# COMPOSUITE: A COMPOSITIONAL REINFORCEMENT LEARNING BENCHMARK

**Jorge A. Mendez***, **Marcel Hussing***, **Meghna Gummadi**, and **Eric Eaton**
Department of Computer and Information Science, University of Pennsylvania
{mendezme,mhussing,meghnag,eeaton}@seas.upenn.edu

## ABSTRACT

We present CompoSuite, an open-source simulated robotic manipulation benchmark for compositional multi-task reinforcement learning (RL). Each CompoSuite task requires a particular *robot* arm to manipulate one individual *object* to achieve a task *objective* while avoiding an *obstacle*. This compositional definition of the tasks endows CompoSuite with two remarkable properties. First, varying the robot/object/objective/obstacle elements leads to hundreds of RL tasks, each of which requires a meaningfully different behavior. Second, RL approaches can be evaluated specifically for their ability to learn the compositional structure of the tasks. This latter capability to functionally decompose problems would enable intelligent agents to identify and exploit commonalities between learning tasks to handle large varieties of highly diverse problems. We benchmark existing single-task, multi-task, and compositional learning algorithms on various training settings, and assess their capability to compositionally generalize to unseen tasks. Our evaluation exposes the shortcomings of existing RL approaches with respect to compositionality and opens new avenues for investigation.

## 1 INTRODUCTION

Compositionality is ubiquitous in artificial and biological computational systems: it arises in natural language, logical reasoning, and software programs, among others. Artificial intelligence (AI) has leveraged compositionality from early work on hierarchical planning (Sacerdoti, 1974) and logic-based reasoning (Doyle, 1979), to modern learning-based techniques like neural module networks (Andreas et al., 2016) and skill discovery (Konidaris & Barto, 2009). The ability to decompose a complex problem into easier subproblems, such that the solutions to these subproblems can be combined into an overall solution, could increase the capabilities of learning agents. First, the learning problem itself might become easier, due to the ease of learning each subproblem. In particular, if the agent is learning multiple tasks that share common subproblems, it could amortize the cost of discovering the decomposition across the multiple tasks. Moreover, the agent could quickly solve new tasks by discovering which components are suitable to these new tasks—in the most extreme case, if the agent is informed about which components are needed (e.g., in the form of a task descriptor), then it could achieve *zero-shot compositional generalization* without requiring any data from the new tasks.

Despite the appeal of these ideas, few reinforcement learning (RL) efforts have sought to leverage compositional properties of the environment to generalize to *unknown* combinations of *known* components. In this work, we present CompoSuite[1], a benchmark for compositional RL that exploits the compositionality of robot learning tasks to evaluate the compositional capabilities of learning agents. We follow the functional composition formulation, which decomposes the learning problem into subproblems whose outputs become the inputs to other subproblems (Mendez et al., 2022), akin to the decomposition of programs for solving robot tasks into software modules for sensing, planning, and acting.

CompoSuite comprises *hundreds* of RL tasks, each made up of four components: robot arm, obstacle, object, and task objective. For example, one task requires an IIWA arm to circumvent a wall, pick up a dumbbell, and place it in a bin. Another task instructs a Jaco arm to traverse a doorway, pick up a plate, and insert it into a shelf. If a learner appropriately decomposes its solutions to these two problems into functional components, it could reuse the IIWA motor module in place of the Jaco motor module to solve the plate-on-shelf task without any experience with the IIWA arm on that task. More generally, multi-task and continual RL approaches can be evaluated on CompoSuite for their ability to handle large numbers of highly varied tasks. This is in stark contrast to most existing multi-task RL benchmarks, which are typically limited to at most a few dozen RL tasks: we offer an order of magnitude more tasks, enabling the study of multi-task and continual RL at scale. The main contributions of this work include:

- CompoSuite, a benchmark of 256 compositional simulated robotic manipulation tasks with distinct optimal behaviors.

*The two first authors contributed equally to this work.
[1] https://github.com/Lifelong-ML/CompoSuite

- Evaluation schemes and metrics for reproducible evaluation of future approaches under CompoSuite.
- An evaluation of single-task, end-to-end multi-task, and modular multi-task agents on various settings under CompoSuite, showing that these existing approaches fall short of fully exploiting the compositional properties of CompoSuite.

## 2 Related work

**Composition in supervised learning** The majority of work on learning compositional representations has been in the supervised setting. One recently popular idea has been to learn a separate neural net module for each task component, encoding the composition directly in the architecture. Approaches in this setting assume that the agent is given access to the ground-truth compositional graph (Andreas et al., 2016; Hudson & Manning, 2018), that it must learn the graph directly from data (Rosenbaum et al., 2018; Alet et al., 2018; Chang et al., 2019), or that it must learn a soft approximation to the graph (Kirsch et al., 2018; Meyerson & Miikkulainen, 2018). These works have shown not only that modularity can increase data efficiency, but also that modular agents are better able to generalize to new combinations of known task components. In the continual or lifelong learning setting, most modular approaches assume that modules can be trained on a single task and used to generalize to future tasks (Reed & de Freitas, 2016; Fernando et al., 2017; Valkov et al., 2018; Veniat et al., 2021). More recent techniques instead follow a continual learning process that continues to improve modules with knowledge from future tasks (Mendez & Eaton, 2021; Ostapenko et al., 2021).

**Composition in reinforcement learning** Compositionality has also been studied in hierarchical RL in the form of *temporal* abstractions (Sutton et al., 1999). This has led to approaches that automatically learn skills that can be chained in sequence to execute complex behaviors (Konidaris & Barto, 2009; Bacon et al., 2017; Sharma et al., 2020), and that can be transferred to new tasks by learning a new mechanism to combine known skills (Florensa et al., 2017). Another form of hierarchical RL instead leverages state abstractions to improve learning efficiency by learning the policy on a compressed representation of the state (Dayan & Hinton, 1993; Dietterich, 2000; Vezhnevets et al., 2017; Abel et al., 2018). Other recent works have composed policies to solve new tasks that are logical compositions of known tasks (Todorov, 2009; Barreto et al., 2018; Haarnoja et al., 2018; Van Niekerk et al., 2019; Nangue Tasse et al., 2020).

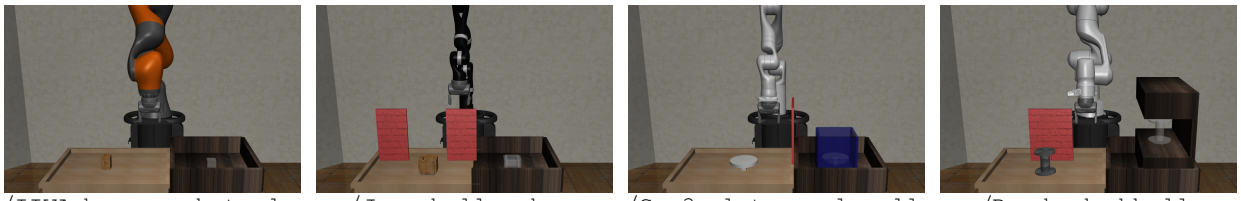
Functional composition has received far less attention in RL. Devin et al. (2017) combined neural net modules to solve robotics tasks, and others have automatically discovered the decomposition of a policy into modules (Goyal et al., 2021; Mittal et al., 2020; Yang et al., 2020). Most recently, Mendez et al. (2022) formalized functional compositionality in RL, and demonstrated that it improves sample efficiency and compositional generalization in multi-task and continual RL.

**Existing benchmarks** The development of large-scale, standardized benchmarks has been key to the acceleration of deep learning research (e.g., ImageNet; Deng et al., 2009). Efforts to create equivalent advancements in deep RL have led to popularly used evaluation domains in both discrete- (Bellemare et al., 2013; Vinyals et al., 2017) and continuous-action (Brockman et al., 2016; Tunyasuvunakool et al., 2020) settings. However, these benchmarks are restricted to single-task training—each task is designed to be learned *in isolation*. Consequently, work in multi-task and continual RL has resorted to ad hoc evaluation settings, slowing down progress. Recent efforts have sought to bridge this gap by creating evaluation domains with multiple tasks that share a common structure that is (hopefully) transferable across the tasks. One example varied dynamical system parameters (e.g., gravity) of continuous control tasks (Henderson et al., 2017). Other work created a grid-world evaluation domain with tasks of progressive difficulty (Chevalier-Boisvert et al., 2019). In the continual learning setting, a recent benchmark was proposed based on multi-agent coordination (Nekoei et al., 2021). In the context of robotics, large sets of tasks have recently been created for evaluating multi-task, continual, and meta-learning algorithms (Yu et al., 2019; James et al., 2020; Wołczyk et al., 2021).

Despite this recent progress, it remains unclear exactly *what* an agent can transfer between tasks in these benchmarks, and so existing algorithms are typically limited to transferring neural net parameters in the hopes that they discover reusable information. Unlike these existing benchmarks, CompoSuite is designed around a set of shared components, such that the commonalities across tasks are precisely understood, following equivalent efforts from the supervised setting (Bahdanau et al., 2018; Lake & Baroni, 2018; Sinha et al., 2020; Vedantam et al., 2021). A similar benchmark was recently proposed for evaluating temporal (instead of functional) compositionality (Gur et al., 2021). Another related benchmark procedurally created robotics tasks by varying dynamical parameters to study causality in RL (Ahmed et al., 2021), but considered a single robot arm and continuous variations in the physical properties of objects.

## 3 Background on functionally compositional RL

Many complex problems can be solved by considering smaller components of the problem separately and combining their solutions. This type of knowledge composition has long been considered a promising direction in AI. A recently proposed approach to composition is the notion of functional compositionality of RL tasks (Mendez et al., 2022).



⟨IIWA, box, no_obstacle, pick-and-place⟩ ⟨Jaco, hollow_box, object_door, push⟩ ⟨Gen3, plate, goal_wall, trash_can⟩ ⟨Panda, dumbbell, object_wall, shelf⟩

Figure 1: Initial conditions of four CompoSuite tasks, containing all elements of each compositional axis. Robots: `IIWA`, `Jaco`, `Gen3`, and `Panda`. Objects: `box`, `hollow_box`, `plate`, and `dumbbell`. Obstacles: `no_obstacle`, `object_door`, `goal_wall`, `object_wall`. Objectives: `pick-and-place`, `push`, `trash_can`, `shelf`.

An RL problem is given by a Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $R(s, a)$ is the reward function, $T(s' \mid s, a)$ is the transition function, and $\gamma$ is the reward discount factor. Solving the MDP involves finding the policy $\pi^*$ that maximizes the expected returns $E_\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$.

