Augmenting Reinforcement Learning with Behavior Primitives for Diverse Manipulation Tasks

Soroush Nasiriany, Huihan Liu, Yuke Zhu
The University of Austin at Texas
{soroush,huihanl,yukez}@cs.utexas.edu

Abstract

Realistic manipulation tasks require a robot to interact with an environment with a prolonged sequence of motor actions. While deep reinforcement learning methods have recently emerged as a promising paradigm for automating manipulation behaviors, they usually fall short in long-horizon tasks due to the exploration burden. This work introduces **Ma**nipulation **P**rimitive-augmented Reinforcement **Le**arning (MAPLE), a learning framework that augments standard reinforcement learning algorithms with a pre-defined library of behavior primitives. These behavior primitives are robust functional modules specialized in achieving manipulation goals, such as grasping and pushing. To use these heterogeneous primitives, we develop a hierarchical policy that involves the primitives and instantiates their executions with input parameters. We demonstrate that MAPLE outperforms baseline approaches by a significant margin on a suite of simulated manipulation tasks. We also quantify the compositional structure of the learned behaviors and highlight our method's ability to transfer policies to new task variants and to physical hardware. Videos and code are available at `https://ut-austin-rpl.github.io/maple`

1 Introduction

Enabling autonomous robots to solve diverse and complex manipulation tasks has been a grand challenge for decades. In recent years, deep reinforcement learning (DRL) approaches have made great strides towards designing robot manipulation behaviors that are difficult to engineer manually [28, 52, 51, 29]. Nonetheless, state-of-the-art DRL models fall short in long-horizon tasks due to the exploration challenge — the robot has to explore a prohibitively large space of possible behaviors for accomplishing a task. To remedy the exploration burden, prior DRL work has developed various temporal abstraction frameworks to exploit the hierarchical structure of manipulation tasks [9, 44, 4, 13]. These methods learn low-level controllers, often modeled as *skills* or *options*, together with high-level controllers from trial-and-error. While they have demonstrated greater scalability than vanilla DRL methods, they often suffer from high sample complexity, lack of interpretability, and brittle generalization.

In the meantime, decades-long research in robotics has developed a rich repertoire of functional modules specialized at particular robot behaviors, such as grasping [7] and motion planning [30, 24]. These pre-built functional modules, which we refer to as *behavior primitives*, exhibit a high degree of robustness and reusability for achieving certain manipulation goals, such as picking up objects with the end-effector and moving the robot to a target configuration in a collision-free path. In spite of their specialties, it remains a challenge for DRL algorithms to use them as the building blocks to scaffold complex tasks. The challenge is primarily due to the fact that these behavior primitives are *heterogeneous* by design. They take non-uniform parameters as input, operate at varying temporal resolutions, and exhibit distinct behaviors. This thus requires an algorithm to

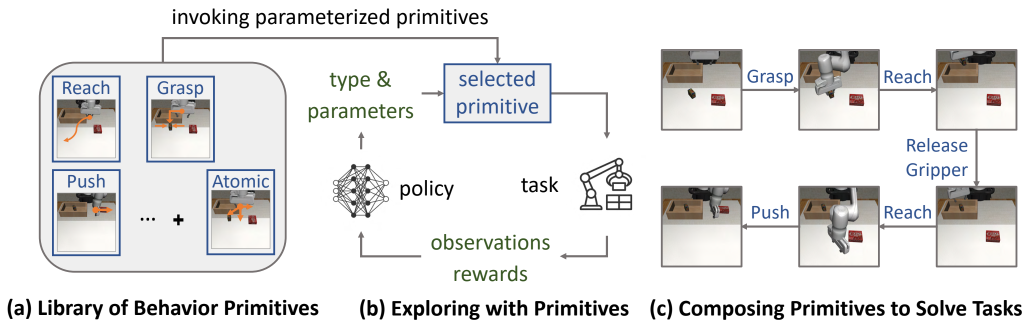

invoking parameterized primitives
Reach
Grasp
Push
Atomic
... +
type &
parameters
selected
primitive
policy
task
observations
rewards
Grasp
Reach
Release
Gripper
Push
Reach
(a) Library of Behavior Primitives
(b) Exploring with Primitives
(c) Composing Primitives to Solve Tasks

Figure 1: **Overview of MAPLE.** (a) We present a learning framework that augments the robot's atomic motor actions with a library of versatile behavior primitives. (b) Our method learns to compose these primitives via reinforcement learning. (c) This enables the agent to solve complex long-horizon manipulation tasks.

reason about the temporal decomposition of a complex task and adaptively compose these behavior primitives accordingly.

A variety of hierarchical modeling approaches in robotics have used behavior modules as low-level building blocks. Notably, task-and-motion planning [27, 18, 66] and neural programming [71, 23] methods have used primitives such as motion planners and pick-and-place controllers to model manipulation tasks in a compositional fashion. They require well-specified domain knowledge to perform task planning or strong human supervision to train a high-level controller with ground-truth task decomposition. These assumptions limit the scalability of these methods in realistic tasks.

