action verb or type (m_t). A FOON describes object-state transitions via *functional units* ($\mathcal{FU} = \{\mathcal{O}_{in}, \mathcal{O}_{out}, \tilde{m}\}$) at a level closer to human language. A functional unit defines preconditions and effects of executing an action ($\tilde{m}$), where a set of input nodes ($\mathcal{O}_{in}$) are required to produce a new set of output object nodes ($\mathcal{O}_{out}$). We illustrate an example of a functional unit as Figure 3, which describes an action for a block stacking task. Foundation models naturally interface with object-level representations like FOON due to its similarity to human language, which in turn allows it to interface with tools that combine vision and language (Brown et al. 2020; Zhang et al. 2024).

LLM Prompting to Object-level Plan

Our goal is to extract attributes for object-state transitions before and after each action is performed (i.e., preconditions and effects) to construct an OLP describing task subgoals. We instantiate object-level planning with FOONs. Our approach constructs a FOON $\mathcal{G}_{\mathcal{T}}$, where $\mathcal{T}$ is a task given in natural language (such as *"Make a tower of two red blocks"*—see Figure 3) via a two-stage process. In addition to the task instruction $\mathcal{T}$, we include a language description of objects in the scene (from which the LLM will determine which ones are relevant to the task) as well as a set of example object-level plans (as FOONs) $\mathcal{X}_{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$ for reference.

The first stage involves prompting a LLM for a plan sketch comprised of natural language instructions denoted by $\mathcal{P}_L = \{\xi_1, \xi_2, ..., \xi_n\}$, where ξ_i refers to an instruction as text. As an example in Figure 3, given a task and available objects (without *any* context about their present configuration), we expect text instructions $\mathcal{P}_L$ that solely mention red blocks for the task *"Make a tower of 2 red blocks."* During this step, we transform the top-k most similar FOON in $\mathcal{X}_{\mathcal{G}}$ into example plan sketches, from which the LLM selects the

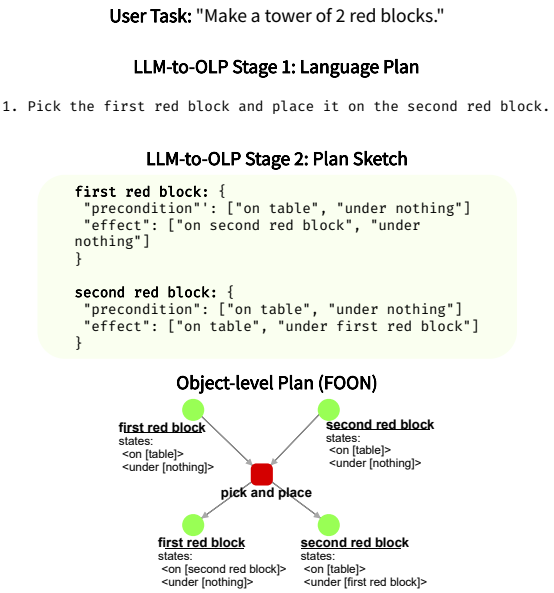


Figure 3: Illustration of how a user task specified in natural language is transformed into an object-level plan (OLP) as a FOON via LLM prompting.

one closest to the new task to use as reference (denoted by $\hat{\mathcal{G}}$). We identify the top k candidates using cosine similarity between text embeddings of the task prompt $\mathcal{T}$ and the set of instructions for a given reference $\hat{\mathcal{G}} \in \mathcal{X}_{\mathcal{G}}$. An example sketch may describe how three generic blocks (regardless of type) can be stacked into a tower. In the second stage, the LLM must reason about each instruction $\xi_i \in \mathcal{P}_L$ to generate an OLP for the novel task. We prompt the LLM to reason about state changes of task-relevant objects, specifically geometric relations for task-level planning (Section). In the previous example, we expect output with state descriptions such as *"first red block on second red block," "first red block under nothing," "second red block under first red block,"* and *"second red block on table"* (Figure 3). We assist the LLM by providing $\hat{\mathcal{G}}$ in the prompt, with which it must gen-

erate a new FOON $\mathcal{G}_\mathcal{T}$ for the novel task. Inspired by prior work on code writing for robots (Liang et al. 2023), we codify $\hat{\mathcal{G}}$ as a JSON. The LLM then outputs a codified OLP $\tilde{\mathcal{G}}_\mathcal{T}$, capturing each instruction $\xi_i \in \mathcal{P}_L$. Finally, each action in $\tilde{\mathcal{G}}_\mathcal{T}$ forms a functional unit $\mathcal{FU}_i \in \mathcal{G}_\mathcal{T}$.

Bridging to Task and Motion Planning

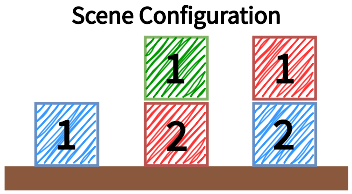
We generate a plan schema $\mathcal{G}_\mathcal{T}$, with which we can solve a task $\mathcal{T}$ given in natural language. However, this schema is too abstract to be executed in its present form, and it must be grounded to the robot's embodiment and environment (Paulius*, Agostini*, and Lee 2023). Therefore, we use $\mathcal{G}_\mathcal{T}$ to *bootstrap* TAMP by providing PDDL subgoals. This is done through a hierarchical approach that automatically transforms each object-level action in $\mathcal{G}_\mathcal{T}$ into PDDL problem definitions and searches for a robot-executable plan given a predefined set of robot skills or operators.

