Hierarchical Primitive Composition: Simultaneous Activation of Skills With Inconsistent Action Dimensions in Multiple Hierarchies

Jeong-Hoon Lee ⓘ and Jongeun Choi ⓘ, *Member, IEEE*

Abstract—**Deep reinforcement learning has shown its effectiveness in various applications, providing a promising direction for solving tasks with high complexity. However, naively applying classical RL for learning a complex long-horizon task with a single control policy is inefficient. Thus, policy modularization tackles this problem by learning a set of modules that are mapped to primitives and properly orchestrating them. In this study, we further expand the discussion by incorporating simultaneous activation of the skills and structuring them into multiple hierarchies in a recursive fashion. Moreover, we sought to devise an algorithm that can properly orchestrate the skills with different action spaces via multiplicative Gaussian distributions, which highly increases the reusability. By exploiting the modularity, interpretability can also be achieved by observing the modules that are used in the new task if each of the skills is known. We demonstrate how the proposed scheme can be employed in practice by solving a pick and place task with a 6 DoF manipulator, and examine the effects of each property from ablation studies.**

Index Terms—**Deep learning in grasping and manipulation, incremental learning, reinforcement learning.**

I. INTRODUCTION

RECENT studies on deep reinforcement learning (RL) have proven their effectiveness when applied to the field of robotics within simulators [1] or in the real world [2]–[6]. However, it is inefficient to naively apply classical RL for learning a complex long-horizon task with a single control policy (flat policy), as it is required for the agent to explore a vast state-action space and collect tens to hundreds of millions of samples till it sufficiently encounters a proper learning signal [7]–[9]. Moreover, it has been reported that such a method is susceptible to overfitting the policy to idiosyncrasies of the environment [10].

A more practical approach is to reduce the scope of the exploration by reusing the prior knowledge obtained from solving

relevant and comparatively easier tasks or *subtasks*. Among various methods of leveraging prior knowledge, the transfer learning scheme proposes to compose policies that are trained from previous tasks and fine-tune on newly appointed tasks [11], [12]. Although this approach had successfully reduced the amount of data required, knowledge transfer via fine-tuning incurs catastrophic forgetting when the gradients of the new task flow through [13].

Policy modularization, on the other hand, suggests learning a set of modules that are mapped to primitive *skills* and properly orchestrating them to tackle such problems [11], [14]–[23]. If the modules are fixed, not only it prevents forgetting, but it also offers interpretability by observing the modules that are used in the new task [24]. A straightforward way of coordinating these skills is to arrange them to be activated one at a particular time. This, however, will restrict the behavior of the agent to the prescribed primitive while it is necessary to perform more than one subtask simultaneously as the complexity of the system grows [19]. Thus, [16], [19] studied concurrent activation of skills to mitigate such a problem, but they required every skill to have identical action dimensions. Lastly, the task should be divided into multiple hierarchies. For instance, even a simple robotic task such as pick-and-place, as shown in Fig. 1, can be seen as a task consisting of multiple hierarchies of subtasks with different difficulty levels.

To this end, we propose a concept of Hierarchical Primitive Composition (HPC), where each of the known predefined/pretrained primitives is assembled in a hierarchy to construct an overall compound policy. The main challenge is to properly coordinate simultaneous activation of skills with inconsistent action spaces. Thus, we compose the primitives via multiplicative Gaussians with a factored sum, as is illustrated in Section III-D. Additionally, we focus on recursively constructing the hierarchies of skills to structuralize our architecture into multiple levels of hierarchy. We test the proposed methodology on the `pick-and-place` task with a 6 DoF manipulator and examine the effects of simultaneous activation, multi-level hierarchy, and our action composition with ablation studies. The contributions of this work are:

- HPC can simultaneously activate the skills with inconsistent action spaces. This will allow any kinds of pretrained
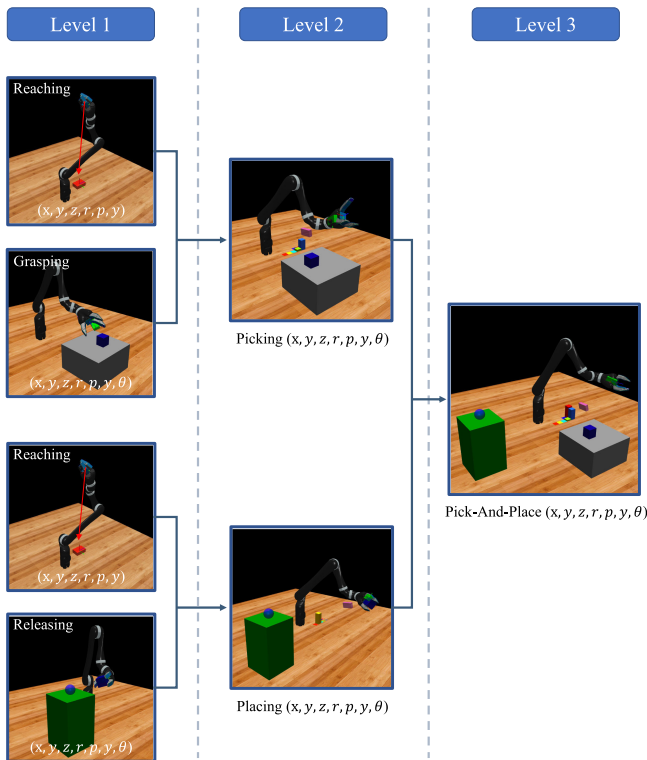
Fig. 1. Schematics of the `pick-and-place` task formulation. The compound `pick-and-place` is assigned to level 3. It can be decomposed into level 2 tasks, `picking` and `placing`. These tasks can again be decomposed into level 1 tasks, {`reaching`, `grasping`} and {`reaching`, `releasing`}, respectively.

policies to be modularized as primitives and thus increase reusability and modularity.