In this work, we introduce MAPLE (**Ma**nipulation **P**rimitive-augmented Reinforcement **Le**arning), a general DRL algorithm that harnesses a set of pre-built behavior primitives for solving long-horizon manipulation tasks. To address the exploration challenge of DRL algorithms, our method uses a library of high-level behavior primitives (such as grasping or pushing objects) in conjunction with low-level motor actions to autonomously learn a hierarchical policy (see Fig. 1). Our algorithm models each behavior primitive as an implementation-agnostic controller that produces a temporally extended behavior. At a given state, our DRL policy invokes a behavior primitive (or an atomic motor action) and instantiates it with input parameters. For example, the input parameters to a 6-DoF grasping module can be the pre-grasp end-effector pose. The selected primitive interprets the input parameters and executes one or a sequence of motor actions to realize its specialized behavior. By integrating behavior primitives into DRL algorithms, MAPLE shields away a substantial portion of complexity in manipulation planning, while leaving the flexibility to a generic reinforcement learning algorithm to discover the compositional structure of tasks without strong domain knowledge. Furthermore, by retaining low-level motor actions MAPLE can rely on these actions for the stages of tasks where the finite library of behavior primitives is insufficient to express a desired behavior.

We conduct an extensive set of experiments on a suite of eight manipulation tasks of varying complexities in the robosuite simulation framework [74]. We compare our method to standard DRL approaches [20] that only use low-level motor actions, hierarchical DRL methods that learn options [73, 44, 8] or open-loop task schemas [8]. MAPLE achieves a 70% increase in task success rate compared to using only atomic actions, becoming the only method that consistently solved all single-arm tasks in the standard robosuite benchmark. We also devise a data-driven metric to quantitatively examine the compositionality of manipulation tasks contingent on the available primitives, offering new insight on the challenges and opportunities of compositional modeling for realistic manipulation tasks.

We highlight three contributions of this work: 1) We develop a novel method that augments standard DRL algorithms with pre-defined behavior primitives to reduce the exploration burden; 2) We validate the effectiveness of our method in solving diverse manipulation tasks and quantitatively analyze the compositional structure of these tasks; and 3) We show that the modularity and abstraction offered by the behavior primitives facilitate knowledge transfer of the learned policies to new task variants and physical hardware.

2 Related Work

Deep Reinforcement Learning. Prior work on DRL has investigated a number of approaches to solve long-horizon tasks, through improved exploration strategies [6, 54, 22, 55], learning options [44, 4, 63, 73, 5], unsupervised skill discovery [12, 59], and integrating planning [13, 48]. Despite these efforts, today's DRL methods still struggle in long-horizon robotic tasks due to the exploration burden of learning from scratch. A growing amount of work has examined the use of offline data to alleviate the exploration burden in DRL, namely through demonstration-guided RL [57, 47, 19], learned behavioral priors [56, 62] and action spaces [1, 2] from demonstrations, and offline RL [17, 41, 34, 16]. While promising, these methods can be difficult to scale up due to the costs of acquiring offline data.

Hierarchical Modeling in Robotics. Outside of DRL, there has been a plethora of work in robotics dedicated to building customized functional modules that emit specific robot behaviors, such as grasping [7, 39] and motion planning [30, 3]. Prior works on task-and-motion planning [27, 18, 66] and neural programming [71, 23] have developed hierarchical models that leverage these modules as building blocks to scaffold manipulation tasks. While these methods have demonstrated impressive capabilities in restrictive domains, their applicability has been limited by their reliance on domain knowledge or human supervision.

To bridge the gap between hierarchical models and DRL algorithms that learn from scratch, recent work has harnessed pre-built primitives, such as model-based planners [35], motion planners [72, 68], movement primitives [24, 49], and pre-built skills [8, 36, 64, 60, 61], to expedite DRL algorithms. These approaches aim at retaining the flexibilities of RL algorithms to learn general-purpose behaviors while benefiting from the temporal abstraction provided by the primitives. However, these works are limited as they are confined to using only one or two specific primitives [35, 72, 68], employ rigid primitives that are not reconfigurable [64, 60], or hard-code how the primitives are composed [61]. In contrast, our method adopts a set of versatile primitives and composes them in conjunction with low-level motor actions to solve diverse manipulation tasks.

Reinforcement Learning with PAMDPs. Our formalism specifically falls under the established reinforcement learning framework of Parameterized Action MDPs (PAMDPs) [42], in which the agent executes a parameterized primitive at each decision-making step. We note that several prior works [21, 67, 69, 14, 25] have adapted off-the-shelf deep RL algorithms to the PAMDP setting. Nonetheless, they have focused on relatively simple game domains, shielding away practical challenges in robot manipulation, such as high-dimensional continuous state/action spaces and heterogeneous primitives. Our work is closest to Chitnis et al. [8] and Lee et al. [36], which have modeled robot manipulation with PAMDPs. We provide empirical comparisons to demonstrate the limitations of their modeling choices, yielding less competitive performance in challenging manipulation tasks than ours. We note that concurrent work by Dalal et al. [10] also studies the application of robotic primitives for manipulation tasks, further validating the ability of robotic primitives to accelerate exploration in RL. Our work complements theirs with additional analysis on the compositional structure of the learned behavior and experiments demonstrating the ability to transfer learned policies to novel task variants and to physical hardware.

3 Method

Our goal is to enable robots to leverage behavior primitives to solve manipulation tasks effectively and efficiently. To that end, we seek a library of behavior primitives that serve as the building blocks to scaffold manipulation tasks and a reinforcement learning algorithm that composes these primitives to solve tasks. To evaluate whether our algorithm facilitates compositional behaviors, we also propose a metric to quantify the degree to which the resulting learned behavior is compositional. See Fig. 1 for an overview of our method.