Object-level to Task-level Planning

The aim of task-level planning is to find a robot-executable *task plan* $\mathcal{P}_\mu$ solving a task $\mathcal{T}$. In more detail, a task plan is comprised of a sequence of smaller plan segments for each functional unit, i.e., $\mathcal{P}_\mu = \{\tilde{\mathcal{P}}_{\mu_1}, \tilde{\mathcal{P}}_{\mu_2}, ..., \tilde{\mathcal{P}}_{\mu_n}\}$, where $\tilde{\mathcal{P}}_{\mu_i} = \{a_{\mu_1}, a_{\mu_2}, ..., a_{\mu_m}\}$ denotes a plan segment achieving the subgoals described by a functional unit $\mathcal{FU}_i$ and a_{μ_i} refers to the i-th step corresponding to a parameterized skill in $\mathcal{A}$.

PDDL solvers require two components: 1) a domain definition, and 2) a problem definition (McDermott et al. 1998). A domain definition provides details on what actions can be taken by a robot or agent as well as possible object types, while a problem definition captures the initial state of the robot and its environment (s_μ) as well as the goal state (g_μ) as logical predicates. In this work, we assume that we have a predefined domain definition with planning operators corresponding to a repertoire of parameterized, robot-executable skills $\mathcal{A}$. However, for problem definition generation, both s_μ and g_μ are adapted from a functional unit $\mathcal{FU} \in \mathcal{G}_\mathcal{T}$: predicates are constructed based on object-state pairs in $\mathcal{FU}$. In other words, transforming a FOON into PDDL requires the mapping of object properties defined by each object o to predicates (where $o \in \mathcal{O}_{in} \cup \mathcal{O}_{out}$).

Object-centered Predicates: We use object-centered predicates (Agostini et al. 2020; Paulius*, Agostini*, and Lee 2023), which describe constraints for collision-free action and motion. They are written as `(⟨rel⟩ ⟨?obj_1⟩ ⟨?obj_2⟩)`, where `⟨rel⟩` refers to a geometric relation using the spatial adpositions *in*, *on*, or *under*, while `⟨?obj_1⟩` and `⟨?obj_2⟩` refer to objects described by a given predicate; These relations are described from the reference frame of each object, which permits propagating motion constraints at task planning for the generation of feasible plans (Agostini and Piater 2023). For example, the predicate `(on block1 block2)` denotes that `block2` is lying on top part of `block1`. We also use a virtual object *air* to describe free space in or on objects, which is important for collision-free picking (e.g., `(on block1 air)`—nothing is on a block, making it free for grasping).



Task-level State (after Grounding)

```
(on table blue_block_1)
(on table blue_block_2)
(on table red_block_2)
(on blue_block_1 air)
(on blue_block_2 red_block_1)
(on red_block_1 air)
(on red_block_2 green_block_1)
(on green_block_1 air)
```

Figure 4: Example of task-level grounding for an object-level plan (Figure 3). This state description is compatible with PDDL definitions (like Figure 5).

Grounding: Each subgoal in an OLP (i.e., functional unit in FOON) must be grounded to the robot's environment for effective task-level planning. For starters, object-level aliases must be linked to object references at the task level. This can be likened to how we as humans use recipes: a recipe refers to ingredients with words, but we must resolve their references to object instances around us when completing a recipe. This work assumes that there exists an exact mapping of objects described in an OLP to those existing in the environment, and we prompt the LLM to map each alias to an instance. For example, if we have two red blocks as objects in FOON, an LLM will map them to object instances `red_block_1` and `red_block_2` (Figure 4). Once these are resolved, we can then map object-state pairs described in FOON to task-level predicates: we use object poses (both position and orientation) and bounding boxes to derive object-centered predicates for each object o in $\mathcal{G}_\mathcal{T}$ using the mechanism from prior work (Agostini and Piater 2023).

Task-level to Motion-level Planning

With each plan segment $\tilde{\mathcal{P}}_\mu \in \mathcal{P}_\mu$, a robot can then execute a sequence of actions that will resolve object-level subgoals. We use motion-level planning to find collision-free robot movements that will resolve the effects of a robot's skills. This work considers picking and placing actions (Figure 5). For the pick action (Figure 5a), we generate a trajectory that moves the robot's end-effector from its initial position to a target object, while the place action (Figure 5b) moves the robot's end-effector grasping an object from its initial pose to a position above a target surface or object. Initial and final poses of the hand for these actions can be directly obtained from object-centered hand-object relations encoded in the precondition and effects of their corresponding planning operators using rotation and translation geometric transformations (Agostini and Piater 2023).