- HPC tackles a complex task through multiple hierarchies in a recursive fashion, which builds on a collection of primitives from other tasks or the HPC module itself.
- HPC greatly reduces the learning time of the complicated tasks by abstracting the scope of the action leveraging on the surrogate Markov decision process (MDP).

II. RELATED WORK

Studies on learning reusable skills have long been of an interest and still remain an active research area [25]. Transfer learning schemes [11], [12] involve priorly training an agent on relevant or comparatively easier tasks and fine-tuning the agent on newly appointed tasks. [20] explores this area further to examine how well a skill can be learned and transferred by disentangling the dimension of the problem. While they have successfully reduced the amount of data required, catastrophic forgetting can occur when the gradients of the new task flow through the existing skill primitives [13].

Policy modularization mitigates such incurrence by regarding each primitive as a module and detaching the gradient flow of the new task. [26] learns modules of skill-specific embedder and aggregates the embeddings of all relevant skills with a compositional layer to then learn a policy that maps the composition of embeddings to an action. On the other hand, other approaches involve separating the agent into a hierarchy where the higher level agent aims to properly coordinating the lower level module. One class of studies focuses on sequentially chaining the modules. [27]–[29] naively arrange the modules to be activated one after another, when [23] provides additional arguments to each primitive as a subgoal with a fixed time horizon. [21] also activates one skill at a time using a high-level choreographer, but the skills and the choreographer share the same feature extraction layer. [22], [30] try to resolve the mismatch between the terminal and initial state distribution of the chained primitives. However, this will restrict the behavior to the prescribed primitive, not allowing the composition or interpolation of multiple primitives to produce new skills.

Thus, another class of studies involves orchestrating the modules in parallel. [16] assemble each attribute from skills by solving the constrained optimization problem, whereas [19] forms a collection of Gaussian policies via multiplications, where each of primitives are independently factored by weights given a state and goals. [31] also composes a mixture policy of subskills by factoring them with weights from the encoder-decoder model consist of bi-directional recurrent neural networks. Notwithstanding that the parallel execution allows the primitives to be activated simultaneously, these works restrict the action space of each primitive to be identical to one another. [3] incorporates agent-specific primitives and provides skill index with behavior embedding from the meta policy to append more agents, but still lacks a methodology on properly fusing multiple skills of each agent.

We build upon the formulation of [19] and further improve it by structuring multiple levels of hierarchical primitives in a recursive fashion. In contrast to the proposed scheme in [19], the role of each of the primitives is known, and the weights distributed over the primitives can display the intention of the overall policy. Moreover, we differentiate the dimensionality of states and actions among primitives and thus, remove the constraints on selecting a proper state and action spaces. This plays an important role when it comes to reusability. We verify this property in the experiments with tasks with and without the gripper angle. Lastly, our novel action composition rule stabilizes the action dimensions that are produced from a single primitive. This is also discussed in the ablation study.

III. HIERARCHICAL PRIMITIVE COMPOSITION

A. Overview

We assume that a single intricate task can be decomposed into a collection of sub-tasks. A primitive is assigned to each sub-task to achieve the goal of the corresponding tasks. These primitives then compose a compound policy with respect to the weights given from a meta-policy, where this compound policy now tries to solve the overall goal of the original task. It is worth noting that the difficulties of tasks are relative, and in some cases, a task once considered to be complex might become a sub-task of even more complex problems. Intuitively, this indicates that a compound

policy can also be a primitive of another compound policy. Therefore, we design our structure in a recursively hierarchical fashion and manually assign a **level** to the compound policy and the corresponding task that indicates how many stacks of modules comprise it. We denote the compound policy, the meta-policy, and the primitive of ith sub-task at level l, by Π_l, π_l^{meta}, and π_{li}, respectively. Our goal here is to learn the optimal π_l^{meta} given primitives such that that maximizes the policy objective of task l.

B. The MDP Formulation

In this section, we formulate the MDP for task l via MDPs of its sub-tasks to induce the optimal policy Π_l^*. We then introduce the meta-MDP, the surrogate of the original MDP. Since the meta-MDP greatly reduces the original action space by mapping compound actions to weights, the scope of the exploration also diminishes and thus increases the sample efficiency. It will later be proven that the optimal policy of this surrogate MDP, π_l^{meta*}, is identical to the original optimal policy, Π_l^*.

To formulate the l-th MDP for task l, we first consider the case for its sub-task i. Sub-task i at l is allocated with its own MDP $M_{li} = \{\mathcal{S}_{li}, \mathcal{A}_{li}, \mathcal{P}_{li}, \mathcal{R}_{li}, \gamma\}$ where each element within the tuple shares the same formulation with the standard MDP setting. M_{li} has its own optimal policy, π_{li}, which we call a primitive. This primitive can either be a flat policy or another compound policy at level $l-1$, Π_{l-1}. Please note here that the state and action space of each primitive within the same task level can differ among others.