3.1 Decision-Making with Parameterized Behavior Primitives

We adopt reinforcement learning (RL) as the underlying decision-making framework. The objective of RL is to maximize the expected infinite sum of discounted rewards in a Markov Decision Process (MDP), defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p, p_0, \gamma)$. The entities in the tuple represent the state space, the action space, the reward function, the transition function, the initial state distribution, and the discount factor. In most robotic RL problems, the action space $\mathcal{A}$ consists of all atomic actions $u \in \mathbb{R}^{d_{\text{control}}}$ provided by the robot, such as joint torque commands or end-effector displacements. We would like to augment this action space with a heterogeneous library of behavior primitives $\mathcal{L} = \{a^1, a^2, \cdots, a^k\}$ that perform semantically meaningful behaviors. Formally, each behavior primitive — which we will call *primitive* for brevity — $a \in \mathcal{L}$ is represented by a control module $M_a(x)$ that executes a finite, variable sequence of atomic actions $(u_1, u_2, \cdots, u_t), u_i \in \mathbb{R}^{d_{\text{control}}} \forall i$ to achieve a certain behavior, where the exact action sequences are specified by input parameters $x \in \mathbb{R}^{d_a}$. Here d_a is the dimension of the input parameters to the primitive a that varies across different primitives. To incorporate these behavior primitives into the action space, we recast our decision-making problem as a Parameterized Action MDP (PAMDP) [42]. Under this formulation the agent executes at each decision-making step a parameterized action $(a, x) \in \mathcal{A}$ consisting of the type of primitive a and its parameters x.

3.2 Behavior Primitives: Building Blocks for Manipulation

We are interested in equipping agents with a library of versatile primitives that serve as the core building blocks for diverse manipulation tasks. To devise a general learning framework for composing primitives, our decision-making algorithm assumes no knowledge on the detailed implementations of these primitives. These primitives can come in any generic form, ranging from closed-loop skills learned via reinforcement [20, 58] or imitation learning [53], analytical motion planners [30], to even full-fledged grasping systems [39, 7]. Regardless of their inner workings, we must ensure that our primitives are versatile and adaptive to behavioral variations. In our learning framework, we consider these primitives as *functional APIs* that take input parameters x that instantiate action execution. The input parameters usually have clear semantics, such as the 6-DoF end-effector pose for a grasping primitive or a target robot configuration for a motion planning primitive. These parameters significantly improve the flexibility and utility of our primitives for solving complex tasks. Even so, we recognize that our library of primitives may still not be universally applicable in every setting, and equipping the agent solely with these primitives may limit the set of possible behaviors that the agent can achieve. We address this limitation by introducing an additional *atomic primitive* a^{atom} dedicated to performing atomic robot actions. The addition of this atomic primitive will allow the agent to fill in any missing gaps that cannot be fulfilled by the other primitives.

Here we design a library of five primitives, including prehensile and non-prehensile motions, that forms the basis for many manipulation tasks:

- **Reaching:** The robot moves its end-effector to a target location (x, y, z), specified by the input parameters. Execution takes at most 15 atomic actions.
- **Grasping:** The robot moves its end-effector to a pre-grasp location (x, y, z) at a yaw angle ψ, specified by the input parameters, and closes its gripper. Execution takes at most 20 atomic actions.
- **Pushing:** The robot reaches a starting location (x, y, z) at a yaw angle ψ and then moves its end-effector by a displacement $(\delta_x, \delta_y, \delta_z)$. The input parameters are 7D. Execution takes at most 20 atomic actions.
- **Gripper Release:** The robot repeatedly applies atomic actions to open its gripper. This primitive has no input parameters. Execution takes 4 atomic actions.
- **Atomic:** The robot applies a single atomic action of dimension d_{control}.

We implemented these primitives as hard-coded closed-loop controllers, each requiring only a handful of lines of code. We highlight that these primitives take input parameters of different dimensions, operate at variable temporal lengths, and produce distinct behaviors. These properties make them challenging to utilize in a learning framework. In the following, we will introduce our algorithm for composing these primitives to solve diverse manipulation tasks.

3.3 Composing Primitives via Reinforcement Learning

We follow the PAMDP framework outlined in Section 3.1, where at each decision-making step a policy π must select a discrete behavior primitive type a and its corresponding continuous parameters x. Previous work has explored various policy structures that reason over parameterized primitives.

The simplest approach is a flat policy [68, 21] that outputs a distribution over the primitive type a and all primitive parameters $\{x^1, x^2, \cdots, x^k\}$. A major drawback of this approach is that the total number of policy outputs can quickly become intractable as additional primitives are introduced. We address this limitation with a hierarchical policy where at the high level a *task policy* π_{tsk} determines the primitive type a and at the low level a *parameter policy* π_p determines the corresponding primitive parameters x. See Fig. 2 for an illustration of our policy architecture. In addition to reducing the overall number of output parameters our hierarchical design facilitates modular reasoning, delegating the high-level to focus on *which* primitive to execute and the low-level to focus on *how* to instantiate that primitive. We note that a few prior works have previously explored this hierarchical design [67, 14] but to our knowledge we are the first to demonstrate its utility on complex manipulation domains with a set of heterogeneous primitives.