In multi-task RL, the agent is faced with a set of MDPs $\mathcal{T} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_N\}$, and its goal is to solve all the MDPs. In particular, functionally compositional MDPs are assumed to be compositions of different subsets from a set of $k$ shared subproblems $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$. To solve the MDPs, the agent could find the set of solutions to these subproblems $M = \{m_1, ..., m_k\}$, such that any MDP $\mathcal{M}_i \in \mathcal{T}$ can be solved by combining the correct subproblem solutions from $M$. In the context of robotics, each of these subproblems can be thought of as a function in a processing pipeline, such as obstacle detection, object recognition, planning, and control. This idea can be formalized as a graph $G = \{\mathcal{V}, \mathcal{E}\}$, where the vertices $\mathcal{V} = \mathcal{F} \bigcup \breve{\mathcal{S}} \bigcup \breve{\mathcal{A}}$ correspond to subproblem solutions $\mathcal{F}$, state spaces $\breve{\mathcal{S}} = \text{unique}\left(\left\{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(N)}\right\}\right)$, and action spaces $\breve{\mathcal{A}} = \text{unique}\left(\left\{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}\right\}\right)$. Then, any MDP $\mathcal{M}_i \in \mathcal{T}$ is specified as a pair of nodes $(\mathcal{S}_i, \mathcal{A}_i)$ in the graph, and the optimal policy $\pi_i^*$ can be represented as a path on the graph.

A natural solution to this problem is to learn individual functions (e.g., neural net modules) to solve each subproblem $m_i \in M$. However, the problem setting is not restricted to this specific choice. Moreover, while this notion is related to hierarchical RL, it departs from *temporal* compositions and studies general *function* compositions.

## 4 The CompoSuite benchmark for compositional RL

CompoSuite is a simulated robotic manipulation benchmark designed to study the ability of RL algorithms to learn functional decompositions of the solutions to the tasks, yet more broadly applicable to multi-task and continual RL. The key idea is to build the tasks compositionally, so that 1) we can create combinatorially many (distinct) tasks, and 2) tasks are explicitly compositionally related. Sampled tasks are illustrated in Figure 1 and all tasks are shown in Appendix A.

### 4.1 Task design

CompoSuite is implemented on top of `robosuite` (Zhu et al., 2020), a framework for the design of new simulated robotics environments in MuJoCo (Todorov et al., 2012). Concretely, CompoSuite is built around four compositional axes, which represent common modules that typically make up robotic manipulation programming pipelines: object grasp-pose detection, obstacle avoidance, task planning, and low-level motor control. There are four elements of each type (i.e., for each axis), so that combining them yields a total of 256 tasks; this represents the largest discrete set of tasks in a multi-task RL benchmark to date, yet remains computationally feasible. Within each axis, elements are designed such that a policy that succeeds at one task is very unlikely to succeed at another task—and the optimal policy for one task is even less likely to be optimal for another. To focus on the property of compositionality, variations within each axis are discrete, such that an agent can not trivially interpolate between the elements within one axis. Each environment contains two bins: one for objects and one for targets. This standardization encourages the agents to find the commonalities between the tasks. The reward functions are crafted to facilitate learning each individual task.

#### 4.1.1 Task components

**Robots** As the first axis of CompoSuite, we use simulated versions of commercially available robotic manipulators: KUKA's `IIWA`, Kinova's `Jaco`, Franka's `Panda`, and Kinova's `Gen3`. These manipulators vary in sizes, kinematic configurations, and position and torque limits, leading to semantic discrepancies between their observations and actions that require the agent to specialize its control policy for each arm. Consequently, a policy that works on one robot arm

cannot be directly applied to another arm. To ensure compatibility with existing multi-task RL methods, we use arms with seven degrees of freedom (7-DoF). All arms use the Rethink Robotics two-finger gripper to manipulate objects.

**Objects** We next consider four objects of distinct shapes that require orthogonal grasping orientations. The `box` is a cuboid that can be picked up from the top. The `hollow_box` resembles an open package, with a size sufficiently large that the gripper cannot grasp it by both sides like the `box`, and must instead grip one of its edges. The `dumbbell` is placed upright, and its weights are larger than the gripper, and so it can only be grasped horizontally by the bar. The `plate`'s diameter is also greater than the gripper size and therefore can only be grasped horizontally by the edge.

**Obstacles** The third axis of variation in CompoSuite is a set of four obstacles that block off distinct areas for trajectory planning. The `object_wall` is a brick wall placed between the robot and the object, while the `object_door` is a similarly placed doorway between two brick walls. These two obstacles require avoiding opposite regions of the space while reaching for the object. The `goal_wall` is also a brick wall, but is placed between the left and right bins, blocking the direct path to goal after grasping the object. Additionally, we consider tasks with `no_obstacle`.

**Task objectives** The final compositional axis is a set of different task objectives, each of which requires a unique sequence of steps for successful completion. The objectives are to `pick-and-place` an object into the right bin, `push` the object from the left to the right bin, drop the object into a `trash_can`, and place the object on a `shelf`.

Thanks to combinatorial explosion, there are 256 possible combinations of these components, leading to a set of 256 highly varied tasks. By design, each task requires a unique policy, but we know exactly how tasks relate to one another, enabling researchers to extract insights about the kind of compositionality that deep RL methods exhibit.

#### 4.1.2 Observation and action spaces

The observation space is split into the following factors, tied to the task components described in the previous section:

- *Robot observation* The proprioceptive portion of the observation space includes the sine and cosine of the robot's joint positions, its joint velocities, end effector pose, finger positions, and finger velocities.
- *Object observation* The agent observes both the absolute position and orientation of the object in world coordinates, as well as its position and orientation with respect to the robot's end effector. Note that this observation deliberately does not give away any information that distinguishes objects from one another (e.g., their geometric properties).
- *Obstacle observation* The agent also observes the absolute and relative positions and orientations of the obstacles. Similarly, this does not give away what the free space of the environment is (e.g., `object_wall` and `object_door` are always placed in the same location, but they block off opposite parts of the space).
- *Goal observation* The agent is also given the absolute and relative position and orientation of the goal, as well as the relative position of the goal with respect to the object. However, for simplicity, `pick-and-place`, `trash_can`, and `shelf` tasks are considered solved at any arbitrary location in the target region (e.g., the right bin or the shelf).
- *Task observation* The agent may also be given access to a multi-hot indicator that identifies each of the components of the task (i.e., the robot, object, obstacle, and objective). This is used as a task descriptor for multi-task training.

The action space is eight-dimensional, with the first seven dimensions providing target joint angles. Under the hood, a proportional-derivative (PD) controller executes the motor commands that follow the joint positions provided by the agent. The eighth dimension is a binary action that indicates whether the gripper should be open or closed.

#### 4.1.3 Reward functions

While CompoSuite supports sparse rewards for successful completion, this leads to an extremely hard exploration problem. Consequently, to isolate the problem of multi-task compositional learning, we provide a crafted reward that encourages exploration in stages, such that each stage leads the agent to a state that is closer to task completion.

During the initial **reach** stage, the agent is rewarded for reducing the distance from the gripper to the object. This stage terminates once the agent **grasps** the object, which gives a binary reward. These two initial stages are common to all objectives. In all tasks except for `push`, the agent is next rewarded for **lifting** the object up to a given height. In the case of `shelf` tasks, the agent is then encouraged to **align** the gripper with the horizontal plane, facing the shelf. The next stage rewards the agent for **approaching** the right bin (or the goal, in `push` tasks) based on the horizontal distance. In `pick-and-place` tasks, the reward then encourages the agent to **lower** the object down to the bin. In `trash_can` tasks, the agent is instead rewarded for **dropping** the object while above the trash can with a binary reward.

The final stage is a binary **success** reward. `pick-and-place` tasks succeed if the object is in the bin and the robot is near the object; this latter constraint differentiates `pick-and-place` and `trash_can` tasks. `push` tasks are solved if the object is near the goal location. The agent succeeds on `trash_can` tasks if the object is inside the trashcan and the gripper is *not*. The success criterion for `shelf` tasks is that the object is on the shelf.

The maximum possible reward is $R = 1$ and is only attained upon successfully executing the task. Table 1 summarizes the stages of each task objective, and precise formulas for the task objective rewards are included in Appendix B.

Table 1: Reward stages per task objective. The agent is encouraged to solve each stage before moving to the next.

| Task | Stages |
|---|---|
| `pick-and-place` | reach → grasp → lift → approach → lower → success |
| `push` | reach → grasp → approach → success |
| `trash_can` | reach → grasp → lift → approach → drop → success |
| `shelf` | reach → grasp → lift → align → approach → success |

#### 4.1.4 Episode initialization and termination

Upon initialization of each new episode, the graspable object is placed in a random location of the left bin. In tasks that contain an obstacle, the object's initial location is restricted to the regions of the space that would explicitly require the robot to circumvent the obstacle. The goal locations are initialized in the right bin, and the robot arm is initialized at a fixed position with the gripper facing downward. Sampled initial conditions are displayed in Figure 1.

Each episode terminates after $H = 500$ time steps. In addition, `push` tasks terminate if the robot lifts the object more than a set (small) threshold above the table, in order to avoid success by the robot executing a pick-and-place strategy.

### 4.2 Evaluation settings

CompoSuite evaluates agents for training speed and final performance over a subset of *training* tasks, akin to training sets in supervised single-task settings. While this is a measure of training performance, it corresponds to the standard evaluation setting of the large majority of works in RL. After training, agents are evaluated on a *test* set of unseen tasks. Both of these evaluations explore the ability of agents to discover compositional properties of the tasks.

#### 4.2.1 Metrics

Agents are evaluated according to two metrics. For an agent evaluated over $N$ tasks, with $M$ evaluation trajectories for each task, each trajectory of length $H$, the average metrics are computed as follows:

**Return** The standard cumulative returns:

$$\overline{R} = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{t=1}^{H} R_i(s_t, a_t) \ . \tag{1}$$

This is the usual evaluation criterion for RL works and directly relates to the optimization objective.

**Success** The per-task success rate:

$$\overline{S} = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \max_{t \in [1,H]} \mathbb{1}[R_i(s_t, a_t) = 1] \ , \tag{2}$$

where $\mathbb{1}$ is the indicator function. Note that a trajectory is successful if at *any* time the agent is in a success state.

#### 4.2.2 Evaluation on training tasks

The agent is first evaluated on the tasks that it trains on. An agent that is capable of extracting the compositional properties of the tasks should be able to achieve transfer across the tasks. Ideally, this transfer should translate to both faster convergence in terms of the number of samples required to learn, as well as higher final performance after convergence. In particular, agents in this setting should be compared against an equivalent single-task agent that uses the same training mechanism but does so individually on every task, without any notion of shared knowledge.

#### 4.2.3 Evaluation on test tasks

The key property that CompoSuite assesses is the ability of approaches to combine trained components in novel combinations to handle new tasks. Following Mendez et al. (2022), this can take the following two forms:

**Zero-shot generalization with task descriptors** If the agent is given the multi-hot indicators described in Section 4.1.2, then it could (in principle) solve new, unseen tasks without any training on them. This would be possible only if the agent learns the compositional structure of the tasks and is able to combine its existing components into a solution to the new task. Intuitively, after learning 1) the `pick-and-place` task with the `box` object avoiding the `object_door` obstacle using the `IIWA` arm, and 2) the `push` task with the `plate` object avoiding the `object_wall` obstacle using the `Panda` arm, if the agent knows how each of the components relates to the overall task, it could for example swap the `IIWA` and `Panda` arms and solve the opposite tasks without any additional training.

**Few-shot generalization without task descriptors** Alternatively, the agent might not be informed of which components make up the current task, and be required to discover this information through experience. The goal of the agent should then be to discover this information as rapidly as possible in order to solve the new task with little experience.