Once we define MDPs of all sub-tasks within task l, we then construct a compound MDP for task l, $M_l = \{\mathcal{S}_l, \mathcal{A}_l, \mathcal{P}_l, \mathcal{R}_l, \gamma\}$. The state and action spaces of the compound task l is defined to be a jointed space of its primitives: $\mathbf{s}_l \in \mathcal{S}_l = \bigcup_i \mathcal{S}_{li}$ and $\mathbf{a}_l \in \mathcal{A}_l = \bigcup_i \mathcal{A}_{li}$, respectively. The transition probability $\mathcal{P}_l = \mathcal{P}(\mathbf{s}_l'|\mathbf{s}_l, \mathbf{a}_l)$, the reward function $\mathcal{R}_l = r(\mathbf{s}_l, \mathbf{a}_l)$, and the discount factor γ are the same.

To induce the solution for M_l we employ the surrogate MDP of M_l, a meta-MDP M_l^{meta} defined by a tuple $\{\mathcal{S}_l^{meta}, \Omega_l, \mathcal{P}_l^{meta}, \mathcal{R}_l^{meta}, \gamma\}$. The state space $\mathcal{S}_l^{meta}$ is identical to the $\mathcal{S}_l$ under the assumption that the same information is required to induce the same solution. $[\omega_{l1}, \ldots, \omega_{li}, \cdots] = \boldsymbol{\omega}_l \in \Omega_l$ denotes a collection of weights which represents the measure of the influence of each primitive π_{li} on Π_l. Since the users have the knowledge on to which task the primitives are designated, the weight indicates the purpose of the current action. The weight is the output from the meta-policy π_l^{meta} given state. Although the weight $\boldsymbol{\omega}_l$ theoretically needs not be bounded [19], we restrict its lower bound to be 0, and the sum of all weights to be 1 in order to regard its elements as parameters for a categorical distribution. We later employ the entropy of the meta-policy as the entropy of this distribution to define the maximum entropy objective of the meta-MDP. Weights ω_l are analogous to the actions of conventional MDPs since they lead to the next state. The only difference from the original setup is that the weights are no longer stochastic but are deterministically chosen from each state. $\mathcal{P}_l^{meta} = \mathcal{P}(\mathbf{s}_l'|\mathbf{s}_l, \boldsymbol{\omega}_l)$ is a state transition probability
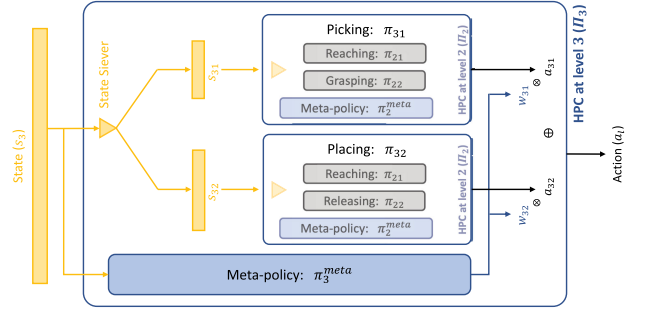
Fig. 2. The Structure of HPC.

of the next state $\mathbf{s}_l'$ given $\mathbf{s}_l$ and $\boldsymbol{\omega}_l$. $\mathcal{R}_l^{meta} = r^m(\mathbf{s}_l, \boldsymbol{\omega}_l)$ is a reward function of a state-weight pair. We define the reward function $r^m(\mathbf{s}_l, \boldsymbol{\omega}_l)$ to be equal to the expectation of the reward of the compound MDP over actions of Π_l given same states and weights (Definition 1). The discount factor γ is the same.

C. Model Construction

The compound policy is composed of two key structures: the meta-policy and primitives, as shown in Fig. 2 . First, the meta-policy receives a state $\mathbf{s}_l$ from the space identical to that of the compound MDP and produces weights $\boldsymbol{\omega}_l$. Simultaneously, the same state $\mathbf{s}_l$ is also given to the state siever. Since the state space from the compound MDP is the union of all the state spaces of the primitives, this layer sieves $\mathbf{s}_l$ into $\mathbf{s}_{li}$ and distributes them to the primitives accordingly. Given the states, the corresponding actions are calculated from the primitives. These actions are then combined with the weights to form an overall action of the compound policy.

D. Action Composition

Each primitive $\pi_{li}(\mathbf{a}_{li}|\mathbf{s}_{li})$ is regarded as a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{li}(\mathbf{s}_{li}), \Sigma_{li}(\mathbf{s}_{li}))$, where $\boldsymbol{\mu}_{li}(\mathbf{s}_{li}) \in \mathbb{R}^{|\mathcal{A}_{li}|}$ and $\Sigma_{li}(\mathbf{s}_{li}) = \text{diag}(\{\sigma_{li,j}^2(\mathbf{s}_{li})\}_{j=1}^{|\mathcal{A}_{li}|})$ denote a mean vector and a diagonal covariance matrix given the state $\mathbf{s}_{li}$, respectively. We formulate the compound policy at level l, Π_l, to be the multiplication of primitives powered by their respective normalized weights. This is related to the product of experts method in the sense that each skill constrains a different subset of the action space conditioned on the states, while their product constrains the combination of them [32]. We weigh the mean of each element of the action dimension with the summation of the weights of primitives to which they activate on. The probability distribution of the j-th action parameter of Π_l, $\Pi_{l,j}$ can be formalized as follows.