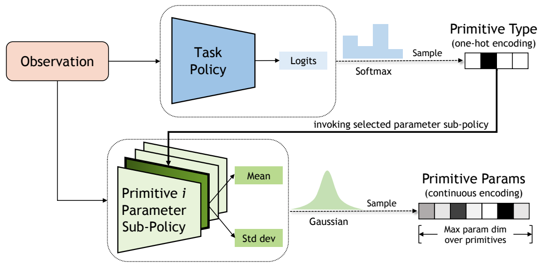


Figure 2: **Policy Architecture.** We adopt a hierarchical policy, with a high-level task policy that determines *which* primitive to apply and a low-level parameter policy that determines *how* to instantiate that primitive.

In principle, we can integrate our policy architecture with any DRL algorithm designed for continuous control. We choose Soft Actor-Critic (SAC) [20], a state-of-the-art DRL algorithm that aims to maximize environment rewards as well as the policy entropy. We modify the standard critic neural network $Q_\theta(s, a)$ and actor neural network $\pi_\phi(a|s)$ with our critic network $Q_\theta(s, a, x)$ and our hierarchical policy networks $\pi_{tsk_\phi}(a|s)$, $\pi_{p_\psi}(x|s, a)$. Under these changes the losses for the critic, task policy, and parameter policy are defined respectively (we highlight components pertaining to the task policy in red and the parameter policy in blue):

$$J_Q(\theta) = \Bigg(Q_\theta(s,a,x) - \Big(r(s,a,x) + \gamma\Big(Q_{\bar\theta}(s',a',x') \quad (1)$$

$$- \alpha_{tsk} \log(\pi_{tsk_\phi}(a'|s')) - \alpha_p \log(\pi_{p_\psi}(x'|s',a'))\Big)\Big)\Bigg)^2$$

$$J_{\pi_{tsk}}(\phi) = \mathop{\mathbb{E}}_{a \sim \pi_{tsk_\phi}} \left[\alpha_{tsk} \log(\pi_{tsk_\phi}(a|s)) - \mathop{\mathbb{E}}_{x \sim \pi_{p_\psi}} Q_\theta(s,a,x)\right] \quad (2)$$

$$J_{\pi_p}(\psi) = \mathop{\mathbb{E}}_{a \sim \pi_{tsk_\phi}} \mathop{\mathbb{E}}_{x \sim \pi_{p_\psi}} \left[\alpha_p \log(\pi_{p_\psi}(x|s,a)) - Q_\theta(s,a,x)\right] \quad (3)$$

Here α_{tsk} and α_p control the maximum entropy objective for the task policy and parameter policy, respectively.

3.4 Facilitating Exploration with Affordances

Compared with existing methods that reason purely over atomic actions, our algorithm benefits from accelerated exploration due to the temporal abstraction provided by our behavior primitives. However, as previous work [56] has noted, even reasoning with temporally extended actions can present an exploration challenge. One way to address this issue is to equip the agent with *affordances* that help to discern the utility of actions in different settings. For example, a grasping skill is only appropriate when applied in the vicinity of graspable objects, and a pushing skill is only appropriate in the vicinity of pushable objects.

In our framework, these affordances can be expressed by adding to the reward function an auxiliary affordance score $s_{\text{aff}}(s, x; a) \in [0, 1]$ that measures the affinity for parameters x at a particular state s for a given primitive a. These affordances scores can in principle come from learned models trained on robot interaction data [61, 46, 40, 43, 70] or human data [11, 15, 45, 33]. Nonetheless, as our primitive parameters carry clear semantic meanings, we can analytically define these affordance scores based on the objects' physical states. Concretely, for the *atomic* and *gripper release* primitives, we always give an affordance score of 1 to enable the universal applicability of these

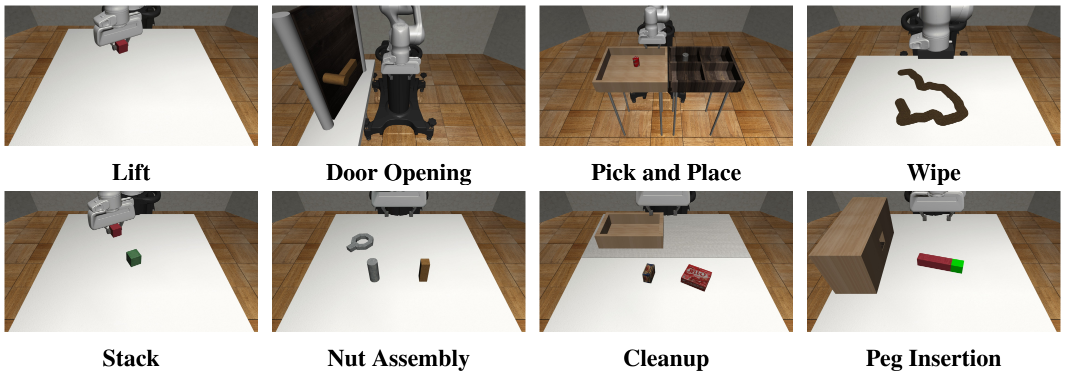


Figure 3: **Simulated Environments.** We perform evaluations on eight manipulation tasks. The first six come from the robosuite benchmark [74]. We designed the last two to test our method in multi-stage, contact-rich tasks: Cleanup requires storing a spam can into a storage bin and a jello box at a corner; Peg Insertion requires inserting a peg into a block.

primitives. For the remaining *reach*, *grasp*, and *push* primitives we implement general, easy-to-define affordances encouraging the agent to reach relevant areas of interest in the workspace. More specifically, these primitives all involve reaching a location x_{reach} = `[x[0], x[1], x[2]]` and we encourage the agent to specify the reaching parameters x_{reach} to be within a threshold of a set of keypoints P:

$$s_{\text{aff}}(s, x; a) = \max_{p \in P} 1 - \tanh\Big(\max(\|x_{reach} - p\| - \tau, 0)\Big) \tag{4}$$

The keypoints P for pushing are the locations objects to push, the locations of objects to grasp for grasping, and the target reaching location for reaching.