Table 1: Experimental results for several block stacking tasks across 10 trials per setting and block counts

Task Setting	Planning Approach	% Plan Complete	% Success	Avg. Plan Time (s)	Avg. Tokens	Avg. Plan Length
Tower	OLP	**86.00%**	88.37%	$\mathbf{0.0043 \pm 0.0021}$	2406.38 ± 335.0091	10.2791 ± 4.5687
	LLM-Planner	44.00%	77.27%	12.0486 ± 6.3784	$\mathbf{744.94 \pm 120.9533}$	10.2791 ± 4.5687
	LLM+P	18.00%	**94%**	0.0346 ± 0.0312	1656.42 ± 170.2912	$\mathbf{8.5556 \pm 3.9291}$
	DELTA	**86.00%**	69.77%	0.0067 ± 0.01206	$4871.88438.2610$	9.0233 ± 4.7883
Spelling	OLP	**80.00%**	**77.50%**	0.02715 ± 0.0828	2588.66 ± 379.1376	$\mathbf{8.45 \pm 3.4932}$
	LLM-Planner	22.00%	44.44%	7.5734 ± 2.4650	$\mathbf{754.06 \pm 102.7516}$	9.7778 ± 3.3529
	LLM+P	30.00%	46.00%	0.0268 ± 0.0266	1671.26 ± 189.9115	9.0435 ± 4.0393
	DELTA	78.00%	64.10%	$\mathbf{0.0075 \pm 0.0061}$	4836.72 ± 475.0059	10.5641 ± 5.5998
Organize	OLP	**81.43%**	94.74%	$\mathbf{0.0080 \pm 0.0053}$	3051.5 ± 499.4818	15.3684 ± 7.1630
	LLM-Planner	35.71%	64.00%	24.1510 ± 15.9711	$\mathbf{885.0571 \pm 126.3086}$	$\mathbf{8.40 \pm 2.2361}$
	LLM+P	37.14%	**100.00%**	0.0538 ± 0.1139	1891.3286 ± 212.4083	11.3077 ± 4.1547
	DELTA	67.14%	80.85%	0.0139 ± 0.0312	5329.9571 ± 470.9973	13.8298 ± 6.4042

```
(:action pick
 :parameters (
  ?obj - object
  ?surface - object)
 :precondition (and
  ; collision-free constraints
  (in hand air)
  (on ?obj air)
  ; object is on surface
  (on ?surface ?obj)
  (under ?obj ?surface))
 :effect (and
  ; hand contains object
  (in hand ?obj)
  (not (in hand air))
  ; object is grasped
  (on ?obj hand)
  (under ?obj air)
  (not (on ?obj air))
  ; nothing is on surface
  (on ?surface air)
  (not (under ?obj ?surface))
  (not (on ?surface ?obj)) )
)
```

(a) Pick Action

```
(:action place
 :parameters (
  ?obj - object
  ?surface - object)
 :precondition (and
  ; hand contains object
  (in hand ?obj)
  (on ?obj hand)
  ; collision-free constraints
  (on ?surface air)
  (under ?obj air))
 :effect (and
  ; hand no longer contains object
  (in hand air)
  (not (in hand ?obj))
  ; nothing is on object
  (on ?obj air)
  (not (on ?obj hand))
  ; object is on surface
  (on ?surface ?obj)
  (not (on ?surface air))
  (under ?obj ?surface)
  (not (under ?obj air)) )
)
```

(b) Place Action

Figure 5: Planning operators for *pick* and *place* actions using object-centered predicates (Agostini et al. 2020) and executable via motion-level planning (Section).

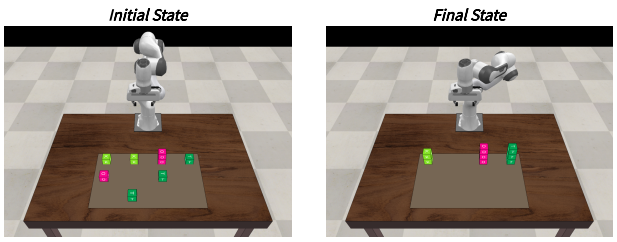


Figure 6: Example of the initial and final states for the *organizing table* task. The *tower building* and *spelling* tasks involve stacking similar lettered blocks.

Evaluation

We evaluate the flexibility of our hierarchical planning approach, which exploits a LLM for extracting OLPs, which are transformed into task- and motion-level problems. Particularly, we highlight that a LLM cannot reliably produce PDDL definitions, and are unable to reliably task plan; however, we can prompt a LLM for object-level details useful to construct PDDL subgoals similar to prior work (Paulius*, Agostini*, and Lee 2023). We compare our approach (OLP in Table 1) with alternative methods on simulated experiments across several tasks.

Experimental Setup

We perform experiments in a simulated table-top environment in CoppeliaSim (Rohmer, Singh, and Freese 2013) with a Franka Emika Panda robot affixed to a table upon which blocks are randomly initialized. Given a task in natural language, the robot must perform a series of *pick* and *place* actions (defined as Figure 5) fulfilling the task. We assume that the state of the environment is fully observable, where object poses and bounding boxes are known via perception. This information is used in motion-level planning to derive collision-free trajectories (if one exists for a given task plan action). In addition to Chat-GPT[1] (OpenAI 2024) as our LLM of choice, we use Fast Downward (Helmert 2006), an off-the-shelf PDDL solver, for task-level planning in our method as well as baselines (discussed in Section). When planning with Fast Downward, we use the A* algorithm with the landmark cut (LMCUT) heuristic for plan optimality. For collision-free motion-level planning, we use the RRT-Connect (Kuffner and LaValle 2000) algorithm as provided by OMPL (Şucan, Moll, and Kavraki 2012) (included with CoppeliaSim).