$$\Pi_{l,j}(a_{l,j}|\mathbf{s}_l)$$
$$= \frac{1}{Z(\mathbf{s}_l)} \prod_{i=1}^{P} \mathcal{N}_i \left(\mu_{li,j}(\mathbf{s}_{li}) \sum_{c \in \mathcal{C}_j} \omega_c(\mathbf{s}_l), \sigma_{li,j}^2(\mathbf{s}_{li}) \right)^{\hat{\omega}_i} \quad (1)$$

where $\mathcal{C}_j$ denotes a set of primitives that include the j-th parameter within their action space, P the total number of primitives of task l, and $\hat{\omega}_i = \omega_i(\mathbf{s}_l)/\sum_{p=1}^{P} \omega_p(\mathbf{s}_l)$ the normalized weight
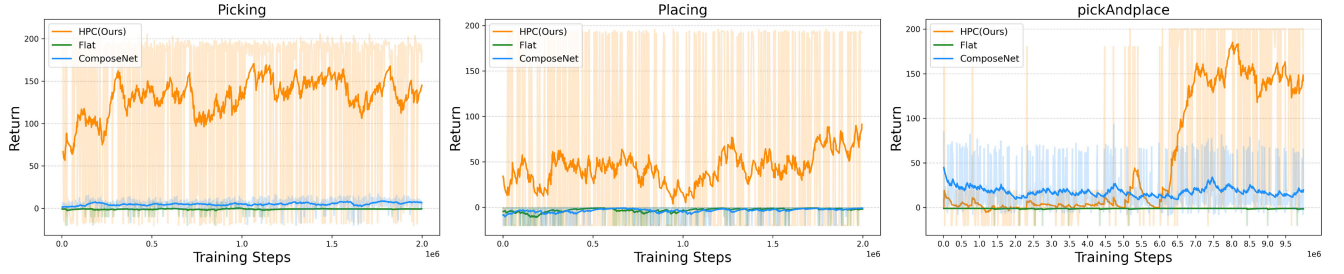
Fig. 3. Training curves of each tasks. The x-axis shows the number of training steps and y-axis represents the episode return.
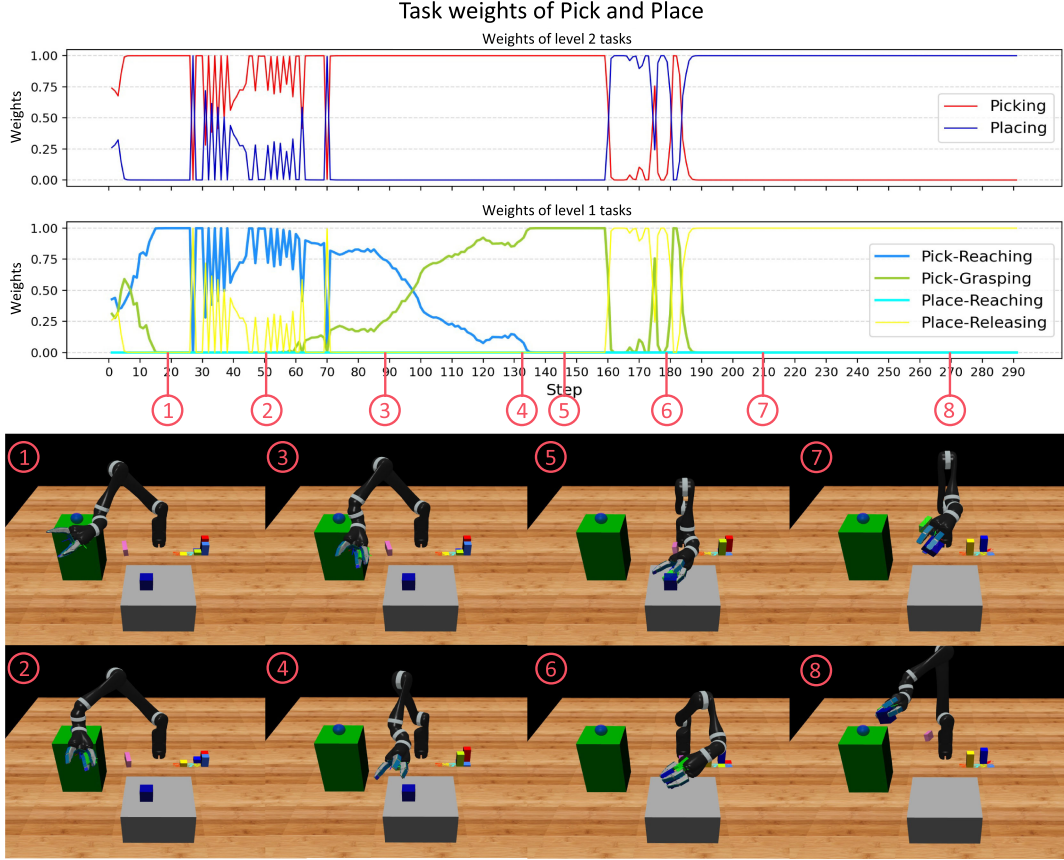

Fig. 4. Weight examination of the pick and place task. **Top:** Weigth plot of the level 2 and level 1 tasks, respectively. **Bottom:** The visualization of the simulation at the corresponding timestep. The video of the pick and place task can be found in https://youtu.be/_MlKP80sVtE.