3.5 Quantifying Compositionality

Our framework relies on the hypothesis that most manipulation tasks have an intrinsic compositional structure and that our algorithm can discover this structure. To examine this hypothesis we propose to measure the degree to which our learned agent exhibits compositional behaviors with a quantifiable metric. Assume that we are given a set of n trajectories in which the agent solved a task $\mathcal{T}$: $\{\tau^i\}_{i=1}^n = \{(s_1^i, (a_1^i, x_1^i), \cdots, s_{T_i}^i, (a_{T_i}^i, x_{T_i}^i), s_{T_i+1}^i)\}_{i=1}^n$. The corresponding *task sketches* $\{K^i\}_{i=1}^n = \{(a_1^i, a_2^i, \cdots a_{T_i}^i)\}_{i=1}^n$ capture the high-level task semantics and provide useful abstractions through which we can analyze the compositional structure of these trajectories.

Intuitively, agents that demonstrate compositional reasoning will express recurring patterns of behaviors across their task sketches and prefer the use of high-level primitives over low-level ones. We quantify this intuition by computing the *Levenshtein distance* [37] among task sketches, which in our context measures the minimum number of single-token edits (insertions, deletions, or substitutions) needed to transform one task sketch to another. In our task sketches we represent each non-atomic primitive *type* as a unique token, and in order to explicitly discourage the use of low-level atomic actions, we represent each *individual occurrence* of an atomic primitive in our task sketches as a unique token. Given a task $\mathcal{T}$ and available primitives $\mathcal{L}$, we compute the compositionality of the agent's behavior as the average pairwise normalized score between the resulting task sketches:

$$f_{comp}(\mathcal{T}; \mathcal{L}) = \frac{1}{n(n-1)} \sum_{i \neq j} 1 - \frac{d_{\text{Lev}}(K_i, K_j)}{\max(|K_i|, |K_j|)} \tag{5}$$

Note that this measure is contingent on the choice of behavior primitives in the library $\mathcal{L}$, and we can use this measure to compare the effectiveness of different libraries.

One question that arises is whether MAPLE incentivizes the agent to discover compositional task structures in the first place. While there is no explicit mechanism to discover recurring patterns of primitives, our algorithm exhibits compositional reasoning by preferring the use of high-level primitives over low-level ones. Due to the temporal abstraction encapsulated by the high-level primitives, the agent can make far greater progress toward solving the task by using high-level primitives and thus receives higher average reward per timestep. This incentivizes the agent to choose higher-level primitives over lower-level actions whenever appropriate.

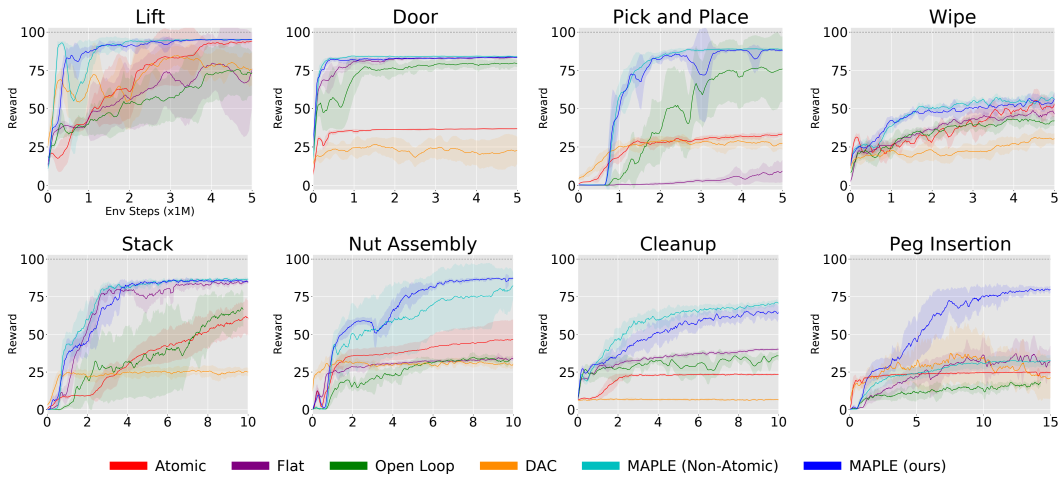


Figure 4: **Main Results.** Learning curves showing average episodic task rewards throughout training, normalized between 0 and 100. All experiments are averaged over 5 seeds, with shaded regions depicting the standard deviation.

4 Experiments

Our experiments study 1) whether our method can compose pre-built behavior primitives and atomic actions to solve complex tasks, 2) the degree to which the learned behavior is compositional, and 3) whether our approach is amenable to transfer to task variants and to real hardware.

