Bootstrapping Object-Level Planning with Large Language Models

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

***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). Existing work uses LLMs to directly output task plans or generate goals in representations like PDDL. However, these methods fall short because they rely on the LLM to do the actual planning or output a hard-to-satisfy goal. Our approach instead extracts knowledge from an 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 method markedly outperforms alternative planning strategies in completing several pick-and-place tasks in simulation.[†]**

I. 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 [1, 2] and embodied agents [3, 4]. These approaches use LLMs as either a planner [1, 5, 2], or a goal generator [6, 7, 8, 9]. As a task planner, an LLM is informed of the task and scene and directly outputs a complete plan, thus forgoing automated planning with off-the-shelf planners [10]. Plan actions generated by an LLM are then grounded to action policies or primitives. As a task goal generator, an LLM generates planning definitions in the form of representations like PDDL [11] (short for *Planning Domain Definition Language*); this type of approach is often associated with task and motion planning (TAMP) [12].

However, existing work in these categories fails to handle complex, goal-oriented tasks in several key aspects. On the 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 [13]. 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.

It is natural to exploit language models for planning as they contain useful domain knowledge and often output useful steps. Similarly, they are useful as goal generators because one can still rely on off-the-shelf planners. This work uses an LLM to generate *partial goal schemas at the object level*, which can then form PDDL subgoals. Such an approach inherits the desirable commonsense planning knowledge of the LLM while supporting sound and complete task-level planning. The object

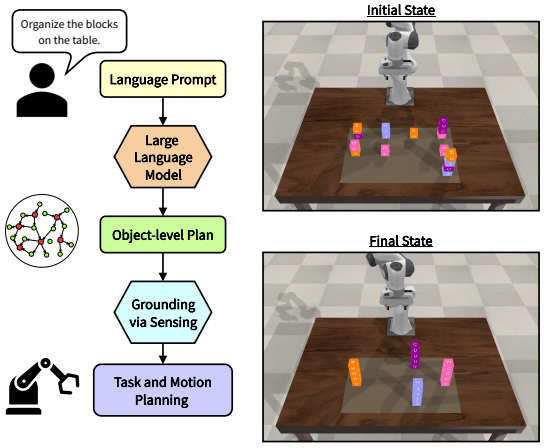


Fig. 1. Our approach prompts an LLM for object-level information with which we construct an object-level plan (as a FOON). This plan schema *bootstraps* task- and motion-level planning (TAMP) via PDDL subgoals.

(as opposed to task) level is the level at which natural language is most appropriate and at which most knowledge is captured and expressed [14, 15]. While task-level planning focuses on action or motion constraints for execution, *object-level planning* focuses on object interactions without committing to *how* these effects will be resolved until runtime.

We propose a modular approach that distills domain knowledge from an LLM to generate *object-level plans* [15], which then 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) [16]. Recent work has shown how object-level knowledge in FOON can automatically generate PDDL subgoals [17]; however, this assumes that partial plan specifications already exist as a FOON. We exploit the capabilities of LLMs for object-level planning, overcoming the inability of LLMs to directly output feasible task plans while exploiting the higher, object-level nature of LLM output and language as a whole.

The contributions of our work are as follows: first, we introduce a modular planning approach (Figure 1) that interfaces with an LLM to generate natural language instructions, from which we transform into an OLR (e.g., FOON) for hierarchical planning. Second, 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. Finally, we showcase markedly better performance than alternative LLM-based methods.

II. Background

Large Language Models: A large language model (LLM) is a complex neural network model trained via self-supervised

[1]Department of Computer Science, Brown University, USA
[2]Department of Computer Science, University of Innsbruck, Austria
*Corresponding Author (*Email*: dpaulius@cs.brown.edu)
[†]Project Website: https://davidpaulius.github.io/olp_llm/