#### 4.2.4 ACCESS TO STATE DECOMPOSITION

The modular architectures of Devin et al. (2017) and Mendez et al. (2022) require knowledge about which components of the observation affect which parts of the architecture. While this information is readily available in CompoSuite, fair performance comparisons would require noting whether the agent is given this decomposition of the state space. Note that both zero-shot and few-shot settings could be targeted with or without the state decomposition.

#### 4.2.5 SAMPLE OF TRAINING TASKS

Understanding the compositional capabilities of RL algorithms requires a careful study of the sample of combinations (i.e., tasks) that is provided to the agent for training. We propose the following evaluation settings:

**Uniform sampling** In the simplest setting, the training tasks are sampled uniformly at random, and the agent is asked to generalize to all possible combinations of the seen components. The agent therefore must learn to combine its knowledge in different ways after having seen each component in various combinations.

**Restricted sampling** In this much harder setting, the training is restricted to a single task for one of the components and many tasks for other components (e.g., in CompoSuite\`IIWA`, the agent sees only one `IIWA` task and must generalize to all other `IIWA` tasks). This is akin to Experiment 3 in the work of Lake & Baroni (2018), which demonstrated that this is an onerous problem even in the supervised setting. While a complete evaluation would require various choices of restricted arms, objects, obstacles, and objectives, as an initial step we propose four evaluation settings: CompoSuite\`IIWA`, CompoSuite\`hollow_box`, CompoSuite\`object_wall`, and CompoSuite\`pick-and-place`. These restricted elements were empirically observed to be easier to learn than others by the single-task agents during development. Restricting access to an "easy" element enables the zero-shot evaluation to focus on generalization, without conflating it with the difficulty of the task itself. As an exception, the CompoSuite\`object_wall` setting was selected over CompoSuite\`no_obstacle` because generalizing to the `no_obstacle` element is trivial from any other obstacle.

**Smaller-scale benchmarks** While large benchmarks like CompoSuite are appealing for studying multi-task RL at scale, developing ideas in such large task sets is often (unfortunately) prohibitively time-consuming. Given the compositional nature of CompoSuite, it is straightforward to extract smaller-scale benchmarks that maintain the properties of the full-scale benchmark. For example, CompoSuite$\cap$`IIWA` considers only the 64 `IIWA` tasks. Interestingly, such reduced benchmarks permit studying the difficulty of generalization across certain axes (e.g., if an agent can transfer knowledge across objects but not across robots, then it would perform much better on CompoSuite$\cap$`IIWA` than on the full CompoSuite). Following the rationale of the restricted setting, we propose to evaluate agents on: CompoSuite$\cap$`IIWA`, CompoSuite$\cap$`hollow_box`, CompoSuite$\cap$`no_obstacle`, and CompoSuite$\cap$`pick-and-place`.

## 5 BENCHMARKING EXISTING RL METHODS ON COMPOSUITE

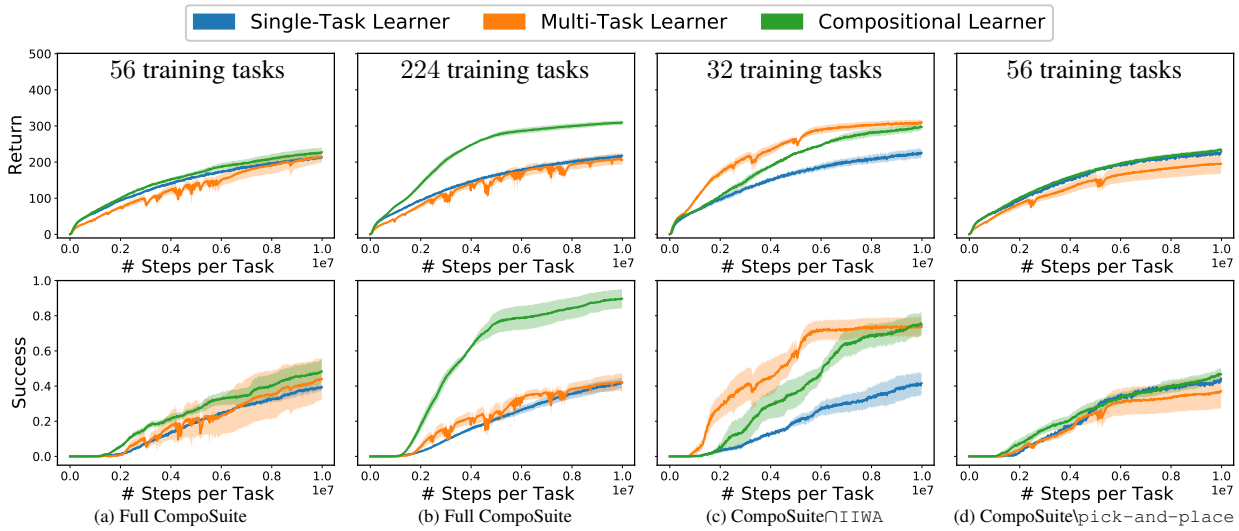
The empirical evaluation in this section had two primary objectives. First, to demonstrate that CompoSuite is a useful evaluation benchmark in terms of: 1) existing algorithms making progress toward solving the problems, 2) the tasks exhibiting compositional properties, and 3) existing approaches leaving substantial room for improvement in performance. Second, to provide benchmarking results of existing algorithms for future work to leverage[2].

### 5.1 EXPERIMENTAL SETTING

The underlying RL algorithm used for all our evaluations was the proximal policy optimization (PPO; Schulman et al., 2017) implementation in `Spinning Up` (Achiam, 2018). Appendix C describes critical modifications that were necessary for learning the tasks. Building upon this base algorithm, we evaluated the following three agents:

- **Single-task** agents that trained on each task individually, without any knowledge-sharing across tasks. Lack of sharing precludes these agents from generalizing to unseen tasks, and so they were only evaluated on training tasks. We also withheld the task descriptor from the observation, as it would appear as a constant to each single-task agent.
- **Multi-task** agents that trained a *shared model* for all tasks, using the task descriptor in the observation to help differentiate between tasks and learn to specialize the policy for each task. Given the need for the multi-task agent to encode multiple policies in a single model, we gave this agent a larger capacity than an individual single-task agent.

[2]Trained models are available at: https://github.com/Lifelong-ML/CompoSuite-Data.




(a) Full CompoSuite (b) Full CompoSuite (c) CompoSuite∩IIWA (d) CompoSuite\pick-and-place

Figure 2: Evaluation on training tasks. (a,b) When trained on all of CompoSuite, the multi-task agent was not capable of accelerating the learning substantially with respect to the single-task agent according to either metric. (b) The compositional agent, when trained on a larger set of tasks, performed noticeably better, demonstrating that leveraging the compositional structure of CompoSuite leads to improved training performance. (c) On the smaller-scale CompoSuite∩IIWA, both the multi-task and the compositional agents were able to outperform the single-task agent under both metrics. This shows that sharing knowledge across tasks with a single robot arm is easier than across different robot arms when not explicitly leveraging compositionality. (d) On CompoSuite\pick-and-place, a single pick-and-place task was sampled for training, and all other tasks were sampled from the remaining task objectives. Results are qualitatively similar to those on the full CompoSuite, as expected—major differences are expected in *zero-shot* performance. Y-axes span the attainable ranges and shaded regions represent std. errors across three seeds.

- **Compositional** agents that constructed a *different* model for each task from a set of *shared* components. We used a variant of the modular network of Mendez et al. (2022) that establishes each policy from a set of modules, with one module for each robot, object, obstacle, and objective. The relevant state component from Section 4.1.2 was fed as input to each module, and the task descriptor was used to select the correct modules. Each module was represented by an MLP, whose output was fed to the next module: obstacle → object → objective → robot (see Appendix C). For fairness, the *overall* number of parameters across modules was equivalent to that of multi-task agents.

Each agent was evaluated on the full CompoSuite benchmark and on all the suggested smaller-scale and restricted benchmarks. In each of these settings, a subset of the tasks was given to the agents for training, and the agents were evaluated for their speed and final performance over the training tasks. After training, the multi-task and compositional agents were additionally evaluated for their ability to solve unseen tasks without any additional training by leveraging the task descriptors. Additional details are provided in Appendix C.

### 5.2 Evaluation of baselines on the full CompoSuite benchmark

We first evaluated the agents on the main CompoSuite benchmark, uniformly sampling tasks for training; learning curves are presented in Figures 2a and 2b. After training for 10 million time steps on each task, the single-task agent had a success rate of around 40%. When the training set was a small portion of the whole set of tasks, the multi-task and compositional agents only slightly improved upon the single-task agent. However, when training on a larger set of tasks, the compositional agent learned much faster, achieving approximately twice as much success. In contrast, multi-task results did not improve with the larger training set. This suggests that the multi-task agent is not appropriately sharing knowledge across tasks, and instead separately allocates capacity in the network to different tasks. As more tasks are seen, capacity is progressively exhausted. Instead, the compositional agent shares components appropriately, and additional training tasks improve the agent's ability to leverage these commonalities. This demonstrates that CompoSuite tasks are indeed compositionally related, and that exploiting these relationships leads to improved performance.

After training, we evaluated the agents on the CompoSuite tasks that they did *not* train on. Intuitively, an agent that correctly decomposes the tasks should achieve high performance on these test tasks by adequately recombining its learned components. Results in the first two columns of Table 2 show that the learners struggled to generalize to

Table 2: CompoSuite zero-shot generalization. All agents struggled to generalize to the majority of the holdout tasks except for the compositional agent trained on 224 tasks; std. errors across three seeds reported after the ±.

| | CompoSuite 56 Tasks | | CompoSuite 224 Tasks | | CompoSuite ∩IIWA | | CompoSuite \pick-and-place | |
|---|---|---|---|---|---|---|---|---|
| | Multi-task | Comp. | Multi-task | Comp. | Multi-task | Comp. | Multi-task | Comp. |
| Return | $115.79_{\pm 18.0}$ | $64.26_{\pm 7.5}$ | $201.74_{\pm 26.9}$ | $302.44_{\pm 12.2}$ | $232.79_{\pm 17.5}$ | $79.85_{\pm 19.2}$ | $74.61_{\pm 10.}$ | $16.63_{\pm 6.7}$ |
| Success | $0.18_{\pm 0.1}$ | $0.08_{\pm 0.0}$ | $0.41_{\pm 0.0}$ | $0.88_{\pm 0.1}$ | $0.49_{\pm 0.1}$ | $0.12_{\pm 0.1}$ | $0.09_{\pm 0.0}$ | $0.01_{\pm 0.0}$ |

unseen tasks when trained on 56 tasks but performed remarkably well when trained on 224 tasks. With the smaller training set, even though the *training* performance was similar for both approaches, the compositional agent achieved substantially worse *zero-shot* performance. This demonstrates that, while the compositional approach can indeed capture the compositional properties of the tasks, this capability requires observing a large portion of the tasks.