over the sum. Since the multiplication of Gaussian distributions powered by some factor yields another Gaussian distribution [19], it allows the resulting policy to be used as another primitive at a higher level. The component-wise parameters for the composed Gaussian distribution can be derived as follows.

$$
\mu_{l,j} = \frac{\sum_{c \in \mathcal{C}_j} \omega_c}{\sum_{p=1}^{P} \omega_p / \sigma_{lp,j}^2} \sum_{i=1}^{P} \frac{\omega_i}{\sigma_{li,j}^2} \mu_{li,j},
$$

$$
\sigma_{l,j}^2 = \frac{\sum_{c \in \mathcal{C}_j} \omega_c}{\sum_{p=1}^{P} \omega_p / \sigma_{lp,j}^2} \tag{2}
$$

where $\mu_{l,j}, \sigma_{l,j}, \omega_c, \omega_p, \omega_i$ are functions of $\mathbf{s}_l$ and $\mu_{li,j}, \sigma_{li,j}, \sigma_{lp,j}$ are functions of $\mathbf{s}_{li}$. Intuitively, the mean of the

compound policy indicates the sum of mean of each primitive factored by their weights over variance, and re-weighted by the summed weights which are relevant to the respective action dimension. The compound variance is the reciprocal of the summation of weights over variances and again, re-weighted by the relevant weights.

E. Training Procedure

The training of HPC is sequentially done from level 2 and iteratively stacks up the learned policies as the level increases. The primitives at level 1 are flat since no compositions occur below that level. To train the compound policy at level l, we assume that it is accessible to all primitives that consist of

the compound policy. Although which primitive to be used is manually determined by the user, the required engineering effort is minimal as the meta-policy learns how to take advantage of the primitives regardless of their role. This can be verified in Section V-B2.

For the meta-policy optimization, any RL methods that are capable of maximizing the meta-objective function defined in (5) can be applied, while we used Soft-Actor-Critic (SAC) in this letter [33]. This can easily be done by regarding weights as an action. The parameters of the primitives are kept fixed to prevent optimization instabilities [34] and invalid off-policy samples [35].

IV. OPTIMAL POLICY EQUIVALENCE

In this section, we first describe the maximum entropy objective of our MDP setting at task l, and show that the objective of the meta-MDP is identical to that of the original MDP. Then, we prove that the solution for the meta-MDP applies to the standard MDP as well under several assumptions.

The objective of reinforcement learning corresponds to the accumulated sum of discounted rewards as follows [36].

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k \sim \mathcal{P}, \mathbf{a}_k \sim \pi}[r(\mathbf{s}_k, \mathbf{a}_k)] \right]$$

To formulate the objective of our MDP, we substitute the standard policy π with the compound policy, Π. In addition, to incentivize the uncertainty of the meta-policy, we introduce the entropy of the primitive weights and formulate it based on a maximum entropy formulation. We omit the subscript l hereafter for simplicity and rewrite the objective as follows.

$$J(\Pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\Pi} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k \sim \mathcal{P}, \mathbf{a}_k \sim \Pi}[r(\mathbf{s}_k, \mathbf{a}_k)] \right.$$
$$\left. + \alpha \mathcal{H}(\boldsymbol{\omega}_k)|\mathbf{s}_t, \mathbf{a}_t] \right] \tag{3}$$

where $H(\boldsymbol{\omega})$ denotes the entropy of the categorical distribution with $\boldsymbol{\omega}$ as its parameter. The entropy term desires the case where the weights are equally distributed to the primitives, and not biased.

Prior to express (3) with the meta-policy, we first define the reward function of the meta-MDP as discussed in Section III-B.

Definition 1 (Meta Reward Function): For any state $\mathbf{s} \in \mathcal{S}$, weight $\boldsymbol{\omega} = \pi^{meta}(\mathbf{s}) \in \Omega$, and the compound policy defined by the weights $\Pi(\boldsymbol{\omega})$, the reward function of the meta-MDP is defined by

$$r^m(\mathbf{s}, \boldsymbol{\omega}) := \mathbb{E}_{\mathbf{a} \sim \Pi(\boldsymbol{\omega})}[r(\mathbf{s}, \mathbf{a})|\mathbf{s}] \tag{4}$$

We next define the objective of the meta-MDP as follows.

Definition 2 (Meta Objective Function): The objective of the meta-MDP given a meta-policy is defined by

$$J(\pi^{meta}) = \sum_{t=0}^{\infty} \mathbb{E}_{\mathbf{s}_t \sim \mathcal{P}} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k \sim \mathcal{P}}[r^m(\mathbf{s}_k, \boldsymbol{\omega}_k) \right.$$

$$\left. + \alpha \mathcal{H}(\boldsymbol{\omega}_k)|\mathbf{s}_t] \right] \tag{5}$$

where, $\boldsymbol{\omega}_k = \pi^{meta}(\mathbf{s}_k)$

From the above definitions, we now can show that the objective of the compound policy is equivalent to that of the meta-policy.

Lemma 1 (Objective Function Equivalence): From Definitions 1 and 2, we obtain the following equivalence.

$$J(\pi^{meta}) = J(\Pi) \tag{6}$$

Proof: See Appendix A. ∎

We assume that there exists an optimal compound policy which behaves identical to the optimal standard policy as follows.