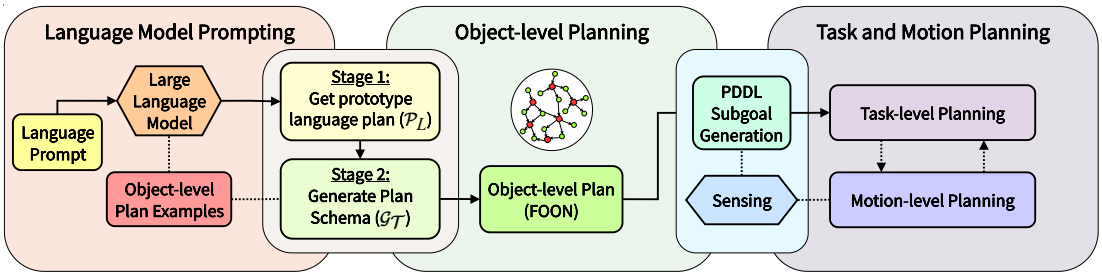


Fig. 2. Our approach interfaces with a language model to generate object-level plans (as FOON graphs) for bootstrapping task and motion planning. We generate task-level subgoals as PDDL subgoals by grounding object-level subgoals to the robot's environment; with these task-level definitions, task planning to obtains task plan segments per object-level action, which are executed using motion-level planning, improving prior work [17].

learning and self-attention [18]. LLMs have shown remarkable performance in natural language processing (NLP) and text generation tasks. Variants such as GPT [19] and LLaMA [20] are trained on large corpora of text collected from the Internet and fine-tuned using RLHF (reinforcement learning from human feedback). For this reason, an LLM can be thought of as a "compressed" representation of domain knowledge from the web [21], which is why we aim to exploit these models to inform planning. This work uses OpenAI's Chat-GPT [22].

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* to enable robots to solve complex long-horizon tasks [12]. At the lower level, *motion planning* finds collision-free robot motion or trajectories that are typically used to achieve a task. However, typical robot tasks are too complex for motion-level planning alone. For this reason, *task planning* is necessary as an added layer to reason over an abstraction of the robot's actions and environment. 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 $\mathcal{P} = \{a_1, ..., a_n\}$ [10]. An action a refers to a robot-executable skill or policy; our work assumes access to a repertoire of skills, which we denote by $\mathcal{A}$, which are defined as planning operators in PDDL [11]. Finally, given the task plan $\mathcal{P}$, motion planning finds collision-free trajectories that reproduce the intended effects of each action; our work uses OMPL [23] for motion planning.

A. 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 [24, 13]. Other works supplement task planning with language models [6, 25, 5, 26, 8, 27]. LLM+P [6] generates PDDL problem file via LLM prompting. Much like our work, existing works use LLMs as an informer of subgoals for classical planning [26, 8, 27, 9]. In particular, DELTA [8] resembles our method in that it decomposes a task into a series of PDDL subgoal definitions directly output by an LLM. Our approach uses an LLM at the object level and not task level (i.e., PDDL). Recent work also iteratively prompts an LLM for FOON generation [28]. 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 [1] combines a language model and affordance detectors for driving robotic execution given a task prompt. PaLM-E [2] 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 language models are capable of performing some degree of embodied reasoning. However, one major drawback of these works is that they require a large amount of engineering effort, particularly to enable them to operate in novel environments and solve long-horizon tasks. Previous works also exploited the reasoning capabilities of an LLM to solve a wide range of tasks both in simulation [3, 29, 4] and with a real robot [4].

III. Object-level Planning with Language Models

There exists a disconnect between language and the task level, which makes TAMP 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 because of the inability of LLMs to correctly reason about task- and motion-level constraints. It is impractical to provide the entire context of a task setting to an 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 strength of language models lies in their ability to provide approximate 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. Imagine your typical cooking recipe, for instance: a recipe provides a sketch of object interactions and is agnostic to the state of the reader's kitchen or the recipe writer's kitchen. It also does not provide details on how actions should be executed (e.g., which hand should be used, how should an object be grasped, etc.). What a recipe expressed in natural language may provide, however, is an idea of the types of actions and inter-object interactions necessary to complete a task. Rather, the exact details of execution at both task and motion levels are resolved at run time.

For these reasons, we adopt an *object-level planning* approach to bootstrap task and motion planning [17]. We generate object-level plan sketches, which provide task-level subgoals that naturally interface language and decision making, using an LLM. Briefly, given a language command to a robot, our approach (Figure 2) uses an 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 is then executed via TAMP. Task-level planning is achieved by transforming each OLP action into PDDL definitions to find task plan segments.