One important question is whether the multi-task agent was automatically learning compositional knowledge that allowed it to solve unseen tasks. The alternative explanation would be that the agent instead found similar tasks in the training set and used the policy for those for generalization. We therefore set up a simple experiment, finding the most similar training task to each test task and using its policy to predict zero-shot performance. Concretely, for every test task $\mathcal{M}_i$ with some zero-shot success, we found the training task $\mathcal{M}_{i'}$ whose policy $\pi_{i'}$ performed best on $\mathcal{M}_i$; this would have been the best policy to choose, and so one would expect the performance of $\pi_{i'}$ to correlate to that of $\pi_i$. To reduce computation, we considered only tasks $i'$ that varied in a single element from $i$. We found that the coefficients of determination between the policies' success rates were very low: $\mathbf{R^2 = 0.19}$ and $\mathbf{R^2 = 0.03}$ for the multi-task and compositional agents, respectively. This shows that the generalization of the multi-task learner was unlikely to come from using trained policies for different tasks, but rather from leveraging the compositional properties of the tasks.

### 5.3 Evaluation of baselines on the smaller-scale CompoSuite∩IIWA benchmark

Next, we evaluated the three baseline agents on the reduced benchmarks, in order to 1) propose a computationally cheaper setting to facilitate progress and 2) shed light on the relative difficulty of generalizing across different CompoSuite axes. Learning curves on the training tasks for the CompoSuite∩IIWA benchmark are included in Figure 2c, and the remaining curves are in Appendix D. The relative performance of the compositional agent with respect to the single-task agent was close to that obtained after training on *over 200 tasks* for the full CompoSuite, demonstrating that the agent is capable of discovering the compositional structure of this reduced benchmark with far fewer training tasks. On the other hand, the multi-task agent performed noticeably better in CompoSuite∩IIWA and CompoSuite∩pick-and-place. This provides evidence that these two axes (robot and objective) are harder to generalize across, as one would intuitively expect. Additional evidence toward this hypothesis is given in Section 5.5.

We further assessed the performance of the agents on the unseen IIWA tasks, and report the results in the third column of Table 2. The multi-task agent achieved notably high performance but the compositional agent was incapable of generalizing. The poor generalization of the compositional agent was likely due to the small number of training tasks: since the agent only trains each module on the subset of tasks that shares that module, each parameter was trained on a small number of tasks which was insufficient for zero-shot generalization. Results on the remaining settings in Appendix D exhibit lower multi-task performance and similar compositional performance in most cases.

### 5.4 Evaluation of baselines on the restricted CompoSuite\pick-and-place benchmark

The results presented so far consider a relatively simple compositional problem: the agent is trained on multiple combinations of all components, and is expected to generalize to new combinations. These previous results already expose the shortcomings of existing approaches in the compositional setting. However, future approaches that solve these simple settings would still fall short from achieving the full compositional capabilities we expect from them. We would hope that agents could learn components that generalize to unseen tasks even if these components are only seen in one single combination. To study this setting, we evaluated the three agents on the restricted task samples. Results on the training tasks for the CompoSuite\pick-and-place benchmark, which includes exactly one pick-and-place task, are shown in Figure 2d, and Appendix D includes results for remaining settings. In many cases, performance was close to that of the full benchmark using 56 tasks, because the training distributions are similar: there are 55 combinations of 15 components (plus one restricted task), compared to 56 combinations of 16 components. The fourth column of Table 2 summarizes the pick-and-place zero-shot performance of the agents on this restricted setting. Both agents failed to generalize to the unseen pick-and-place tasks (Appendix D shows the same trend for the remaining settings). Table 3 shows that the small amount of generalization the multi-task agent achieved was almost entirely on tasks from the same robot arm that was used in the single pick-and-place training task.

Table 3: Zero-shot generalization (success rate) for the multi-task agent on CompoSuite\pick-and-place, separated by tasks that share (or not) each element with the trained pick-and-place task (e.g., if the training pick-and-place task used the IIWA arm, IIWA tasks go in the "trained" column and non-IIWA tasks go in the "untrained" column).

| Element | Trained | Untrained |
|---|---|---|
| robot | $0.30_{\pm 0.10}$ | $0.03_{\pm 0.02}$ |
| object | $0.14_{\pm 0.04}$ | $0.08_{\pm 0.04}$ |
| obstacle | $0.14_{\pm 0.04}$ | $0.08_{\pm 0.03}$ |

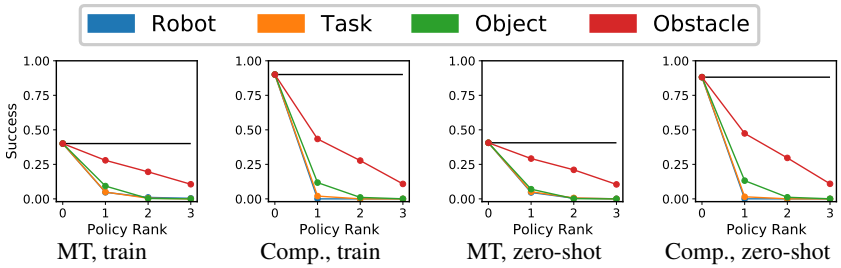



Figure 3: Performance given incorrect task descriptors, with a single changed component: position 0 corresponds to the correct descriptor, and positions $j > 0$ correspond to the $j$-th ranked descriptors of each type. Using the wrong descriptors leads to poor performance, confirming the diversity of CompoSuite tasks.

### 5.5 Empirical analysis of CompoSuite learnability and diversity

We now shift focus to verifying two important properties of CompoSuite: that the large majority of tasks are learnable by current RL mechanisms, and that the tasks are not only compositional, but also highly varied.

**Learnability of tasks** If CompoSuite tasks were unsolvable by current RL methods, that would conflate the difficulty of compositional reasoning with the difficulty of solving RL tasks. To validate that this is not the case, for every task, we found the best performing agent across all those trained in Figure 2 (taking the maximum across experiments *and* random seeds). For any task with a score of 0, we have no evidence that the task is learnable, because no agents solved it to any extent. The result of this computation is shown for all tasks in Appendix E. Only one task received a score of 0, indicating that it *may* be unlearnable; this demonstrates that CompoSuite tasks are attainable for current RL methods.

**Diversity of tasks** Another valid concern is that it might be possible for the agent to solve multiple tasks with a single policy if the tasks are very similar, implying that compositional reasoning is not necessary for generalization. To verify that this is not the case in CompoSuite, after training the multi-task and compositional agents over 224 tasks, we evaluated their performance if they were given the *incorrect* task descriptor. In particular, for a given task $\mathcal{M}_i$, we tested the performance of the agent on task $\mathcal{M}_i$ if given the descriptor for all tasks $\mathcal{M}_{i'}$ that varied in a single component from $\mathcal{M}_i$ (i.e., the tasks most similar to $i$). We sorted the performances with these incorrect descriptors separately for each axis, finding the rank of each incorrect component (e.g., the rank-two robot for task $\mathcal{M}_i$ is the robot that achieved the second-best performance when used as the descriptor for task $\mathcal{M}_i$), and averaged the sorted performances. The results, summarized in Figure 3, show that using the incorrect robot, objective or object descriptor consistently leads to substantially degraded performance, particularly for the compositional agent; this means that the agents are specializing to each of the components. Using the incorrect obstacle descriptor has a much smaller impact, particularly for the multi-task agent, which suggests that the multi-task agent learns a policy that is somewhat agnostic to the obstacles. We conclude that CompoSuite cannot be solved without specializing the policies to each task. Additionally, varying the robot arm causes a drastic drop in performance, demonstrating that solving tasks with varied robots is a challenging problem, yet existing benchmarks are limited to a single robot arm.

## 6 Scope, limitations, and extensions

CompoSuite is designed as a benchmark for studying the compositional properties of multi-task RL algorithms. As such, while it can be used to investigate multiple other problems, it is not intended to cover the spectrum of open questions in multi-task RL. This section discusses limitations and potential extensions to the use of CompoSuite.

**Reliance on PPO** To provide a fair comparison across single-task and multi-task learners, we used PPO (Schulman et al., 2017) as the base RL algorithm for all agents, built off of the Spinning Up implementation (Achiam, 2018). While future research can use any base learning method, only evaluations that use the same PPO implementation could fairly compare against the benchmarking results presented here.

**Input space** The input space used in our evaluation is a 94-dimensional symbolic description of the environment grounded in the system dynamics. However, there is also broad interest from the robot learning community in RL with richer observations (e.g., visual inputs). While such an evaluation falls outside of the scope of this work, the benchmark implementation allows users to request a multi-camera visual observation instead of the low-dimensional observation.

**Task descriptors** Part of the observation space is a multi-hot indicator that describes the components that make up the current task. While this permits assessing the interesting property of zero-shot compositional generalization, there are other questions that might benefit from withholding this information from the agent. As one example, the agent might be given only a task index that indicates *which* task is currently being solved, but not *how* it relates to other tasks. Alternatively, the agent could be given no indication of the current task at all and be required to extract it from data. Note that the symbolic observation does not contain sufficient information to unequivocally identify the task without task descriptors, and so the agent would need to extract this information from *trajectories* of interaction instead. On the other hand, the images in the visual observations do contain sufficient information to differentiate the tasks.

**Additional compositional axes** CompoSuite currently consists of four compositional axes, in part to constrain the size of the benchmark—the number of tasks is exponential in the number of axes. Expanding the benchmark with new axes could be necessary as deep RL methods scale further. As examples, new axes could include discretized object placements at initialization, goal positions, or variations in textures and color if using visual inputs.

**Other forms of composition** While CompoSuite was designed around functional composition as described in Section 3, the benchmark can also be used for other forms of composition. In particular, the standardization of the environments and the use of stage-wise rewards makes this a useful domain for evaluating skill discovery and sequencing. For example, the agent could learn skills for reaching a location, grasping an object, and lifting, all of which are useful for multiple CompoSuite tasks. Note that standard representations of skills would only work for one individual arm.

**Continual learning** Another very natural extension of CompoSuite is to use it in the continual learning setting, particularly when viewed as an online version of multi-task learning. The agent would be presented with CompoSuite tasks one after the next, and evaluated on all previously seen tasks. The goal of the agent would be to learn each new task as quickly as possible by leveraging accumulated knowledge, and to retain performance on the earlier tasks upon training on new tasks. Given the sequential nature of continual learning, it might be prohibitively expensive to train the agent over the full variant of CompoSuite, but the smaller-scale variants described in Section 4.2.5 would be feasible; existing approaches have already been evaluated on similar-length sequences of robotics tasks (Mendez et al., 2022).

**Sim2real transfer** Learning a multitude of tasks in simulation is a common strategy used to transfer policies from simulation to the real world (sim2real). Since CompoSuite uses simulated versions of four robot arms that are commercially available, it could additionally be leveraged to study this promising direction.

## 7 Conclusions

We introduced CompoSuite, a large-scale robotic manipulation benchmark for studying the novel problem of functionally compositional RL. CompoSuite leverages the power of combinatorics to create hundreds of highly diverse tasks, opening the door for multi-task RL at scale. In particular, CompoSuite is designed to study the ability of approaches to discover the decomposition of complex problems into simpler subproblems whose solutions can be combined to solve the overall task. Once appropriate components have been found, they could be combined to solve *new* RL problems that the agent has never trained on. Existing end-to-end and modular multi-task approaches show promising properties in some limited settings under CompoSuite, but we expose that they are far from solving the problem of compositional RL. Progress in that direction will enable RL approaches to automatically detect commonalities across diverse problems, leverage these commonalities to facilitate learning, and eventually handle far more complex tasks than is possible today.