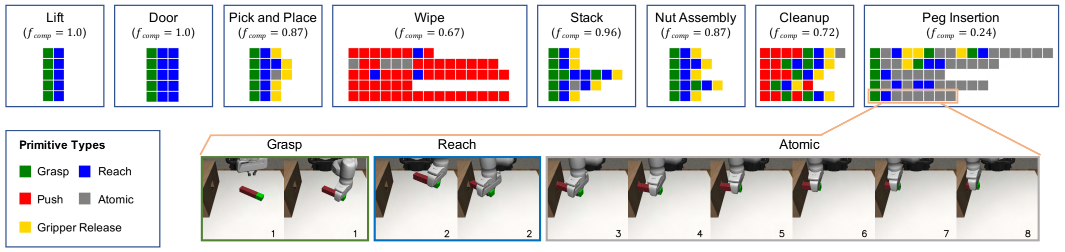
4.1 Experimental Setup

We examine these questions on robosuite [74], a framework for simulated robot manipulation tasks. We consider a comprehensive suite of eight manipulation tasks of varying complexities (see Fig. 3). For all tasks we adopt a Franka Emika Panda robot arm equipped with a parallel jaw gripper (with the exception of the wiping task). The robot is controlled through end-effector displacements with an operational space controller (OSC) [31]. At each decision-making step our agent can execute either an atomic OSC action or one of the temporally extended non-atomic primitives outlined in Section 3.2. In return the agent receives 1) a dense reward signal indicating task progress and 2) an observation comprising the robot's proprioceptive state and pose information of the objects in the environment.

4.2 Quantitative Evaluations

We compare our method (**MAPLE**) to five baselines. The first baseline uses exclusively atomic actions (**Atomic**), which corresponds to the standard Soft Actor-Critic model [20] trained on end-effector commands. To understand the effect of hierarchy on our policy design, we compare to a flat variant where the policy outputs the primitive type and parameters independently (**Flat**), following the design by Lee et al. [36] and Neunert et al. [50]. We also compare to a variant of our method using an open loop task policy (**Open Loop**), following Chitnis et al. [8] which suggests utilizing an open-loop task schema improves the sample efficiency of the learning algorithm. Next, we compare to HIerarchical Reinforcement learning with Off-policy correction (**HIRO**) [44] and Double Actor-Critic (**DAC**) [73], state-of-the-art hierarchical DRL methods which aim to learn low-level policies (or options) along with high-level controllers. HIRO failed to make progress and we thus omit it from our results. Finally, we compare to a self baseline where we include all primitives *except* the atomic primitive **MAPLE (Non-Atomic)**, to understand whether we need atomic actions to satisfy behaviors that cannot be fulfilled by the non-atomic primitives. All baselines using behavior primitives use the affordance score outlined in Section 3.4.

Fig. 4 outlines environment rewards throughout training. We also evaluated the final task success rates at the end of training: MAPLE achieved the highest average success rate across all baselines (90%), compared to 19% for the Atomic baseline, 36% for Flat, 41% for Open Loop, 11% for DAC,



Lift (f_{comp} = 1.0)
Door (f_{comp} = 1.0)
Pick and Place (f_{comp} = 0.87)
Wipe (f_{comp} = 0.67)
Stack (f_{comp} = 0.96)
Nut Assembly (f_{comp} = 0.87)
Cleanup (f_{comp} = 0.72)
Peg Insertion (f_{comp} = 0.24)
Primitive Types
Grasp
Reach
Push
Atomic
Gripper Release
Grasp
Reach
Atomic

Figure 5: **Analyzing Learned Behavior.** (Top) We visualize the task sketches that our agent has learned. Each row corresponds to a single sketch progressing temporally from left to right. For each task we also report the compositionality score f_{comp}. (Bottom) We visualize the behavior for a peg insertion sketch.

and 79% for MAPLE (Non-Atomic). First, we see that the inclusion of non-atomic primitives allows MAPLE to significantly outperform the Atomic baseline, achieving on average 2-3× higher rewards and 71% higher success rate. Qualitatively we found that the Atomic baseline fails to advance past the first stage in most tasks while our method is able to successfully solve all tasks. Next we find that the Flat baseline is unable to reliably solve all tasks, demonstrating that our hierarchical policy design is key to reasoning over a heterogeneous set of primitives. While the Open Loop baseline is able to solve basic tasks such as Door Opening and Pick and Place, it struggles with tasks that require the agent to adaptively reason about the current state of the task. DAC is only able to solve the Lift task, highlighting the difficulty of learning complex tasks *from scratch* even when employing temporal abstraction. Finally we find that the Non-Atomic self baseline is on par with our method in most tasks, yet it notably fails for Peg Insertion as the non-atomic primitives are not expressive enough to perform the contact-rich insertion phase. Together, these results highlight that given an appropriate library of primitives and an appropriate policy structure we can solve a wide range of manipulation tasks.

4.3 Model Analysis

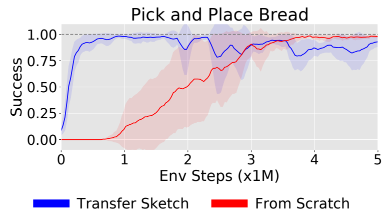
4.3.1 Emergence of Compositional Structures

We present an analysis of the task sketches that our method learned for each task in Fig. 5. We see evidence that the agent unveils compositional task structures by applying temporally extended primitives whenever appropriate and relying on atomic actions otherwise.