Assumption 1: For any MDPs, there exists an optimal compound policy Π^* such that

$$J(\Pi^*) = J(\pi^*) \tag{7}$$

where π^* is the optimal standard policy.

This assumption guarantees that once we find the compound policy which optimizes the compound policy objective function, it is the optimal solution of the MDP, π^*.

Finally, it can be straightforwardly shown that the below satisfies.

Theorem 1 (Optimal Policy Equivalence): From Lemma 1 and Assumption 1, for the optimal meta-policy π^{meta*} which optimizes the objective (5),

$$\pi^{meta*} = \pi^* \tag{8}$$

Proof: From Lemma 1, the objective of the optimal meta-policy and the optimal compound policy is the same, thus from Assumption 1, the following holds.

$$J(\pi^{meta*}) = J(\Pi^*) = J(\pi^*) \tag{9}$$

∎

From Theorem 1, it becomes possible to devise an algorithm which seeks for the optimal meta-policy instead of the standard policy.

V. EXPERIMENT

A. Experimental Setup

As shown in Fig. 1, we have newly designed the environment for the following six skills composing `pick-and-place`. The descriptions of each tasks are as follows: (1) **Reaching**: The end-effector moves from the initial pose to the target pose. (2) **Grasping**: To grasp the target object and lift it above 0.1 m. Since `grasping` is intended to focus solely on grasping the object, we bound the initial position of the end-effector to be within 0.15 m to the object. (3) **Releasing**: To release the object at the destination. Since `releasing` is intended to focus solely on releasing the object, the initial pose is sampled near the destination with the object in hand. (4) **Picking**: Identical to `grasping` except for its initial pose, as it starts farther away from the object. (5) **Placing**: Identical to `releasing` except for its initial pose, as it starts farther away from the object. (6)

Pick-and-place: To pick up an object and place it at the given destination.

To validate the effectiveness of HPC on composing skills with different action dimensions, we differentiate the dimensionality of `releasing` with the others by assigning 6 dimensions (incremental pose of the end-effector), while others include gripper angle as an additional component of the controllable action. We refer the readers to Table I for the details regarding

TABLE I
STATES OF EACH TASK

Task	States
Reaching	$\mathbf{p}_E, \mathbf{q}_E, \mathbf{p}_r, \mathbf{q}_r$
Grasping	$i_E, \mathbf{p}_E, \mathbf{q}_E, \theta_g, \mathbf{p}_{obj}$
Releasing	$i_E, \mathbf{p}_E, \mathbf{q}_E, \theta_g, \mathbf{p}_{obj}, \mathbf{p}_d$
Picking	$i_E, \mathbf{p}_E, \mathbf{q}_E, \theta_g, \mathbf{p}_{obj}, \mathbf{p}_r, \mathbf{q}_r$
Placing	$i_E, \mathbf{p}_E, \mathbf{q}_E, \theta_g, \mathbf{p}_{obj}, \mathbf{p}_d, \mathbf{p}_r, \mathbf{q}_r$
Pick-and-place	$i_E, \mathbf{p}_E, \mathbf{q}_E, \theta_g, \mathbf{p}_{obj}, \mathbf{p}_d, \mathbf{p}_r, \mathbf{q}_r$

state dimensions of each skill.

All the environments are created within the MuJoCo physics simulator [37]. For the manipulator, the 6-DoF Jaco2 model from Kinova Robotics is used. To control the end-effector, we used an operational space controller (OSC) within the abr_control package provided from Applied Brain Research [38], [39]. OSC receives the target pose of the end-effector and calculates required torques to be applied to each joint of the manipulator.

B. Results and Analysis

To examine the effects of HRL when solving the complex task, we compare HPC to the vanilla SAC denoted as Flat [33]. To evaluate the effects of pretrained primitives in terms of the performance, we compare HPC to ComposeNet [26]. To apply ComposeNet to our setting, we detached the last layer of the primitives from our experiment and used the remaining networks as skill trunks. We then stacked a compositional layer along with a policy layer on top of the fixed skill trunks. Shared policy layers are not used.

1) Training: The left and the center plots from Fig. 3 show the training curves of both level 2 tasks - `picking` and `placing`, respectively. The results show that HPC outperforms the baselines by far, while both Flat and ComposeNet do not experience a successful episode. The right plot from Fig. 3 shows the training curve of `pick-and-place` task. It shows that HPC outperforms both baselines in the level 3 task as well. Since the baselines did not manage to solve the lower-level tasks, it is evident that the level 3 task is also impossible to be solved. Although ComposeNet leverages the pretrained skill trunks to obtain skill embeddings, it still requires its policy and the value function to be optimized for the original action space and thus, suffers from the curse of dimensionality. HPC, however, abstracted its scope of action from the {pose, gripper angle} selection to the weight selection problem and managed to solve the task.