A. Object-Level Planning

We adopt another layer of planning above TAMP called object-level planning, which considers changes to object state [15]. We use an object-level representation in the form of a knowledge graph called the *functional object-oriented network* (FOON) [16, 30]. Formally, a FOON $\mathcal{G} = \{\mathcal{O}, \mathcal{M}, \mathcal{E}\}$ is a bipartite graph with 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 close 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 in Figure 3, which describes an action for a block stacking task. We argue that foundation models naturally interface with object-level representations due to their similarity to human language, which in turn allows them to interface with tools that combine vision and language [19, 31].

B. Generating Object-level Plans with LLMs

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 those that 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 prompts an 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

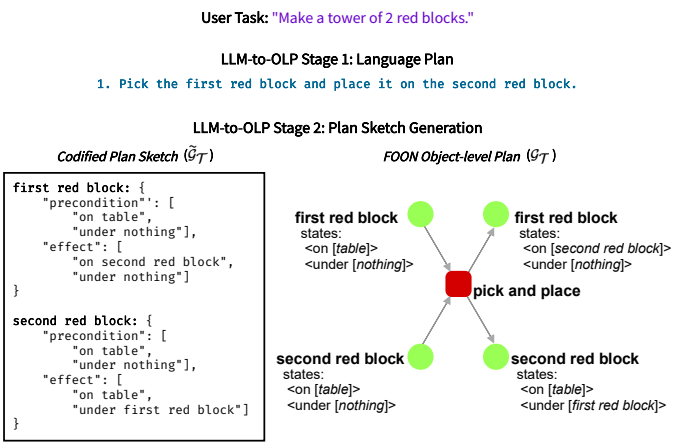


Fig. 3. Illustration of how a user task specified in natural language is transformed into an object-level plan (OLP) as a FOON via LLM prompting.

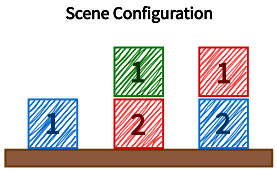
(without *any* context about their present configuration), we expect text instructions $\mathcal{P}_L$ that solely mention red blocks for the task *"Make a tower of 2 red blocks."* During this step, we transform the top-k most similar FOONs in $\mathcal{X}_{\mathcal{G}}$ into example plan sketches, from which the LLM selects the 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 IV-A). 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 generate a new FOON $\mathcal{G}_{\mathcal{T}}$ for the novel task. Inspired by previous work on code writing for robots [32], we codify $\hat{\mathcal{G}}$ as a JSON. The LLM then outputs a codified OLP $\tilde{\mathcal{G}}_{\mathcal{T}}$, which captures each instruction $\xi_i \in \mathcal{P}_L$. Finally, each action in $\tilde{\mathcal{G}}_{\mathcal{T}}$ forms a functional unit $\mathcal{FU}_i \in \mathcal{G}_{\mathcal{T}}$.

IV. 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 [17]. Therefore, we use $\mathcal{G}_{\mathcal{T}}$ to *bootstrap* TAMP via 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 [17].

A. Object-Level to Task-Level Planning

The aim of task-level planning is to find a robot-executable *task plan* $\mathcal{P}_{\mu}$ that solves task $\mathcal{T}$. A task plan is composed