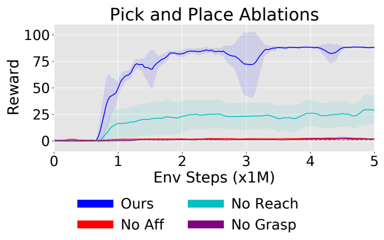
For example, for the peg insertion task the agent leverages the grasping primitive to pick up the peg and the reaching primitive to align the peg with the hole in the block, but then it uses atomic actions for the contact-rich insertion phase. In Fig. 5 we also quantify the degree to which these task sketches are compositional via the compositionality score f_{comp} that we defined in Eq. (5). As we can see, tasks involving contact interactions such as Peg Insertion and Wiping have lower scores than prehensile tasks such as Pick and Place and Stacking.

4.3.2 Transfer to Semantically Similar Task Variants

We have seen how task sketches enable interpretability by serving as blueprints of high-level semantic task structure. We can leverage these task sketches to accelerate learning on similar task instances. We propose to re-use the task sketch from a semantically similar task, and only learn the corresponding primitive parameters. We vali-



(a) Task Transfer



(b) Ablations

Figure 6: **(a) Task Transfer.** We transfer the learned task sketch from a source task (pick and place can) to a semantically similar task variant (pick and place bread), enabling us to learn the target task over 5× faster. **(b) Ablations**. Without affordances, reaching, or grasping, the agent is unable to solve tasks due to the exploration burden.

date this idea with a preliminary experiment on the Pick and Place domain, where we transfer the task sketch from a source task of placing a soda can into one bin, to a target task of placing a loaf of bread into a different bin.

As shown in Fig. 6a, we are able to solve the bread task significantly faster than learning the task from scratch with a sample efficiency of over $5\times$. This result implies that our task sketch serves as a high-level scaffold of a manipulation task, which can be re-used by learning algorithms for faster adaptation to related task variants.

Ablation Study. We perform an ablation study examining the role of affordances and individual manipulation primitives in facilitating exploration. We specifically perform experiments on the Pick and Place task, comparing our method (**Ours**) to ablations 1) without affordances in the reward function (**No Aff**), 2) without the reaching skill (**No Reach**), and 3) without the grasping skill (**No Grasp**). We see in Fig. 6b that without these components the agent fails to solve the task, underscoring that our method is reliant on the appropriate primitive skills and affordances to effectively overcome the exploration burden.

4.4 Real-World Evaluation

We conclude with an evaluation on real-world copies of the Stack and Cleanup tasks (see Fig. 7). As our behavior primitives offer high-level action abstractions and encapsulate low-level complexities of motor actuation, our policies can directly transfer to the real world. We trained MAPLE on simulated versions of these tasks and executed the resulting policies to the real world. We re-implemented our behavior primitives on the real robot and used an off-the-shelf pose estimation model [65] to estimate the environment states as the model input. We achieved an average success rate of 93% on Stack and 83% on Cleanup. Videos of the experiments can be found on the project webpage[1].

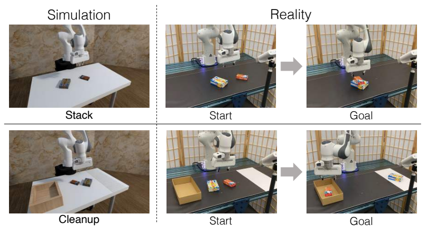


Figure 7: **Transfer to Real-World Tasks.** We transfer our policy trained on simulated environments to the real-world Stack and Cleanup tasks.

5 Conclusion

We presented MAPLE, a reinforcement learning framework that incorporates behavior primitives in conjunction with low-level motor actions to solve complex manipulation tasks. Our experiments demonstrate that behavior primitives can significantly improve exploration while low-level motor actions allow us to retain flexibility to learn intricate behaviors. Our work opens the possibility for several avenues for future work. First, learning affordances using data-driven methods [61, 33, 40, 43] can expand the scalability of our method. Second, while atomic actions can help fill in gaps where the primitives are insufficient (such as peg insertion), we are unable to fill in large gaps that require a significant number of low-level action executions (as seen in the ablation experiments). Further research on exploration and credit assignment is needed to overcome these challenges. Finally, an exciting avenue for future work is to continually discover recurring compositions of primitives and add them to the library of primitives, which can ultimately enable curriculum learning of progressively more challenging tasks.

Acknowledgements

We would like to thank Yifeng Zhu for assisting with the real-world experiments and Abhishek Joshi for providing simulation rendering content. We would also like to thank Yifeng Zhu, Braham Snyder, and Josiah Wong for providing feedback on this manuscript. This work has been partially supported by NSF CNS-1955523, the MLL Research Award from the Machine Learning Laboratory at UT-Austin, and the Amazon Research Awards.

[1]`https://ut-austin-rpl.github.io/maple`

References

[1] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. In *ICLR*, 2021.

[2] Arthur Allshire, Roberto Martín-Martín, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. Laser: Learning a latent action space for efficient reinforcement learning. In *ICRA*, 2021.

[3] N.M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *ICRA*, 1996.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

[5] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *ICLR*, 2020.

[6] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016.

[7] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 2013.

[8] Rohan Chitnis, Shubham Tulsiani, Saurabh Gupta, and Abhinav Gupta. Efficient bimanual manipulation using learned task schemas. In *ICRA*, 2020.

[9] John Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML*, 2018.

[10] Murtaza Dalal, Deepak Pathak, and Ruslan Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. In *NeurIPS*, 2021.