2) Task Performance: We further examine the performance of HPC with a few frames from `pick-and-place` task, as
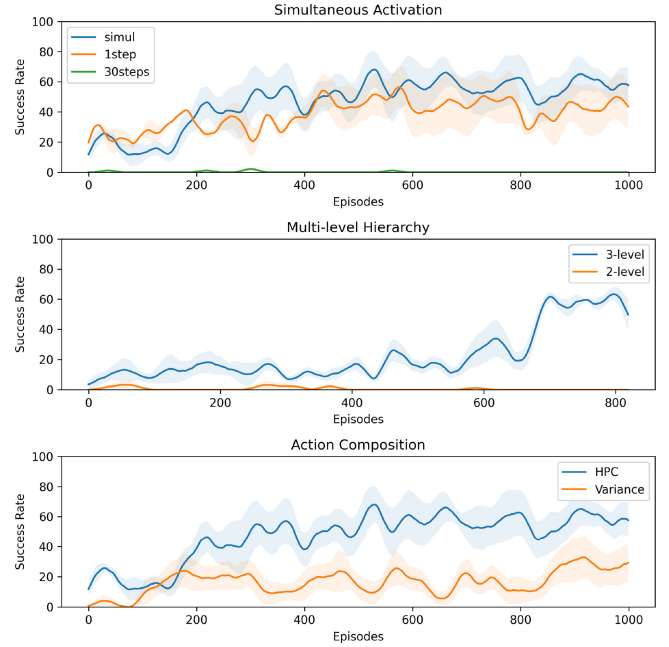

Fig. 5. Success rate of the ablation studies. Top plot shows the effect of the simultaneous activation, plot in the middle shows the effect of the multi-level hierarchy, and the bottom plot shows the effect of our action composition.

shown in Fig. 4. The weights distributed to each primitive of level 2 and level 1 are plotted in the upper and the lower part of the weight plot, respectively. In the beginning, the reaching primitive is mainly used to drive the end-effector towards the object (①). While approaching, the placing primitive is used to properly orient the end-effector (②). This addresses that the meta-policy takes advantage of the primitives within the skill set regardless of their role. It is also worth emphasizing that the weight is not binarily biased to either one of the primitives but rather to take advantage of both. Once the end-effector is well oriented, the picking primitive is fully used until the end-effector successfully grasps the object and lifts it above the table (③–⑤). With the object in the gripper, the meta-policy now leverages the placing primitive and completes the overall task (⑥–⑧).

C. Ablation Study

Our next experiment investigates the effects of simultaneous activation, multi-level hierarchy and our action composition.

1) Simultaneous Activation: We first compare the performance of the conventional HPC on `picking` task with the binary selection of skills with the fixed time horizon of 1 and 30 steps. The former is analogous to the work of [27], [40], where the primitives are either competing or selected to be activated at a given timestep, whereas the latter is to the work of [23] where the higher-level agent assigns which skill to be selected along with the action parameters. From the top plot of Fig. 5, it can be shown that the simultaneous activation outperforms both of the binary selection setups.

2) Multi-Level Hierarchy: The following study analyzes the effect of the hierarchy of multiple levels by composing `pick-and-place` task with level 1 primitives. As the plot in the

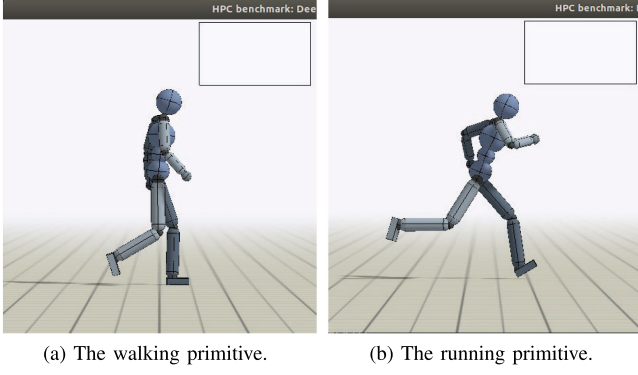| (a) The walking primitive.	(b) The running primitive.

Fig. 6. The walking and running primitives of the humanoid robot.

middle of Fig. 5 represents, adding more levels to the complex task makes the learning efficient. This phenomenon is also in relation to curriculum learning, as the intermediate levels work as stepping stones to guide the agent to seek the beneficial path.

3) Action Composition: Lastly, we verify the effect of our action composition rule by comparing the performance of `picking` task with anoother composition method [19]. Since the reaching skill does not optimize the gripper-angle, it is possible to add an additional action dimension with high variance to the learned Gaussian. From the bottom plot of Fig. 5, it can be seen that the proposed action composition method outperforms the opponent. Given high variance, this is due to the gripper angle of the compound action being identical to that of the grasping primitive when computed according to the formulation from [19]. When the policy encounters the state region in which the grasping primitive was not trained, the action becomes unstable.

D. Failed Case: Operational State Space Mismatch

In this experiment, we show how HPC can fail to learn the task if the primitives do not share the operational state Space space. With the priorly trained primitive for the humanoid robot walking and running, as shown in Fig. 6, we tried to train the robot to jog at a velocity between walking and running by composing two primitives. Although both primitives share the positional state space of the joint motor, the operational state space of the joint velocity differed. Moreover, the region of the velocity space required for the jogging is between the two primitives, and the meta-policy was not able to seek the optimal action. This will be discussed in the following section.

VI. CONCLUSION

In this letter, a concept of hierarchical primitive composition is proposed, where each of the known predefined/pretrained primitives is assembled in a hierarchy to construct a compound overall policy. HPC dramatically reduces the learning time of the complicated robotic tasks by imploying the surrogate MDP. It can represent the intention of the agent by weights distributed to each of the primitives in a human-interpretable manner while reusing pretrained primitives to enhance sample efficiency and guarantee stability in the optimization process. In addition, since

HPC is not restricted to the dimensionality of the state and action spaces, any other works on RL can be applied to compose its primitive with pretrained policies.