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)
```

Fig. 4. Example of task-level grounding for an object-level plan (Figure 3), which is compatible by design with the planning operators in Figure 5.

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: a *domain definition* and a *problem definition* [11]. A domain definition provides details on what actions can be taken by a robot 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 target goal state (g_μ) as logical predicates. We assume a predefined domain definition with planning operators corresponding to a repertoire of parameterized, robot-executable skills $\mathcal{A}$. For the generation of problem definitions, both s_μ and g_μ are adapted from a functional unit $\mathcal{FU} \in \mathcal{G}_\mathcal{T}$: predicates are constructed based on the object-state pairs in $\mathcal{FU}$ [17]. In other words, transforming a FOON into PDDL requires mapping attributes of each object o to predicates (where $o \in \mathcal{O}_{in} \cup \mathcal{O}_{out}$).

Object-centered Predicates: We use object-centered predicates [33, 17] that describe constraints for collision-free 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 [34]. For example, the predicate `(on block_1 block_2)` denotes that `block_2` lies on top of `block_1`. We also use a virtual object *air* to describe free space in or on objects, which is important for collision-free picking, i.e., `(on block_1 air)`—nothing is on a block, which makes it free for grasping).

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 an OLP (Figure 3), an LLM will map them to object instances `red_block_1` and `red_block_2` (Figure 4). Once completed, we obtain a mapping of object-state pairs 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 previous work [34].

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

(a) Pick Action

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

(b) Place Action

Fig. 5. Planning operators for *pick* and *place* actions using object-centered predicates [33] and executable via motion-level planning (Section IV-B).

B. 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 resolve object-level subgoals. We use motion-level planning to find collision-free robot movements that will achieve 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. The initial and final poses of the hand for these actions can be obtained directly from object-centered hand-object relations encoded in the preconditions and effects of their corresponding planning operators using geometric rotation and translation transformations [34].

V. Evaluation

We evaluate the flexibility of our approach (denoted as **OLP** in Table I) with alternative methods on several tasks in simulated experiments. Our results show that an LLM cannot reliably produce PDDL definitions and is unable to reliably task plan due to its lack of spatial understanding; however, we can prompt an LLM for object-level subgoals compatible with our modular approach from previous work [17].

A. Experimental Setup

We perform experiments in a simulated table-top environment in CoppeliaSim [35] with a Franka Emika Panda robot affixed to a table upon which blocks are randomly initialized. Given a task specified in natural language, the robot must perform a sequence of *pick* and *place* actions (defined in Figure 5) fulfilling the task. We assume that the state of the environment is fully observable—object poses and bounding boxes are known via perception. This information is used in motion-level planning to generate collision-free trajectories. In addition to Chat-GPT[3] [22] as our LLM of choice, we

[3]We tested `gpt-4`, `gpt-4o`, and `chatgpt-4o-latest`, but found `chatgpt-4o-latest` to produce the best plans, adhering to instructions.

TABLE I
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%**	**76.00%**	**0.0043 ± 0.0021**	2406.38 ± 335.0091	10.2791 ± 4.5687
	LLM-Planner	44.00%	34.00%	12.0486 ± 6.3784	**744.94 ± 120.9533**	10.2791 ± 4.5687
	LLM+P	18.00%	34.00%	0.0346 ± 0.0312	1656.42 ± 170.2912	**8.5556 ± 3.9291**
	DELTA	**86.00%**	60.00%	0.0067 ± 0.01206	4871.88438.2610	9.0233 ± 4.7883
Spelling	OLP	**80.00%**	**62.00%**	0.02715 ± 0.0828	2588.66 ± 379.1376	**8.45 ± 3.4932**
	LLM-Planner	22.00%	16.00%	7.5734 ± 2.4650	**754.06 ± 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%	50.00%	**0.0075 ± 0.0061**	4836.72 ± 475.0059	10.5641 ± 5.5998
Organize	OLP	**81.43%**	**77.14%**	**0.0080 ± 0.0053**	3051.5 ± 499.4818	15.3684 ± 7.1630
	LLM-Planner	35.71%	22.86%	24.1510 ± 15.9711	**885.0571 ± 126.3086**	**8.40 ± 2.2361**
	LLM+P	37.14%	37.14%	0.0538 ± 0.1139	1891.3286 ± 212.4083	11.3077 ± 4.1547
	DELTA	67.14%	54.29%	0.0139 ± 0.0312	5329.9571 ± 470.9973	13.8298 ± 6.4042

use Fast Downward [36], an off-the-shelf PDDL solver, for task-level planning in our method and baselines (discussed in Section V-B). When planning with Fast Downward, we use the A* algorithm with the landmark cut (LMCUT) heuristic for plan optimality. For motion-level planning, we use RRT-Connect [37] as provided by OMPL [23].