Task Settings: We design scenarios in which the robot has to complete several tasks for three tasks of increasing difficulty: 1) tower building, 2) spelling, and 3) organizing a table (Figure 6). The *tower building* task involves the robot assembling a tower of blocks of a given height n, where $3 \leq n \leq 7$, with $n + 1$ blocks provided on the table. The *spelling* task also involves robot constructing a tower of

[1]We tested `gpt-4`, `gpt-4o`, and `chatgpt-4o-latest`, but found `chatgpt-4o-latest` to produce better plans. See more about these model variants here: https://platform.openai.com/docs/models.

blocks of some height n, but with the added constraint that they correctly spell a given word of length n. This requires correct placement of lettered blocks, thus heavily depending on the LLM's ability to generate the correct sequence of pick and place actions. Finally, the *organizing* involves a robot making piles of matching blocks: here, we initialize a scene of 3 block types each with n block instances (where $2 \leq m \leq 4$). This can be seen as a mix of the two prior tasks, where alike but varying numbers of blocks must be placed into piles.

Metrics: We report the following metrics: 1) *plan completion*, which measures the percentage of all plans that were executed from start to finish; 2) *success*, which measures the percentage of all completely executed plans that were successful (i.e., if the final state matches the task prompt); 3) *average plan computation time* (in seconds); 4) *average number of tokens* for LLM prompting; and 5) *average plan length* across all successful executions.

Baseline Methods

We compare our OLP-based method to several baseline methods, for which we provide details below. These baseline methods also use Chat-GPT to either directly output a task plan or PDDL definitions, following the tracks of LLM-based planning work previously introduced.

LLM-Planner This baseline centers the LLM as a planner, serving as a proxy for such methods (Ichter et al. 2023; Driess et al. 2023). We directly provide a textual description of the state of the robot's environment (denoted by $\tilde{s}$) and the robot's executable skills ($\mathcal{A}$) as input and retrieve a task plan $\mathcal{P}_\mu$ as output. We then parse each action to identify target objects and surfaces needed for pick and place actions while performing the necessary motion-level planning to successfully resolve each action. This baseline approach evaluates the LLM's ability to reason about the robot's embodiment and produce a correct task plan.

LLM+P LLM+P (Liu et al. 2023) uses a LLM to generate a PDDL problem definition given a text description of a task planning domain and initial state. As input to the LLM, we provide a description of the robot's environment ($\tilde{s}$) and an example of a problem definition task, and we obtain a problem definition for task $\mathcal{T}$ as output. We then use this output with our domain definition of predefined skills to acquire a task plan using Fast Downward (Helmert 2006), and this task plan is executed and resolved with motion-level planning. This baseline approach evaluates the LLM's ability to accurately generate a PDDL problem file, compatible with a predefined set of skills, without explicitly performing object-level planning and reasoning.

DELTA DELTA (Liu et al. 2024) is a task planning method which auto-regressively prompts a LLM to derive PDDL domain and problem definitions. Similar to our approach, a task $\mathcal{T}$ is broken down into subgoals, each of which are formulated as their own subgoal PDDL problem file. We prompt the LLM with details about robot actions ($\mathcal{A}$) as well as the objects available to the robot, after which a domain file is generated. The LLM is then provided with state description $\tilde{s}$ and the task prompt $\mathcal{T}$ to generate a problem file containing all goals (similar to the output of LLM+P (Liu et al. 2023)). This problem file is then broken down into subgoal problem files based on PDDL subgoals auto-regressively suggested by the LLM; this scopes the problem into subgoal actions that are akin to functional units. We hypothesize that although this method will create simpler and smaller problem definitions, this heavily relies on the LLM's ability to generate syntactically and semantically correct definitions, which may not be as reliable as our method.

Results and Discussion

Our experimental results show that our OLP-based method performs better than baselines that either generate a task plan or PDDL files (Table 1). Across all tasks and approaches, plans were not completely executable due to motion-level planning failures, where plans were not found in a reasonable amount of time. Despite this phenomenon, our OLP approach produces the most plan completions across all task settings on average (Figure 7). While not always successful in execution, our OLP approach, generates plans that exhibit high success rate, matching the intention of the given instruction. Interestingly, the *spelling* task showed the lowest success rate across all approaches: we attribute this to incorrect reasoning done by the LLM at both object and task levels.

Although LLM-Planner generates task plans without a solver, it does not complete a majority of tasks because the LLM has poor understanding of the configuration of the robot's environment for collision-free motion. Failure cases involve attempts to pick up an object blocked by another object or place an object on an occupied spot. LLM+P also exhibits poor performance: although the LLM is capable of directly outputting PDDL, failures were mainly attributed to inaccurate problem definitions. This may be attributed to the fact that LLM+P uses fewer prompts than OLP and DELTA; also, unlike DELTA, LLM+P does not give definitions of PDDL planning operators, thus providing less context to the LLM. We also observed that PDDL problems generated by LLM+P and DELTA were susceptible to incorrect syntax, which is a drawback of LLM-based PDDL generation. DELTA, whose approach closely resembles our method, performs better than LLM+P and LLM-Planner baselines, but it does not perform as well as our method while also generally requiring more tokens on average to generate planning definitions. Similar to DELTA, OLP also demonstrates the advantage of bootstrapping task-level planning with PDDL subgoal definitions (reflected by low average planning times) but without relying upon the LLM to correctly generate PDDL definitions. Our approach also requires less interaction with the LLM than DELTA as reflected by the average number of tokens.