### Acknowledgments

We would like to thank Spencer Solit for help in the early stages of development of CompoSuite, and David Kent for technical feedback about the robot simulator. We also thank the anonymous reviewers for their valuable comments. The research presented in this paper was partially supported by the DARPA Lifelong Learning Machines program under grant FA8750-18-2-0117, the DARPA SAIL-ON program under contract HR001120C0040, the DARPA ShELL program under agreement HR00112190133, and the Army Research Office under MURI grant W911NF20-1-0080.

## References

[1] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML-18)*, pp. 10–19, 2018.

[2] Joshua Achiam. Spinning up in deep reinforcement learning. https://github.com/openai/spinningup, 2018.

[3] Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Manuel Wuthrich, Yoshua Bengio, Bernhard Schölkopf, and Stefan Bauer. CausalWorld: A robotic manipulation benchmark for causal structure and transfer learning. In *9th International Conference on Learning Representations (ICLR-21)*, 2021.

[4] Ferran Alet, Tomas Lozano-Perez, and Leslie P. Kaelbling. Modular meta-learning. In *Proceedings of the 2nd Conference on Robot Learning (CoRL-19)*, pp. 856–868, 2018.

[5] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-16)*, pp. 39–48, 2016.

[6] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pp. 1726–1734, 2017.

[7] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? In *6th International Conference on Learning Representations (ICLR-18)*, 2018.

[8] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *Proceedings of the 35th International Conference on Machine Learning (ICML-18)*, pp. 501–510, 2018.

[9] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, 2013.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

[11] Michael Chang, Abhishek Gupta, Sergey Levine, and Thomas L. Griffiths. Automatically composing representation transformations as a means for generalization. In *7th International Conference on Learning Representations (ICLR-19)*, 2019.

[12] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *7th International Conference on Learning Representations (ICLR-19)*, 2019.

[13] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 6 (NeurIPS-93)*, pp. 271–278, 1993.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-09)*, pp. 248–255, 2009.

[15] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA-17)*, pp. 2169–2176, 2017.

[16] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13:227–303, 2000.

[17] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

[18] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. PathNet: Evolution channels gradient descent in super neural networks. *arXiv:1701.08734*, 2017.

[19] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. In *5th International Conference on Learning Representations (ICLR-17)*, 2017.

[20] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *9th International Conference on Learning Representations (ICLR-21)*, 2021.

[21] Izzeddin Gur, Natasha Jaques, Yingjie Miao, Jongwook Choi, Manoj Tiwari, Honglak Lee, and Aleksandra Faust. Environment generation for zero-shot compositional reinforcement learning. In *Advances in Neural Information Processing Systems 34 (NeurIPS-21)*, 2021.

[22] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA-18)*, pp. 6244–6251, 2018.

[23] Peter Henderson, Wei-Di Chang, Florian Shkurti, Johanna Hansen, David Meger, and Gregory Dudek. Benchmark environments for multitask learning in continuous domains. *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop*, 2017.

[24] Drew Arad Hudson and Christopher D. Manning. Compositional attention networks for machine reasoning. In *6th International Conference on Learning Representations (ICLR-18)*, 2018.

[25] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[26] Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pp. 2408–2418, 2018.

[27] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22 (NeurIPS-09)*, 2009.

[28] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML-18)*, pp. 2873–2882, 2018.

[29] Jorge A. Mendez and Eric Eaton. Lifelong learning of compositional structures. In *9th International Conference on Learning Representations (ICLR-21)*, 2021.

[30] Jorge A. Mendez, Harm van Seijen, and Eric Eaton. Modular lifelong reinforcement learning via neural composition. In *10th International Conference on Learning Representations (ICLR-22)*, 2022.

[31] Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *6th International Conference on Learning Representations (ICLR-18)*, 2018.

[32] Sarthak Mittal, Alex Lamb, Anirudh Goyal, Vikram Voleti, Murray Shanahan, Guillaume Lajoie, Michael Mozer, and Yoshua Bengio. Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules. In *Proceedings of the 37th International Conference on Machine Learning (ICML-20)*, pp. 6972–6986, 2020.

[33] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A Boolean task algebra for reinforcement learning. In *Advances in Neural Information Processing Systems 33 (NeurIPS-20)*, pp. 9497–9507, 2020.

[34] Hadi Nekoei, Akilesh Badrinaaraayanan, Aaron Courville, and Sarath Chandar. Continuous coordination as a realistic scenario for lifelong learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML-21)*, pp. 8016–8024, 2021.

[35] Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition. In *Advances in Neural Information Processing Systems 34 (NeurIPS-21)*, 2021.

[36] Scott Reed and Nando de Freitas. Neural programmers-interpreters. In *4th International Conference on Learning Representations (ICLR-16)*, 2016.

[37] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *6th International Conference on Learning Representations (ICLR-18)*, 2018.

[38] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[40] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *8th International Conference on Learning Representations (ICLR-20)*, 2020.

[41] Koustuv Sinha, Shagun Sodhani, Joelle Pineau, and William L. Hamilton. Evaluating logical generalization in graph neural networks. *arXiv:2003.06560*, 2020.

[42] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[43] Emanuel Todorov. Compositionality of optimal control laws. *Advances in Neural Information Processing Systems 22 (NeurIPS-09)*, pp. 1856–1864, 2009.

[44] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-12)*, pp. 5026–5033, 2012.

[45] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6, 2020.

[46] Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pp. 8687–8698, 2018.

[47] Benjamin Van Niekerk, Steven James, Adam Earle, and Benjamin Rosman. Composing value functions in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML-19)*, pp. 6401–6409, 2019.

[48] Ramakrishna Vedantam, Arthur Szlam, Maximillian Nickel, Ari Morcos, and Brenden M. Lake. CURI: A benchmark for productive concept learning under uncertainty. In *Proceedings of the 38th International Conference on Machine Learning (ICML-21)*, pp. 10519–10529, 2021.

[49] Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. In *9th International Conference on Learning Representations (ICLR-21)*, 2021.

[50] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the 345h International Conference on Machine Learning (ICML-17)*, pp. 3540–3549, 2017.

[51] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John P. Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[52] Maciej Wołczyk, Michał Zając, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Continual World: A robotic benchmark for continual reinforcement learning. In *Advances in Neural Information Processing Systems 34 (NeurIPS-21)*, 2021.

[53] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. In *Advances in Neural Information Processing Systems 33 (NeurIPS-20)*, pp. 4767–4777, 2020.

[54] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the 3rd Conference on Robot Learning (CoRL-19)*, 2019.

[55] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

## A VISUALIZATION OF ALL TASKS

CompoSuite consists of a total of 256 possible combinations of elements, each representing a separate task. Figures 4– 7 show each of the different robot arms in action solving the enormous diversity of tasks in CompoSuite.

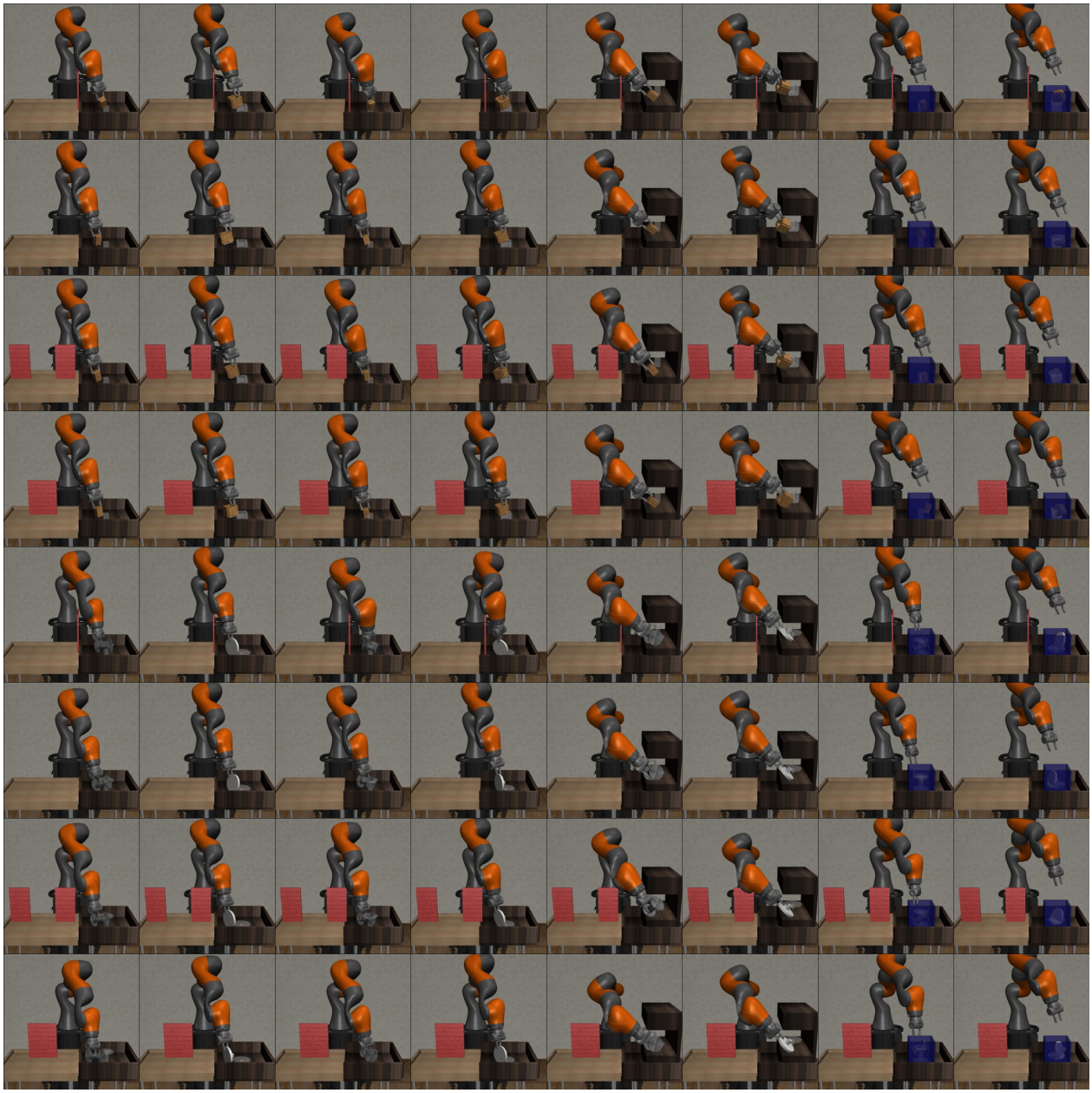


Figure 4: Visualization of the 64 `IIWA` tasks.