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 blocks of some height n, but with the added constraint that the blocks 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* task 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 *regardless* of whether the task objective was achieved; 2) *success*, which measures the percentage of successfully executed plans that achieve the task objective; 3) *average plan computation time* (in seconds); 4) *average number of tokens* for LLM prompting; and 5) *average plan length* across all successful executions.

B. Baseline Methods

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

1) LLM-Planner: This baseline serves as a proxy for methods that directly plan with LLMs [1, 2]. We directly provide a textual description of the state of the robot's environment

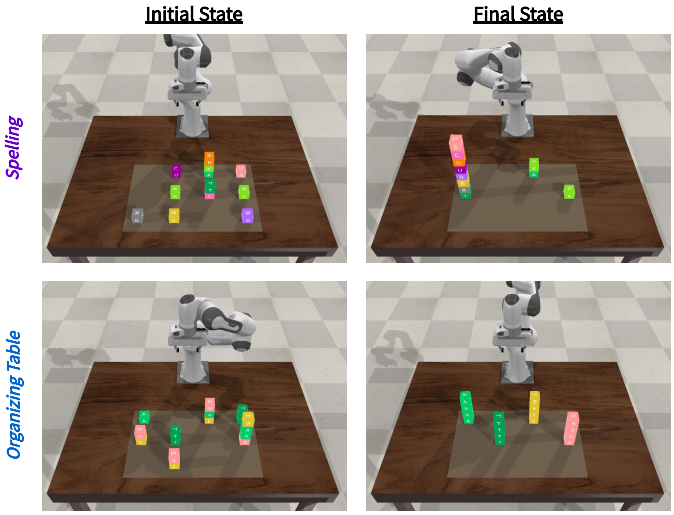


Fig. 6. Example of initial and final states for the *spelling* and *organizing table* tasks. The *tower building* task is akin to *spelling* without ordering constraints.

(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.

2) LLM+P: LLM+P [6] uses an LLM to generate a PDDL problem definition given a text description of a task planning domain and the initial state of the scene. 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 [36], 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.

3) DELTA: DELTA [8] is a task planning method that auto-regressively prompts an LLM to derive PDDL domain and problem definitions. Similar to our approach, a task $\mathcal{T}$ is broken down into subgoals, each of which is 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 a state description $\tilde{s}$ and a task prompt $\mathcal{T}$ to generate a problem file that contains all goals (similar to the output of LLM+P [6]). 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, it heavily relies on the LLM's ability to generate syntactically and semantically correct definitions, which may not be as reliable as our method.

C. Results and Discussion

Our experimental results show that our OLP-based method performs better than baselines that either directly generate a task plan or PDDL files (Table I). Across all tasks and evaluated approaches, we found that some plans were not fully executable due to motion-level planning failures, where the plans were not found in reasonable time. Despite this phenomenon, our approach produces the most plan completions in all task settings on average (Figure 7). Although OLP was not always successful in execution, our approach generates plans that exhibit the highest success rates, matching the intention of the given instruction. Interestingly, the *spelling* task showed the lowest success rate in all approaches. We attribute this to incorrect reasoning performed by the LLM at both the object and task levels, where the LLM may generate a plan sketch to stack the blocks in an incorrect or reversed order.

Although LLM-Planner generates plans without a solver, it does not complete a majority of tasks because the LLM poorly understands the configuration of the robot's environment for collision-free motion. As a result, it incorrectly proposes actions that attempt to pick up an object blocked by another object or place an object in 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 due to the fact that LLM+P uses fewer prompts than OLP and DELTA; also, unlike DELTA, LLM+P does not provide definitions of PDDL planning operators, thus providing less context to the LLM. We also observed that the 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.

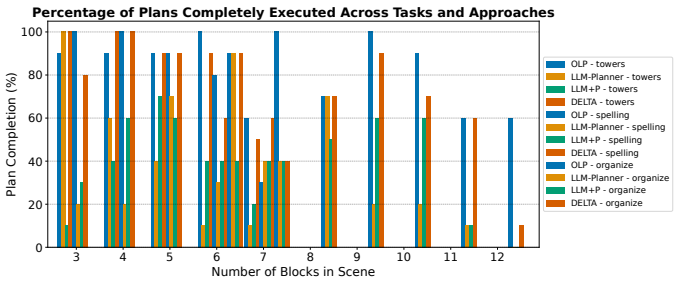


Fig. 7. Graph showing percentage of plans completely executed using all approaches for different number of blocks across tasks (best viewed in colour).

Limitations: Much like how we humans plan, object-level planning serves as a critical interface between language and TAMP. Our approach requires robot skill definitions specified as PDDL, which may not always transfer across robot systems. However, we assume a set of FOON samples for few-shot learning. Further, we did not consider plan recovery if objects were knocked down during execution, thus lowering the success rate of completely executed plans. Most importantly, as with baselines, this approach depends on a correctly generated object-level plan compatible with task-level planning for subgoal definitions. In addition, our evaluations are performed solely on pick-and-place tasks, which do not highlight the benefits of the semantic richness of object-level plans. In future work, we will explore broader task diversity and examine how we can use an LLM to adapt existing object-level plans to novel scenarios similar to prior work [38, 39]. Like recent work [27], we can also integrate human feedback to correct LLM-generated errors at the object level. We will also explore learning from demonstration to acquire task-level domain definitions to address our assumption of predefined skills.