Limitations: Much like how we humans plan, object-level planning serves as a critical interface between language and TAMP. However, we acknowledge several shortcomings of our work: 1) our approach requires explicit robot skill definitions in PDDL, which may not always transfer across robot systems; 2) we assume a set of FOON samples for few-

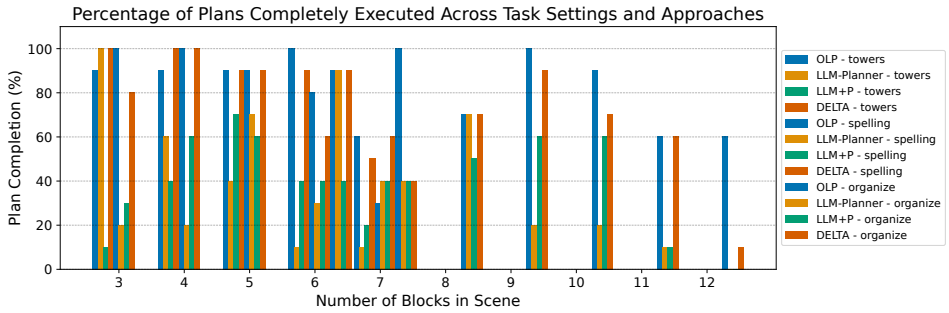


Figure 7: Graph showing percentage of plans completely executed across all tasks and approaches for different number of blocks in the scene.

shot learning; 3) we did not consider plan recovery if objects were knocked down during execution (thus potentially dropping the success rate of completely executed plans); and, most importantly, 4) as with baselines, this approach depends on a correctly generated object-level plan compatible with task-level planning for subgoal definitions. Our evaluations are solely done for pick-and-place tasks, which do not highlight the benefits of the semantic richness of object-level plans. As future work, we will explore broader task diversity and adapting plans to novel settings like prior work (Paulius, Jelodar, and Sun 2018; Sakib, Paulius, and Sun 2022). Like recent work (Han et al. 2024), we can also integrate human feedback to correct LLM-generated errors at the object level. Further, we will explore learning from demonstration to acquire task-level domain definitions to address this work's assumption of predefined skills.

Related Work

Language Models for Planning: Many researchers have explored the use of language models for robotics applications, having been inspired by their remarkable performance in language-related tasks. Prior works have investigated the planning capabilities of LLMs (Silver et al. 2024; Valmeekam et al. 2023). Other works supplement task planning with language models (Liu et al. 2023; Chen et al. 2024; Singh et al. 2023; Singh, Traum, and Thomason 2024; Liu et al. 2024; Han et al. 2024). LLM+P (Liu et al. 2023) generates PDDL problem file via LLM prompting. Much like our work, existing works use LLMs as an informer of subgoals for classical planning (Singh, Traum, and Thomason 2024; Liu et al. 2024; Han et al. 2024) In particular, DELTA (Liu et al. 2024) resembles our method in that it decomposes a task into a series of PDDL subgoal definitions directly outputted by a LLM. Our approach uses a LLM at the object level and not task level (i.e., PDDL). Recent work also iteratively prompts a LLM for generating FOONs (Sakib and Sun 2024). Similar to DELTA, they do not focus on generating nor executing physically valid plans.

Language Models as Planners: Several works treat language models as robotic task planners. SayCan (Ichter et al. 2023) combines a language model and affordance detectors for driving robotic execution given a task prompt. PaLM-E (Driess et al. 2023) is an embodied language model that directly incorporates continuous observations (like images, state estimates, or other sensor modalities) into the language embedding space. These works have shown that LLMs are capable of performing some degree of embodied reasoning. However, one major drawback to these works is that they require a large amount of engineering effort, particularly for them to operate in novel environments or to perform long-horizon tasks. Prior works also exploit the reasoning capabilities of a LLM for solving a wide range of tasks both in simulation (Huang et al. 2022; Gramopadhye and Szafir 2023; Raman et al. 2024) and with a real robot (Raman et al. 2024).

Conclusion

We introduce a hierarchical planning approach that capitalizes on the power of large language models (LLMs) to bootstrap task and motion planning (TAMP). Through an additional planning layer situated above TAMP known as object-level planning (Paulius 2022), we enable robots to flexibly find planning solutions from plan sketches extracted via LLM prompting. When compared to alternative LLM-based planning approaches that either use a LLM as a planner or as a generator of planning definitions like PDDL, our method flexibly enables a robot to solve a wide range of tasks that leverage the expressiveness of natural language.

Acknowledgements

This work was graciously supported by the Office of Naval Research through grant number N00014-21-1-2584, Echo Labs, and by the Austrian Science Fund (FWF) Project P36965 [DOI: 10.55776/P36965].

References

Agostini, A.; and Piater, J. 2023. Unified Task and Motion Planning using Object-centric Abstractions of Motion Constraints. *arXiv preprint arXiv:2312.17605*.

Agostini, A.; Saveriano, M.; Lee, D.; and Piater, J. 2020. Manipulation Planning Using Object-Centered Predicates and Hierarchical Decomposition of Contextual Actions. *IEEE Robotics and Automation Letters*, 5(4): 5629–5636.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell,

A.; et al. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33: 1877–1901.

Chen, Y.; Arkin, J.; Zhang, Y.; Roy, N.; and Fan, C. 2024. AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 6695–6702.

Chiang, T. 2023. ChatGPT is a Blurry JPEG of the Web.