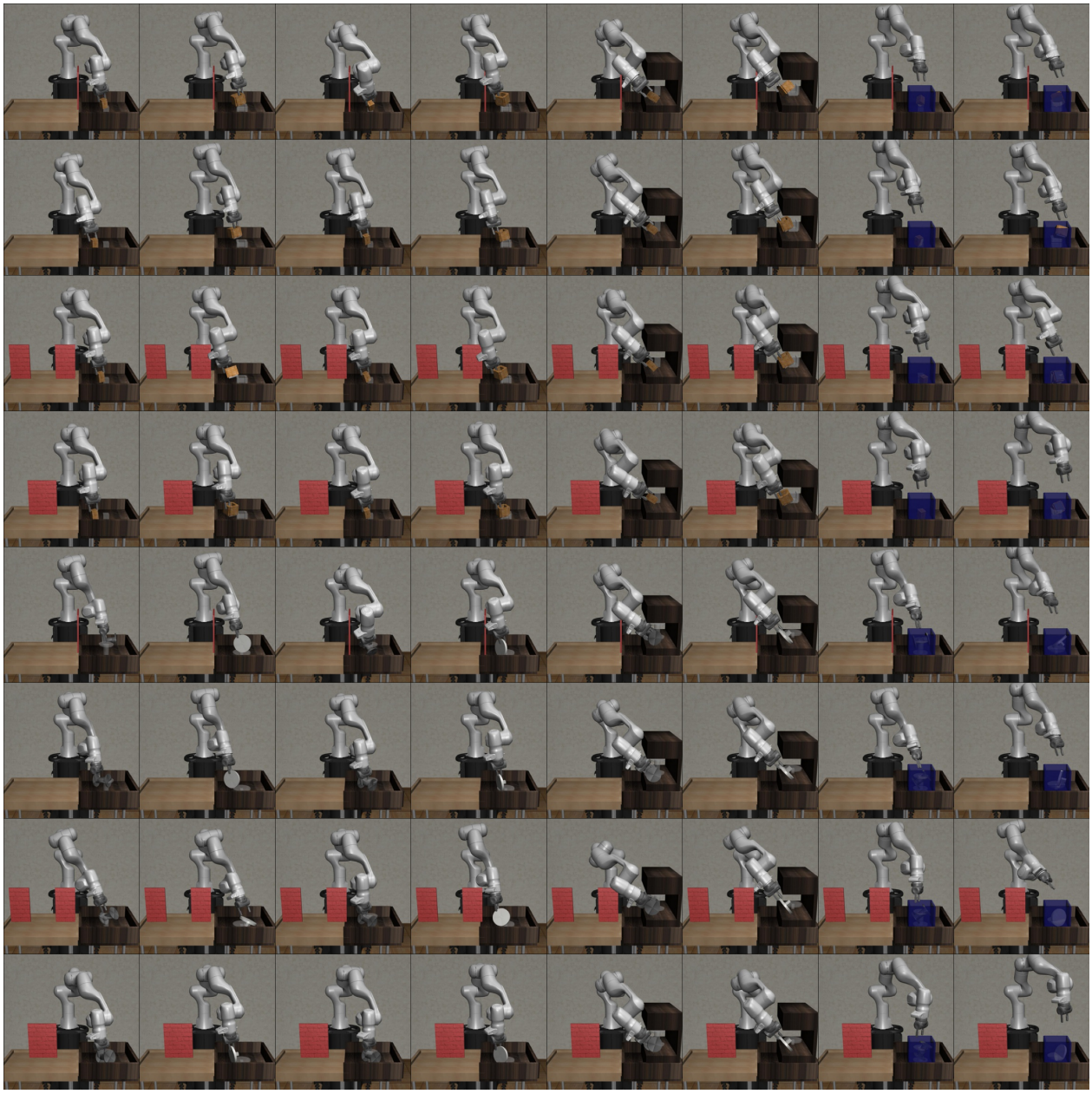


Figure 5: Visualization of the 64 `Panda` tasks.

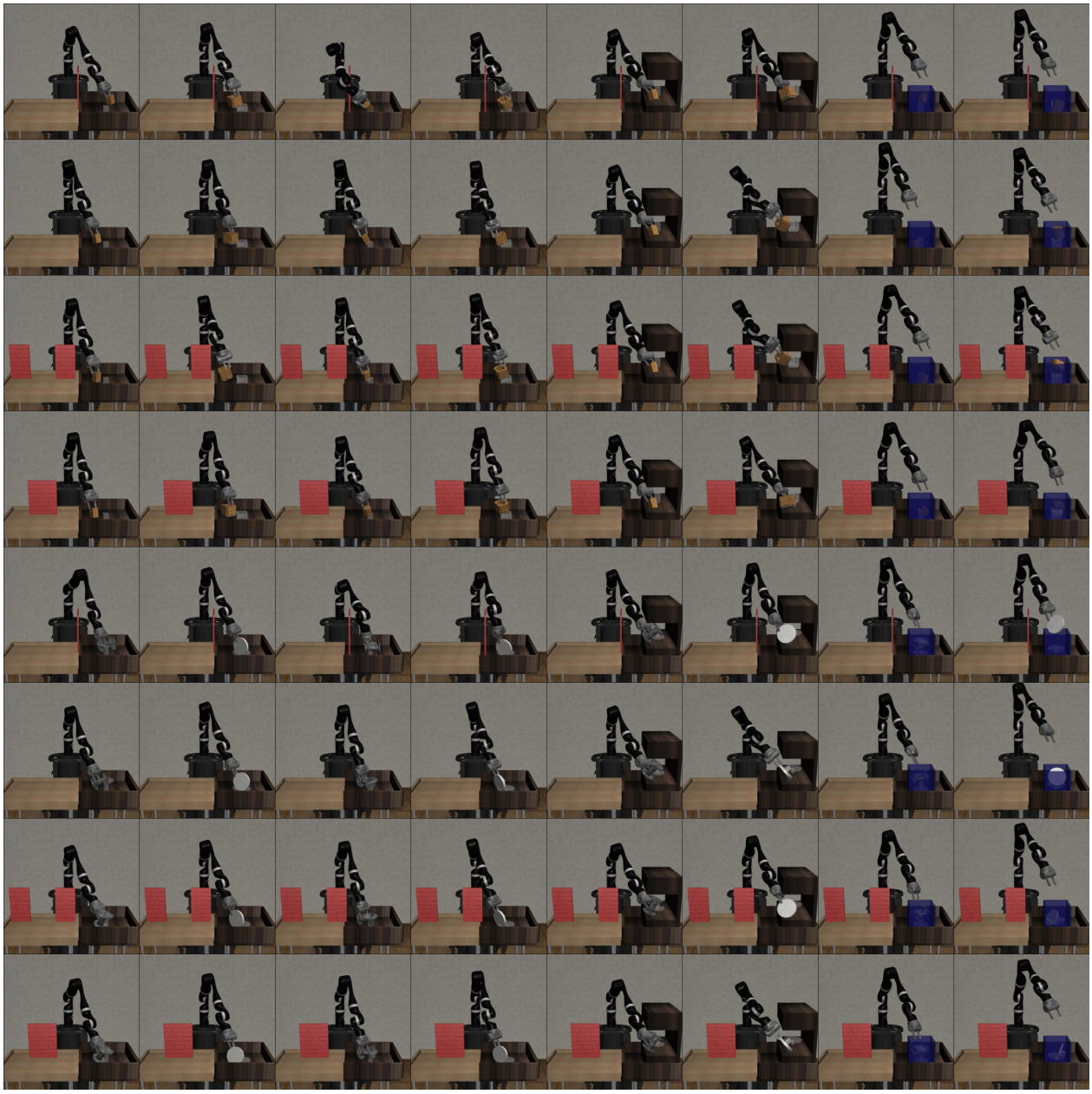


Figure 6: Visualization of the 64 `Jaco` tasks.

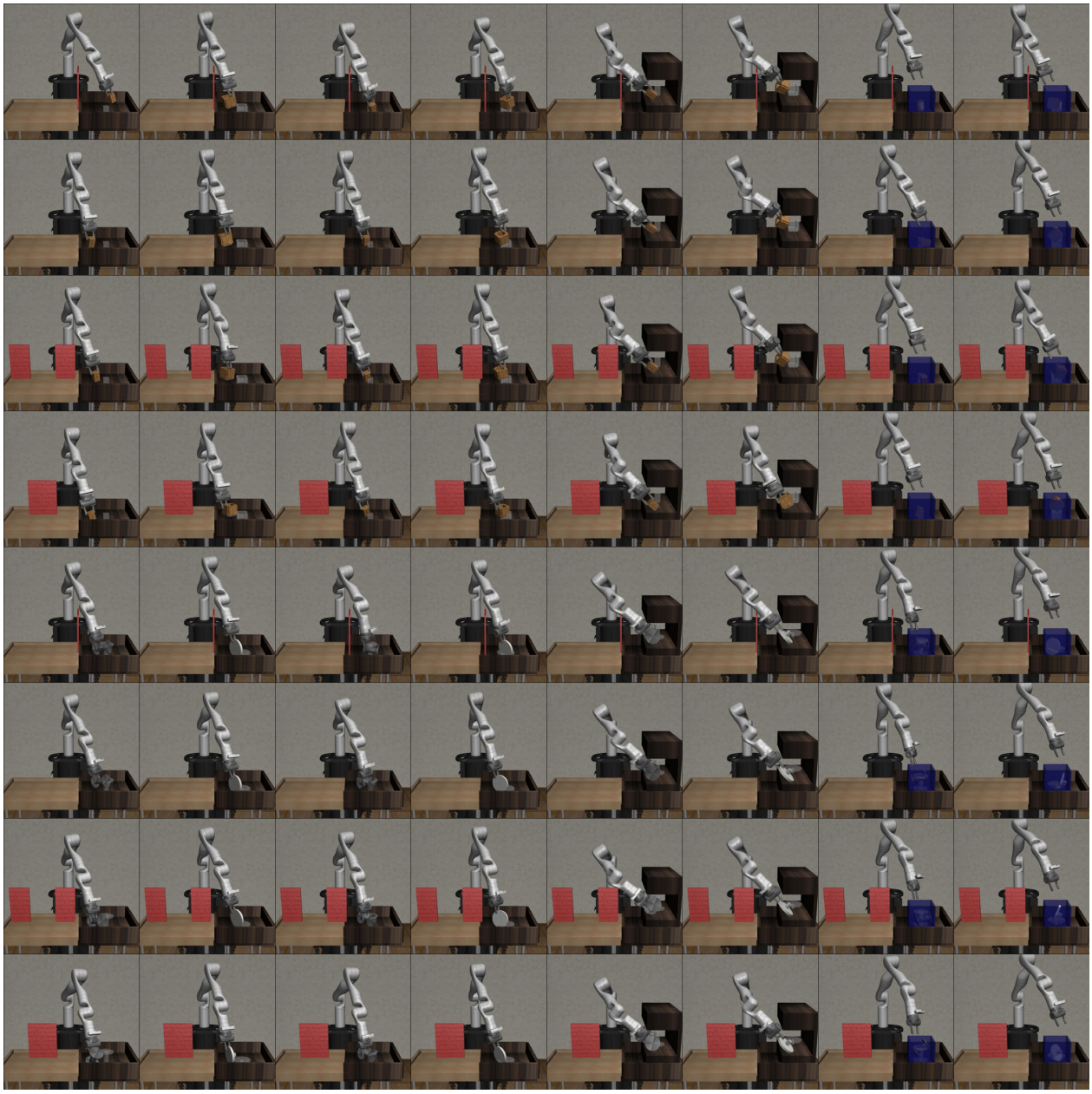


Figure 7: Visualization of the 64 `Gen3` tasks.