[11] Thanh-Toan Do, Anh Nguyen, and Ian Reid. Affordancenet: An end-to-end deep learning approach for object affordance detection. In *ICRA*, 2018.

[12] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2018.

[13] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 2019.

[14] Zhou Fan, Rui Su, Weinan Zhang, and Yong Yu. Hybrid actor-critic reinforcement learning in parameterized action space. In *IJCAI*, 2019.

[15] Kuan Fang, Te-Lin Wu, Daniel Yang, Silvio Savarese, and Joseph J. Lim. Demo2vec: Reasoning object affordances from online videos. In *CVPR*, 2018.

[16] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[17] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

[18] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 2021.

[19] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *CoRL*, 2019.

[20] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[21] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *ICLR*, 2016.

[22] Rein Houthooft, Xi Chen, Yan Duan, J. Schulman, F. Turck, and P. Abbeel. Vime: Variational information maximizing exploration. In *NIPS*, 2016.

[23] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *CVPR*, 2019.

[24] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 2013.

[25] Ayush Jain, Andrew Szot, and Joseph J. Lim. Generalization to new actions in reinforcement learning. In *ICML*, 2020.

[26] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[27] Leslie P. Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, 2011.

[28] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

[29] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv*, 2021.

[30] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *IJRR*, 2011.

[31] Oussama Khatib. Inertial properties in robotic manipulation: An object-level framework. *IJRR*, 1995.

[32] Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learning options end-to-end for continuous action tasks, 2017.

[33] Mia Kokic, D. Kragic, and Jeannette Bohg. Learning task-oriented grasping from human activity datasets. *IEEE Robotics and Automation Letters*, 2020.

[34] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

[35] Michelle A. Lee, Carlos Florensa, Jonathan Tremblay, Nathan Ratliff, Animesh Garg, Fabio Ramos, and Dieter Fox. Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning. In *ICRA*, 2020.

[36] Youngwoon Lee, Jingyun Yang, and Joseph J Lim. Learning to coordinate manipulation skills via skill behavior diversification. In *ICLR*, 2020.

[37] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, February 1966.

[38] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

[39] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *RSS*, 2017.

[40] Priyanka Mandikal and Kristen Grauman. Learning dexterous grasping with object-centric visual affordances. In *ICRA*, 2021.

[41] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *ICRA*, 2020.

[42] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *AAAI*, 2016.

[43] Kaichun Mo, Leonidas Guibas, Mustafa Mukadam, Abhinav Gupta, and Shubham Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *ICCV*, 2021.

[44] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

[45] Tushar Nagarajan, Christoph Feichtenhofer, and Kristen Grauman. Grounded human-object interaction hotspots from video. In *ICCV*, 2019.

[46] Tushar Nagarajan and Kristen Grauman. Learning affordance landscapes for interaction exploration in 3d environments. In *NeurIPS*, 2020.

[47] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *ICRA*, 2018.

[48] Soroush Nasiriany, Vitchyr H. Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In *NeurIPS*, 2019.

[49] Gerhard Neumann, Christian Daniel, Alexandros Paraschos, Andras Kupcsik, and Jan Peters. Learning to combine primitive skills: A step towards versatile robotic manipulation. *Frontiers in Computational Neuroscience*, 2014.

[50] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Jost Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. In *CoRL*, 2019.

[51] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019.

[52] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2019.

[53] Takayuki Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 2018.

[54] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[55] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *ICML*, 2019.

[56] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In *CoRL*, 2020.

[57] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *RSS*, 2018.

[58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[59] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *ICLR*, 2020.

[60] Mohit Sharma, Jacky Liang, Jialiang Zhao, Alex LaGrassa, and Oliver Kroemer. Learning to combine primitive skills: A step towards versatile robotic manipulation. In *CoRL*, 2020.

[61] MAnthony Simeonov, Yilun Du, Beomjoon Kim, Francois R. Hogan, Joshua Tenenbaum, Pulkit Agrawal, and Alberto Rodriguez. Learning to combine primitive skills: A step towards versatile robotic manipulation. In *CoRL*, 2020.

[62] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *ICLR*, 2020.

[63] Matthew Smith, Herke Hoof, and Joelle Pineau. An inference-based policy gradient method for learning options. In *ICML*, 2018.

[64] Robin Strudel, Alexander Pashevich, Igor Kalevatykh, Ivan Laptev, Josef Sivic, and Cordelia Schmid. Learning to combine primitive skills: A step towards versatile robotic manipulation. In *ICRA*, 2020.

[65] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *CoRL*, 2018.

[66] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *IJRR*, 2021.

[67] E. Wei, Drew Wicke, and S. Luke. Hierarchical approaches for reinforcement learning in parameterized action space. *arXiv*, abs/1810.09656, 2018.

[68] Fei Xia, Chengshu Li, Roberto Martín-Martín, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation. In *ICRA*, 2021.

[69] J. Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, T. Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv*, abs/1810.06394, 2018.

[70] Danfei Xu, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. In *ICRA*, 2021.

[71] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *ICRA*, 2018.

[72] Jun Yamada, Youngwoon Lee, Gautam Salhotra, Karl Pertsch, Max Pflueger, Gaurav S. Sukhatme, Joseph J. Lim, and Peter Englert. Motion planner augmented reinforcement learning for obstructed environments. In *CoRL*, 2020.

[73] Shangtong Zhang and Shimon Whiteson. Dac: The double actor-critic architecture for learning options. In *NeurIPS*, 2019.

[74] 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.