Driess, D.; Xia, F.; Sajjadi, M. S. M.; Lynch, C.; Chowdhery, A.; Ichter, B.; Wahid, A.; Tompson, J.; Vuong, Q.; Yu, T.; Huang, W.; Chebotar, Y.; Sermanet, P.; Duckworth, D.; Levine, S.; Vanhoucke, V.; Hausman, K.; Toussaint, M.; Greff, K.; Zeng, A.; Mordatch, I.; and Florence, P. 2023. PaLM-E: An Embodied Multimodal Language Model. In *International Conference on Machine Learning (ICML)*, 8469–8488. PMLR.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4: 265–293.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Gramopadhye, M.; and Szafir, D. 2023. Generating Executable Action Plans with Environmentally-Aware Language Models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3568–3575. IEEE.

Han, M.; Zhu, Y.; Zhu, S.-C.; Wu, Y. N.; and Zhu, Y. 2024. InterPreT: Interactive Predicate Learning from Language Feedback for Generalizable Task Planning. In *Robotics: Science and Systems (RSS)*.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 9118–9147.

Ichter, B.; Brohan, A.; Chebotar, Y.; Finn, C.; Hausman, K.; Herzog, A.; Ho, D.; Ibarz, J.; Irpan, A.; Jang, E.; Julian, R.; Kalashnikov, D.; Levine, S.; Lu, Y.; Parada, C.; Rao, K.; Sermanet, P.; Toshev, A. T.; Vanhoucke, V.; Xia, F.; Xiao, T.; Xu, P.; Yan, M.; Brown, N.; Ahn, M.; Cortes, O.; Sievers, N.; Tan, C.; Xu, S.; Reyes, D.; Rettinghouse, J.; Quiambao, J.; Pastor, P.; Luu, L.; Lee, K.-H.; Kuang, Y.; Jesmonth, S.; Joshi, N. J.; Jeffrey, K.; Ruano, R. J.; Hsu, J.; Gopalakrishnan, K.; David, B.; Zeng, A.; and Fu, C. K. 2023. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In Liu, K.; Kulic, D.; and Ichnowski, J., eds., *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, 287–318. PMLR.

Kroemer, O.; Niekum, S.; and Konidaris, G. 2021. A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms. *Journal of Machine Learning Research*, 22(30): 1–82.

Kuffner, J.; and LaValle, S. 2000. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, 995–1001.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as Policies: Language Model Programs for Embodied Control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 9493–9500.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. *arXiv preprint arXiv:2304.11477*.

Liu, Y.; Palmieri, L.; Koch, S.; Georgievski, I.; and Aiello, M. 2024. DELTA: Decomposed Efficient Long-Term Robot Task Planning using Large Language Models. *arXiv preprint arXiv:2404.03275*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

OpenAI. 2024. GPT-4o. https://openai.com/index/hello-gpt-4o/. Accessed on November 30, 2025.

Paulius, D. 2022. Object-Level Planning and Abstraction. In *CoRL 2022 Workshop on Learning, Perception, and Abstraction for Long-Horizon Planning*.

Paulius*, D.; Agostini*, A.; and Lee, D. 2023. Long-Horizon Planning and Execution with Functional Object-Oriented Networks. *IEEE Robotics and Automation Letters*, 8(8): 4513–4520.

Paulius, D.; Huang, Y.; Milton, R.; Buchanan, W. D.; Sam, J.; and Sun, Y. 2016. Functional Object-Oriented Network for Manipulation Learning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2655–2662. IEEE.

Paulius, D.; Jelodar, A. B.; and Sun, Y. 2018. Functional Object-Oriented Network: Construction and Expansion. In *2018 IEEE International Conference on Robotics and Automation*, 5935–5941.

Raman, S. S.; Cohen, V.; Idrees, I.; Rosen, E.; Mooney, R.; Tellex, S.; and Paulius, D. 2024. CAPE: Corrective Actions from Precondition Errors using Large Language Models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 14070–14077.

Rohmer, E.; Singh, S. P. N.; and Freese, M. 2013. CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 1321–1326. http://www.coppeliarobotics.com.

Sakib, M. S.; Paulius, D.; and Sun, Y. 2022. Approximate Task Tree Retrieval in a Knowledge Network for Robotic Cooking. *IEEE Robotics and Automation Letters*, 7(4): 11492–11499.

Sakib, M. S.; and Sun, Y. 2024. Consolidating Trees of Robotic Plans Generated Using Large Language Models to Improve Reliability. *International Journal of Artificial Intelligence and Robotics Research*, 2450002.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L.; and Katz, M. 2024. Generalized Planning in PDDL Domains with Pretrained Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18): 20256–20264.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 11523–11530.

Singh, I.; Traum, D.; and Thomason, J. 2024. TwoStep: Multi-agent Task Planning using Classical Planners and Large Language Models. *arXiv preprint arXiv:2403.17246*.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4): 72–82. https://ompl.kavrakilab.org.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Advances in Neural Information Processing Systems*, volume 36, 75993–76005.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 30.

Xie, Y.; Yu, C.; Zhu, T.; Bai, J.; Gong, Z.; and Soh, H. 2023. Translating Natural Language to Planning Goals with Large-Language Models. *arXiv preprint arXiv:2302.05128*.

Zhang, J.; Huang, J.; Jin, S.; and Lu, S. 2024. Vision-Language Models for Vision Tasks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–20.

Appendix A: Object-level Planning Prompts

System: You are a helpful assistant that will generate plans for robots. You will be given the following:
1. A simple plan sketch, with which you will generate an entirely new plan sketch describing object states before
   (preconditions) and after (effects) actions are executed.
2. A list of objects available to the robot.