# B REWARD FUNCTIONS

Section 4.1.3 in the main paper describes at a high level the structure of the reward function used for each task. Here, we include the precise mathematical formulas used to calculate them. In particular, the reward function computation depended only on the task objective and not any of the other axes.

## B.1 PICK-AND-PLACE TASKS

$$
\begin{aligned}
R_{\text{reach}} &= 0.2\,(1 - \tanh(10 \cdot \text{target\_dist})) \\
R_{\text{grasp}} &= \begin{cases} 0.3 & \text{if grasping} \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{lift}} &= \begin{cases} 0.3 + 0.2\,(1 - \tanh(5 \cdot \text{z\_dist\_target\_height})) & \text{if } R_{\text{grasp}} > 0 \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{approach}} &= \begin{cases} R_{\text{lift}} + 0.2\,(1 - \tanh(2 \cdot \text{goal\_xy\_dist})) & \text{if } R_{\text{lift}} > 0.45 \text{ and object is not above bin} \\ 0.5 + 0.2\,(1 - \tanh(2 \cdot \text{goal\_xy\_dist})) & \text{if } R_{\text{lift}} > 0.45 \text{ and object is above bin} \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{lower}} &= \begin{cases} 0.7 + 0.2\,(1 - \tanh(5 \cdot \text{z\_dist\_bin})) & \text{if object is above bin and } R_{\text{grasp}} > 0 \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{success}} &= \begin{cases} 1 & \text{if object is in bin and } R_{\text{reach}} > 0.07 \\ 0 & \text{otherwise} \end{cases} \\
R &= \max_{\text{stage}} R_{\text{stage}}
\end{aligned}
$$

## B.2 PUSH TASKS

$$
\begin{aligned}
R_{\text{reach}} &= 0.2\,(1 - \tanh(10 \cdot \text{target\_dist})) \\
R_{\text{grasp}} &= \begin{cases} 0.3 & \text{if grasping} \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{approach}} &= \begin{cases} 0.3 + 0.4\,(1 - \tanh(5 \cdot \text{goal\_xy\_dist})) & \text{if } R_{\text{grasp}} > 0 \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{success}} &= \begin{cases} 1 & \text{if goal\_xy\_dist} \leq 0.03 \\ 0 & \text{otherwise} \end{cases} \\
R &= \max_{\text{stage}} R_{\text{stage}}
\end{aligned}
$$

## B.3 TRASH_CAN TASKS

$$
\begin{aligned}
R_{\text{reach}} &= 0.2\,(1 - \tanh(10 \cdot \text{target\_dist})) \\
R_{\text{grasp}} &= \begin{cases} 0.3 & \text{if grasping and object is not in trash can} \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{lift}} &= \begin{cases} 0.3 + 0.2\,(1 - \tanh(5 \cdot \text{z\_dist\_target\_height})) & \text{if } R_{\text{grasp}} > 0 \text{ and object is not in trashcan} \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{approach}} &= \begin{cases} R_{\text{lift}} + 0.2\,(1 - \tanh(2 \cdot \text{goal\_xy\_dist})) & \text{if } R_{\text{lift}} > 0.45 \text{ and object is not in or above trash can} \\ 0.5 + 0.2\,(1 - \tanh(2 \cdot \text{goal\_xy\_dist})) & \text{if } R_{\text{lift}} > 0.45 \text{ and object is above trash can} \\ 0 & \text{otherwise} \end{cases} \\
R_{\text{drop}} &= \begin{cases} 0.95 & \text{if object is above trashcan and } R_{\text{grasp}} = 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

$$R_{\text{success}} = \begin{cases} 1 & \text{if object is in trash can and gripper is not in trash can} \\ 0 & \text{otherwise} \end{cases}$$
$$R = \max_{\text{stage}} R_{\text{stage}}$$

### B.4 SHELF TASKS

$$R_{\text{reach}} = 0.2\,(1 - \tanh(10 \cdot \text{target\_dist}))$$
$$R_{\text{grasp}} = \begin{cases} 0.3 & \text{if grasping} \\ 0 & \text{otherwise} \end{cases}$$
$$R_{\text{lift}} = \begin{cases} 0.3 + 0.2\,(1 - \tanh(5 \cdot \text{z\_dist\_target\_height})) & \text{if } R_{\text{grasp}} > 0 \\ 0 & \text{otherwise} \end{cases}$$
$$R_{\text{align}} = \begin{cases} 0.5 + 0.3\,(1 - \tanh(\text{y\_axis\_orientation})) & \text{if object is in front of shelf} \\ 0 & \text{otherwise} \end{cases}$$
$$R_{\text{approach}} = \begin{cases} 0.8 + 0.1\,(1 - \tanh(5 \cdot \text{y\_dist\_shelf})) & \text{if object is in front of shelf and } R_{\text{align}} > 0.6 \\ 0 & \text{otherwise} \end{cases}$$
$$R_{\text{success}} = \begin{cases} 1 & \text{if object is in shelf} \\ 0 & \text{otherwise} \end{cases}$$
$$R = \max_{\text{stage}} R_{\text{stage}}$$

## C EXPERIMENTAL DETAILS

This section provides details of the experimental setting used to obtain all results in Section 5 in the main paper.

### C.1 PPO DETAILS AND HYPER-PARAMETERS

We used a modified version of `Spinning Up`'s PPO implementation for all experiments. These changes were made to encourage improved exploration because, despite the introduction of highly crafted rewards, initial experiments with the original implementation were suffering from premature convergence.

In particular, we used a multi-layer perceptron (MLP) to represent the mean of a Gaussian policy. Against popular wisdom, which encourages using linear activations in the final layer, we found that adding a $\tanh$ activation led to substantially improved exploration. The rationale is that robot actions are typically capped (artificially in simulators, and by physical limits in real robots). Therefore, if the MLP outputs high-magnitude means for the Gaussian distribution, the sampled actions are all likely to reach the range limits, regardless of the variance of the Gaussian. In consequence, the agent could "cheat" existing techniques to avoid premature convergence (e.g., entropy regularization) by learning a high variance but being deterministic in practice by saturating the actions. The $\tanh$ activation ensures that the actions are never too large in magnitude, which permits the sampling to induce stochasticity (and, consequently, exploration).

The second modification was to use a constant variance for the Gaussian policy, instead of propagating gradients through it. The reason this was necessary is that, with a learnable variance, the agent was finding pathological regions of the optimization landscape that (once more) cheated existing entropy regularization approaches. Concretely, the agent was inflating the variance along dimensions where actions were inconsequential (e.g., joints that rotate in directions orthogonal to the motion of the gripper), and reducing the variance to a minimum along critical dimensions. The resulting policy was therefore deterministic along all interesting dimensions, and so the exploration the agent was engaging in was ineffective. Setting a fixed variance of $\sigma^2 = 1$ ($\log(\sigma) = 0$) for the seven joint actions and $\sigma^2 = 1/e$ ($\log(\sigma) = -0.5$) for the gripper action ensured that the robot consistently explored throughout the learning process and was critical toward enabling the agent to learn the CompoSuite tasks.

The hyper-parameters used for training, reported in Table 4, were obtained via grid search on a set of tasks using single-task training and maintained for the multi-task and compositional settings.

Table 4: PPO hyper-parameters used to train all agents, obtained via grid search with the single-task agent.

| | Single-Task Learner | Multi-Task Learner | Compositional Learner |
|---|---|---|---|
| $\gamma$ | 0.99 | 0.99 | 0.99 |
| # layers | 2 | 2 | — |
| # hidden units | 64 | 256 | — |
| # steps per task per update | 16,000 | 16,000 | 16,000 |
| # total step per task | 10,000,000 | 10,000,000 | 10,000,000 |
| PPO clip value | 0.2 | 0.2 | 0.2 |
| $\pi$ learning rate | $1 \cdot 10^{-4}$ | $1 \cdot 10^{-4}$ | $1 \cdot 10^{-4}$ |
| $V$ learning rate | $1 \cdot 10^{-4}$ | $1 \cdot 10^{-4}$ | $1 \cdot 10^{-4}$ |
| # $\pi$ update iterations | 128 | 128 | 128 |
| # $V$ update iterations | 128 | 128 | 128 |
| Target KL | 0.02 | 0.02 | 0.02 |

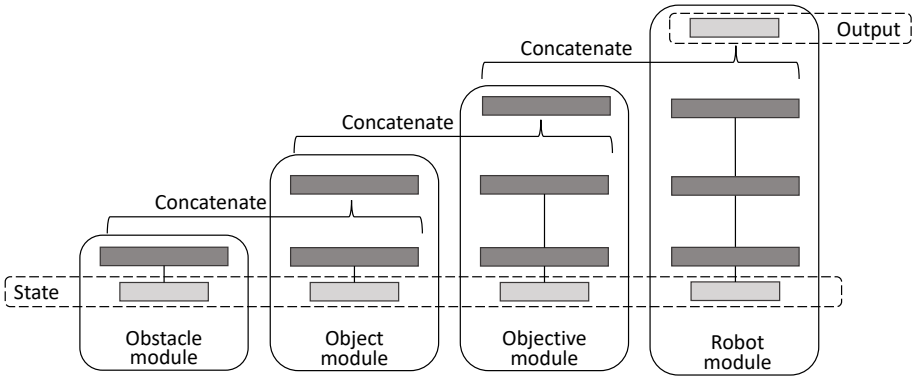


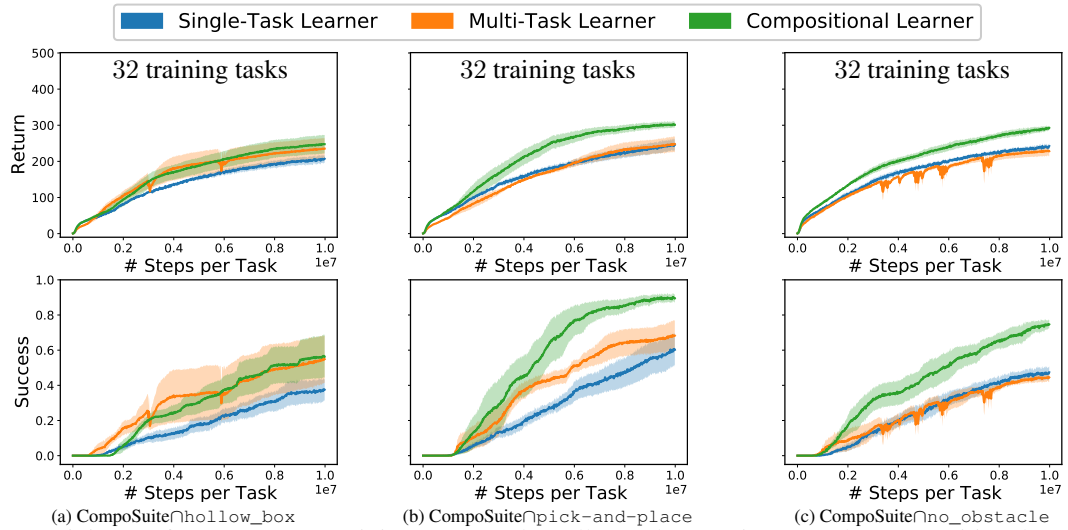

Figure 8: Modular architecture used for learning compositional policies.