VI. 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 added layer of planning situated above TAMP known as object-level planning [15], we enable robots to flexibly find planning solutions from plan sketches extracted via LLM prompting. Compared to alternative LLM-based planning approaches that either use an LLM as a planner or as a generator of planning definitions like PDDL [11], our method flexibly enables a robot to solve a wide range of tasks that greatly benefit from the expressiveness of natural language.

Acknowledgements

This work was supported by the Office of Naval Research (ONR) under the REPRISM MURI N000142412603, ONR grants N00014-21-1-2584 and N00014-22-1-2592, Echo Labs, and the Austrian Science Fund (FWF) Project P36965 [DOI: 10.55776/P36965]. Partial funding for this work was provided by The Robotics and AI Institute (formerly "The AI Institute").

REFERENCES

[1] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu, "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2023, pp. 287–318.

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

[3] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 162, 2022, pp. 9118–9147.

[4] S. S. Raman, V. Cohen, I. Idrees, E. Rosen, R. Mooney, S. Tellex, and D. Paulius, "CAPE: Corrective Actions from Precondition Errors using Large Language Models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 14 070–14 077.

[5] 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)*, 2023, pp. 11 523–11 530.

[6] 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.

[7] 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.

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

[9] N. Kumar, W. Shen, F. Ramos, D. Fox, T. Lozano-Pérez, L. P. Kaelbling, and C. R. Garrett, "Open-World Task and Motion Planning via Vision-Language Model Inferred Constraints," *arXiv preprint arXiv:2411.08253*, 2024.

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

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

[12] 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, pp. 265–293, 2021.

[13] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati, "On the Planning Abilities of Large Language Models - A Critical Investigation," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 75 993–76 005.

[14] O. Kroemer, S. Niekum, and G. Konidaris, "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms," *Journal of Machine Learning Research*, vol. 22, no. 30, pp. 1–82, 2021.

[15] D. Paulius, "Object-Level Planning and Abstraction," in *CoRL 2022 Workshop on Learning, Perception, and Abstraction for Long-Horizon Planning*, 2022.

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

[17] D. Paulius*, A. Agostini*, and D. Lee, "Long-Horizon Planning and Execution with Functional Object-Oriented Networks," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 4513–4520, 2023.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[19] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language Models are Few-Shot Learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

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

[21] T. Chiang, "ChatGPT is a Blurry JPEG of the Web," Feb 2023. [Online]. Available: https://www.newyorker.com/tech/annals-of-technology/chatgpt-is-a-blurry-jpeg-of-the-web

[22] OpenAI, "GPT-4o," https://openai.com/index/hello-gpt-4o/, 2024, accessed on March 26, 2025.

[23] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[24] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz, "Generalized Planning in PDDL Domains with Pretrained Large Language Models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 18, pp. 20 256–20 264, Mar. 2024.

[25] Y. Chen, J. Arkin, 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)*, 2024, pp. 6695–6702.

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

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

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

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

[30] D. Paulius and Y. Sun, "A survey of knowledge representation in service robotics," *Robotics and Autonomous Systems*, vol. 118, pp. 13–30, 2019.

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

[32] 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)*, 2023, pp. 9493–9500.

[33] A. Agostini, M. Saveriano, D. Lee, and J. Piater, "Manipulation Planning Using Object-Centered Predicates and Hierarchical Decomposition of Contextual Actions," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5629–5636, 2020.

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

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

[36] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[37] J. Kuffner and S. LaValle, "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)*, vol. 2, 2000, pp. 995–1001.

[38] D. Paulius, A. B. Jelodar, and Y. Sun, "Functional Object-Oriented Network: Construction and Expansion," in *2018 IEEE International Conference on Robotics and Automation*, 2018, pp. 5935–5941.

[39] M. S. Sakib, D. Paulius, and Y. Sun, "Approximate Task Tree Retrieval in a Knowledge Network for Robotic Cooking," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 492–11 499, 2022.