Note the following rules:
- Closely follow the task prompt. You must use all objects except any objects not related to the task.
- Be consistent with object names throughout the plan.
- All objects are on the table in front of the robot.
- Use one action verb per step. However, any steps involving "pick" or "place" must be written as a single step with
   the action "pick and place".
- Use as many states as possible to describe object preconditions and effects.
- Only use the states "in", "on", "under", or "contains" for describing objects. List them in the format "<relation> <obj>", where <relation> is a state and <obj> is a single object.

Figure 8: System prompt for our OLP method (refer to Section).

User: Your task will be to create a step-by-step plan for the following prompt: $\mathcal{T}$. The following objects are available in
the scene: $<$ objects_in_scene $>$. Say 'Okay!' if you understand the task.

LLM: Okay!

User: Below are a list of prototype recipes. You must select the closest one that is the closest to the given task prompt. Simply provide the number corresponding to the closest prototype.

$<$ Step-by-step language plans for examples $\mathcal{X}_{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\} >$

LLM: $<$ Prototype selection $\hat{\mathcal{G}} \in \mathcal{X}_{\mathcal{G}} >$

User: Generate a concise plan using the prototype as inspiration for the task: $\mathcal{T}$. Follow all guidelines. Give evidence to
support your plan logic.

LLM: $<$ Step-by-step instructions for task $\mathcal{T}$ as $\mathcal{P}_L >$

User: Make a Python list of used objects in the following format: ["object_1", "object_2", ...]'. If there are several
instances of an object type, list them individually (e.g., ['first apple', 'second apple'] if two apples are used). Do not add any explanation.

LLM: $<$ List of objects needed for task $\mathcal{T} >$

User: Format your generated plan as a JSON dictionary. List as many states as possible when describing each object's preconditions and effects. Each required object should match a key in "object_states": Be consistent with object names across actions. Use this JSON prototype as reference:

$<$ JSON equivalent of $\hat{\mathcal{G}} >$

LLM: $<$ JSON for task $\mathcal{T}$ as $\tilde{\mathcal{G}}_{\mathcal{T}} >$

Figure 9: Progressive chain-of-thought (CoT) prompting for extracting object-level plans from a LLM (refer to Section). We highlight special input provided to the LLM (particularly the user task query $\mathcal{T}$, objects in the scene, and few-shot examples—both as language instructions and JSON) as well as output acquired from the LLM in blue and orange colours respectively. A few-shot sample codified as a JSON structure is shown as Figure 10.

```
{
   "plan": [{
      "step": 1,
      "action": "pick and place",
      "required_objects": ["first block", "second block"],
      "object_states": {
         "first block": {"preconditions": ["under nothing", "on table"], "effects": ["under second block", "on table"]},
         "second block": {"preconditions": ["under nothing", "on table"], "effects": ["on first block", "under nothing"]},
      },
      "instruction": "Pick and place second block from table on first block."
   }]
}
```

Figure 10: JSON equivalent of a functional unit presented as Figure 3.

Appendix B: Baseline Method Prompts

LLM-Planner

System: You are a helpful PDDL planning expert. Your job is to process a task prompt, a list of objects in the scene, and
a list of statements describing the environment state, reason about how to solve the task, and produce a plan that solves the task.

A task plan has the format of:
1. (< action_1 >< arg1 >< arg2 >)
2. (< action_2 >< arg1 >< arg2 >)
3. ...

Observe the following rules:
- In the task plan, you can only use these actions:
1. (< pick >< obj1 >< obj2 >) - pick < obj1 > that is on top of < obj2 >; this causes nothing to be on < obj2 >.
2. (< place >< obj1 >< obj2 >) - place < obj1 > on top of < obj2 >; < obj2 > must have nothing on it for < obj1 > to
be placed on it.
- Note the order of the arguments for both actions!
- The agent executing this task has a single hand: in order to pick up an object, the agent's hand must be free.

User: There is a scenario with the following objects: < objects_in_scene >. Please await further instructions.

User: Your task is as follows: $\mathcal{T}$. Transform this instruction into a PDDL goal specification in terms of 'on' relations.
Do not add any explanation.

LLM: < Goal state description for task $\mathcal{T}$ >

User:: Find a task plan in PDDL to achieve this goal given the initial state below. Only specify the list of actions needed. Use the actions defined above. Do not add any explanation.

Initial state: < State description as text $\tilde{s}$ >

LLM: < Task plan as text $\mathcal{P}_\mu$ >

Figure 11: Prompts for LLM-Planner baseline (Section). This baseline simply provides a textual description of the robot's actions as well as the current object configuration, with which it must reason about the correct sequence of actions that will result in collision-free execution and resolve the task.

LLM+P

User: I want you to generate a PDDL problem file for robot problem solving. An example planning problem is:

< PDDL Problem File Example$\tilde{s}$ >

Now I have a new planning problem and its description is as follows: These objects are on the table: < objects_in_scene >. The current state of the world is: < State description as text $\tilde{s}$ >.

Your goal is to achieve this task: $\mathcal{T}$. Provide me with the problem PDDL file that describes the new planning problem directly without further explanations.

LLM: < PDDL Problem Definition >

Figure 12: Prompts for the LLM+P (Liu et al. 2023) baseline (Section). Similar to our OLP method, we provide the LLM with an example PDDL problem file definition for the most similar few-shot example. These prompts were based on those provided by Liu et al. (2023) online.[2]

DELTA

User: Role: You are an excellent PDDL domain file generator. Given a description of action knowledge in natural language, you can generate a PDDL domain file.