### C.2 Compositional network architecture

The network architecture for the compositional agent (Figure 8) follows a graph structure similar to that proposed by Mendez et al. (2022). The network consists of a total of 16 modules, each of which is represented by an MLP and corresponds to one of the compositional elements in CompoSuite. More specifically, there are four obstacle, object, objective, and robot modules, respectively. The modules are assigned to levels in a graph hierarchy, such that the previous level's MLP output is concatenated with the output of the second-to-last layer of the current level's MLP. The concatenated tensor is then used as input to the final layer of the MLP of the current level. Concretely, the obstacle observation is processed first. Every obstacle module consists of a single-hidden-layer MLP with 32 hidden units—because this is the first level, there is no additional input other than the obstacle observation. The second input that is processed is the object input. Object modules consist of two hidden layers, each of size 32. The output of the obstacle network is concatenated with the output of the first layer of the object module and used as input to the second layer of the object module. The object module feeds into the second layer of the objective module, which consists of three layers of size 64. Finally, the objective module's output is routed into the third layer of the robot module, which has a total of four layers: the first three layers are of size 64, and the final layer is the output layer. The same network architecture was used to model both the value function and the policy. This architecture was selected to approximately match the total number of parameters in the multi-task network architecture.

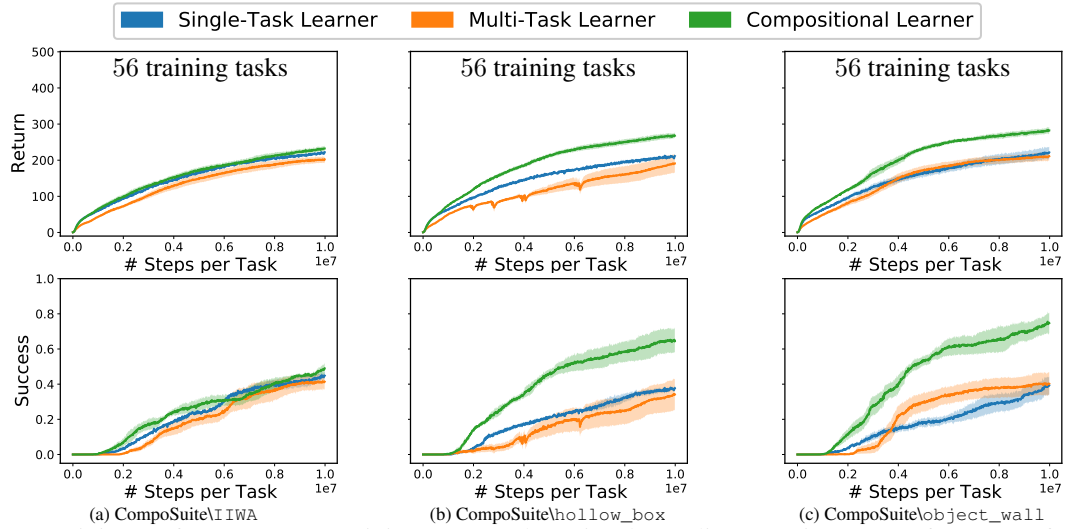
### C.3 Computational resources

We ran experiments using multiple AMD EPYC™ central processing units (CPUs) with 128-thread support. To parallelize data collection, each experiment was run across several processes using MPI with one CPU core per two MPI processes. Every process corresponded to an additional environment collecting samples. The single-task experiments used 16 parallel processes running environments of the same task. Each single-task experiment had an approximate wall-clock training time of 12 hours. For multi-task and compositional training, a single process per task was used. For an experiment of 56 tasks, the agent was trained via 56 MPI processes for approximately 4 days. For an experiment of 224 tasks, the training on 224 processes took approximately 12 days. There was no significant run-time difference between multi-task and compositional training. In general, we allocated roughly 1 GB of RAM per MPI process.




(a) CompoSuite∩hollow_box (b) CompoSuite∩pick-and-place (c) CompoSuite∩no_obstacle

Figure 9: Training performance on remaining suggested smaller-scale experiments. The compositional agent outperformed the competing baselines when all tasks involved pick-and-place or no_obstacle, and matched the multi-task agent when they included hollow_box. Shaded regions represent standard errors across three seeds.




(a) CompoSuite\IIWA (b) CompoSuite\hollow_box (c) CompoSuite\object_wall

Figure 10: Training performance on remaining suggested restricted sampling experiments. Performance of single-task and multi-task agents resembled that on the full benchmark, while the compositional agent performed better when trained on a single hollow_box or object_wall task. Shaded regions represent standard errors across three seeds.

Table 5: CompoSuite zero-shot generalization on remaining suggested smaller-scale settings. Only the compositional agent on CompoSuite∩pick-and-place generalized. Standard errors across three seeds reported after the ±.

| | CompoSuite ∩hollow_box | | CompoSuite ∩pick-and-place | | CompoSuite ∩no_obstacle | |
|---|---|---|---|---|---|---|
| | Multi-task | Comp. | Multi-task | Comp. | Multi-task | Comp. |
| Return | $99.71 \pm 31.32$ | $77.64 \pm 9.55$ | $96.05 \pm 15.55$ | $133.27 \pm 30.20$ | $129.70 \pm 22.43$ | $75.19 \pm 12.33$ |
| Success | $0.13 \pm 0.06$ | $0.12 \pm 0.05$ | $0.22 \pm 0.08$ | $0.40 \pm 0.09$ | $0.18 \pm 0.06$ | $0.09 \pm 0.04$ |

Table 6: CompoSuite zero-shot generalization on the additionally suggested restricted settings. The agents achieved nearly zero generalization. Standard errors across three seeds reported after the ±.

| | CompoSuite \IIWA | | CompoSuite \hollow_box | | CompoSuite \object_wall | |
|---|---|---|---|---|---|---|
| | Multi-task | Comp. | Multi-task | Comp. | Multi-task | Comp. |
| Return | $30.62 \pm 7.52$ | $24.32 \pm 1.22$ | $74.53 \pm 42.78$ | $14.78 \pm 3.72$ | $30.33 \pm 8.07$ | $9.88 \pm 2.73$ |
| Success | $0.00 \pm 0.00$ | $0.00 \pm 0.01$ | $0.09 \pm 0.12$ | $0.01 \pm 0.01$ | $0.03 \pm 0.03$ | $0.01 \pm 0.01$ |

## D ADDITIONAL RESULTS ON SMALLER-SCALE AND RESTRICTED SETTINGS

This section provides results on the remaining settings suggested in Section 4.2.5 in the main paper. Results on the training tasks in the smaller-scale setting, summarized in Figure 9, demonstrate that multi-task agents can more easily transfer knowledge across tasks that use a common robot arm or that share a common objective. In contrast, the compositional agent outperformed the single-task agent in all settings, demonstrating its ability to solve even the most highly varied sets of tasks. Training performance on the restricted setting, shown in Figure 10, exhibits similar trends to the full CompoSuite, as expected. As exceptions, when restricting the compositional agent to a single `hollow_box` or `object_wall` task it outperformed the other methods.

Table 5 shows the zero-shot performance in smaller-scale settings, where both agents achieved little generalization, except the compositional agent on the `pick-and-place` setting. In particular, the multi-task agent failed to generalize in all settings with varied robot arms, supporting the claim of difficulty of generalization across robotic manipulators. Table 6 reports zero-shot results in the restricted sampling setting, where both agents completely failed to generalize.

## E MAXIMUM SUCCESS RATE PER TASK

The combination of elements into the combinatorially many tasks in CompoSuite raises the question of whether some configurations might lead to tasks that are unsolvable for current RL algorithms, for example by restricting the physical space such that the robot arm cannot fulfill a task objective. In order to validate that the vast majority of tasks in CompoSuite are solvable, we compute the maximum success over all trained models for every task and visualize it in Figure 11. For all but one task, at least one agent was able to achieve non-zero success. This corroborates that CompoSuite provides a feasible set of tasks to study compositionality of current RL methods.

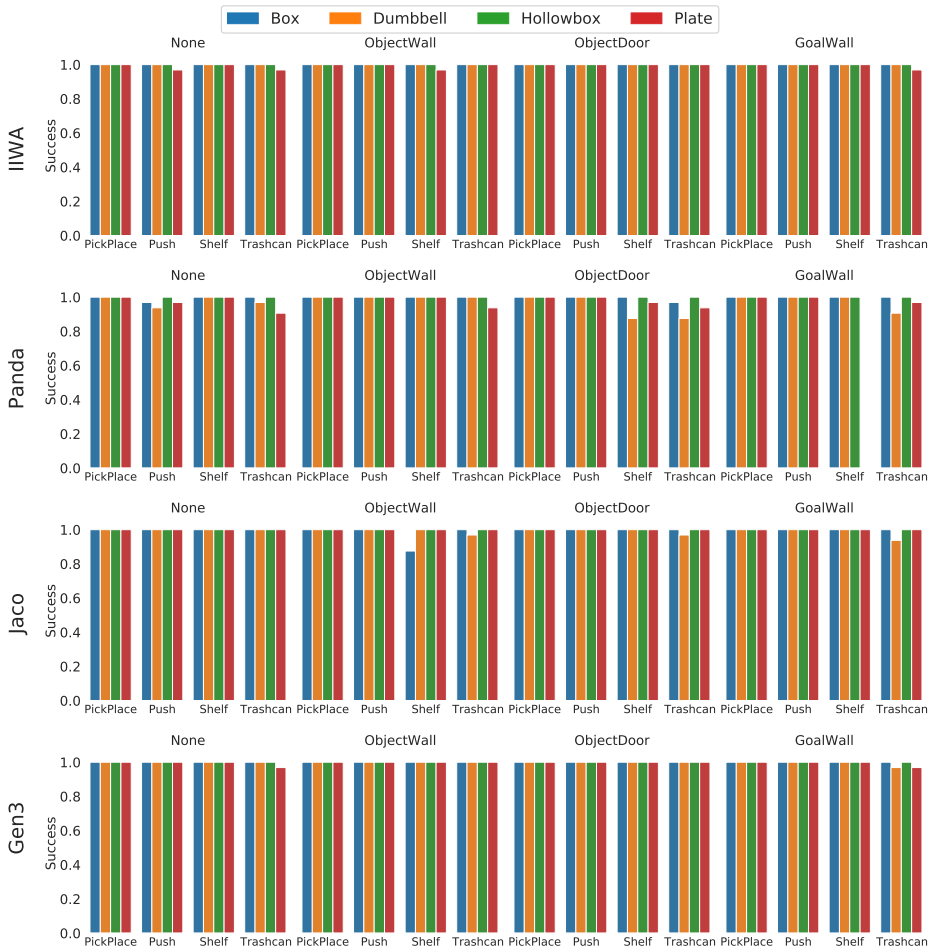



Figure 11: Maximum success rate for each task across all trained agents. All but one tasks are solved at least once.