Example: < PDDL Domain File Example >

Instruction: A new domain includes the following objects: < objects_in_scene >. Please generate a corresponding new PDDL domain file for a robot. Do not add any explanation.

LLM: < PDDL Domain Definition for task $\mathcal{T}$ >

Figure 13: PDDL domain file prompting for the DELTA (Liu et al. 2024) baseline (Section). Similar to our OLP method, we provide the LLM with an example PDDL domain file definition for the most similar few-shot example.

User: Role: You are an excellent PDDL problem file generator. Given a description of the robot's environment and a goal description, you can generate a PDDL problem file.

Example: < PDDL Problem File Example >

Instruction: Now given a new description of the robot's scene and using the predicates in the previously generated PDDL domain file, please generate a new PDDL problem file for the task: $\mathcal{T}$.

< State description as text $\tilde{s}$ >

LLM: < PDDL Problem Definition for task $\mathcal{T}$ >

Figure 14: PDDL problem file prompting for the DELTA (Liu et al. 2024) baseline (Section). Similar to our OLP method, we provide the LLM with an example PDDL problem file definition for the most similar few-shot example. This prompt was slightly modified to account for the lack of a scene graph in this work.

[2]LLM+P Repository: https://github.com/Cranial-XIX/llm-pddl

```
User: Role: You are an excellent assistant in decomposing long-term goals. Given a PDDL problem file, you can decompose
      the long-term goal in a sequence of subgoals.

      Example: < PDDL Subgoals Example >

      Instruction: Given the PDDL problem previously generated, please decompose the long-term goal into a sequence
      of subgoals considering the predicates and actions from the previously generated PDDL domain file. Simply list
      the decomposed PDDL subgoals for each instruction in a similar format as the example and only 1 level deep.

LLM: < PDDL Subgoals for task T >
```

Figure 15: Subgoal prompting for the DELTA (Liu et al. 2024) baseline (Section). To accompany the few-shot example from the prior prompts, we also defined an example decomposition of subgoals as text. Our implementation of the baseline then extracts each set of subgoals to then generate PDDL subgoal problem files using the current state at the start of subgoal execution.

Appendix C: Task Settings

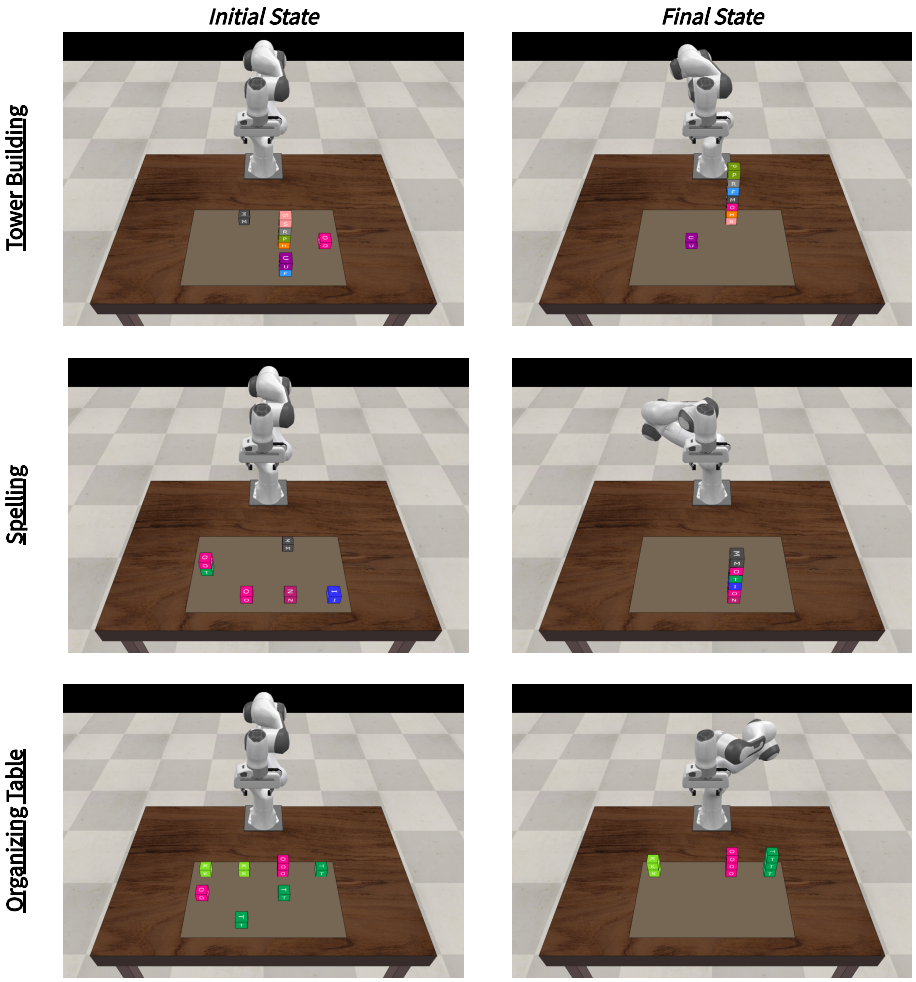


Figure 16: Examples of the initial and final states for all task settings defined in Section . In the *tower building* example, the robot must simply construct a tower of 7 blocks. In the *spelling* task, the robot must assemble a tower spelling the word "MOTION". Finally, in the *organizing table* task, the robot must stack all alike blocks into separate towers or piles.