There still remain future works to do. Firstly, Assumption 1 limits the applicability of HPC to the case where the optimal policy is in the set of the compound policies. If there are not enough skills within the set of primitives, the assumption does not hold. Therefore, it is required to employ a newly trainable auxiliary primitive, which closes the gap between the compound policy and the optimal policy. Second, if the primitives do not share the same region of the operational state state space, the compound policy should bias its weight signal on a single primitive activated at the moment to output a proper action. This can be observed from the experiment discussed in Section V-D. This can be mitigated with the work from [22] to properly match the operational state space of primitives.

APPENDIX A
PROOF OF OBJECTIVE FUNCTION EQUIVALENCE

Lemma 1 (Objective Function Equivalence): From Definitions 1 and 2, we obtain the following equivalence.

$$J(\pi^{meta}) = J(\Pi) \tag{10}$$

Proof: We first rewrite the compound objective as follows.

$$J(\Pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\Pi} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k \sim \mathcal{P}, \mathbf{a}_k \sim \Pi}[r(\mathbf{s}_k, \mathbf{a}_k) \right.$$
$$\left. + \alpha \mathcal{H}(\boldsymbol{\omega}_k)|\mathbf{s}_t, \mathbf{a}_t] \right] \tag{11}$$

By the linearity of the expectation, (11) becomes

$$\sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\Pi} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k \sim \mathcal{P}, \mathbf{a}_k \sim \Pi}[r(\mathbf{s}_k, \mathbf{a}_k)|\mathbf{s}_t, \mathbf{a}_t] \right.$$
$$\left. + \mathbb{E}_{\mathbf{s}_k \sim \mathcal{P}}[\alpha \mathcal{H}(\boldsymbol{\omega}_k)|\mathbf{s}_t] \right] \tag{12}$$

Note here that the entropy of a categorical distribution of the weights $\mathcal{H}(\boldsymbol{\omega}_k)$ is independent to the actions, and thus the expectation over the action can be omited. From Definition 1, (12) now becomes

$$\sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k}[r^m(\mathbf{s}_k, \boldsymbol{\omega}_k)|\mathbf{s}_t] \right.$$
$$\left. + \mathbb{E}_{\mathbf{s}_k}[\alpha \mathcal{H}(\boldsymbol{\omega}_k)|\mathbf{s}_t] \right]$$
$$= \sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k}[r^m(\mathbf{s}_k, \boldsymbol{\omega}_k) + \alpha \mathcal{H}(\boldsymbol{\omega}_k)|\mathbf{s}_t] \right]$$
$$= \sum_{t=0}^{\infty} \mathbb{E}_{\mathbf{s}_t} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{\mathbf{s}_k}[r^m(\mathbf{s}_k, \boldsymbol{\omega}_k) + \alpha \mathcal{H}(\boldsymbol{\omega}_k)|\mathbf{s}_t] \right] = J(\pi^{meta}) \tag{13}$$

APPENDIX B
STATE DIMENSIONS

where

$i_E \in \{0, 1\}$: The touch indicator

$\mathbf{p}_E \in \mathbb{R}^3$: The position of the end-effector (m)

$\mathbf{q}_E \in \mathbb{R}^3$: The orientation of the end-effector (rad)

$\theta_g \in \mathbb{R}$: The finger angle of the gripper (rad)

$\mathbf{p}_{obj} \in \mathbb{R}^3$: The position of the target object (m)

$\mathbf{p}_d \in \mathbb{R}^3$: The orientation of the destination (rad)

$\mathbf{p}_r \in \mathbb{R}^3$: The position of the reaching target (m)

$\mathbf{q}_r \in \mathbb{R}^3$: The orientation of the reaching target (rad)

REFERENCES

[1] O. Kilinc and G. Montana, "Reinforcement learning for robotic manipulation using simulated locomotion demonstrations," *Mach. Learn.*, vol. 111, no. 2, pp. 465–486, 2022.

[2] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 2, no. 4, pp. 2096–2103, Oct. 2017.

[3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Sci. Robot.*, vol. 5, no. 47, p. eabc5986, 2020.

[4] M. Kim, S. Lee, J. Lim, J. Choi, and S. G. Kang, "Unexpected collision avoidance driving strategy using deep reinforcement learning," *IEEE Access*, vol. 8, pp. 17243–17252, 2020.

[5] M. Kim, J. Seo, M. Lee, and J. Choi, "Vision-based uncertainty-aware lane keeping strategy using deep reinforcement learning," *J. Dyn. Syst., Meas., Control*, vol. 143, no. 8, p. 084503, 2021.

[6] J. Lim, S. Ha, and J. Choi, "Prediction of reward functions for deep reinforcement learning via gaussian process regression," *IEEE/ASME Trans. Mechatronics*, vol. 25, no. 4, pp. 1739–1746, Aug. 2020.

[7] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 8234–8244.

[8] R. Yang, H. Xu, Y. Wu, and X. Wang, "Multi-task reinforcement learning with soft modularization," *Adv. Neural Inf. Process. Syst.*, vol. 33, 2018, pp. 4767–4777, 2020.

[9] J. Li *et al.*, "Multi-task batch reinforcement learning with metric learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 6197–6210.

[10] A. Zhang, N. Ballas, and J. Pineau, "A dissection of overfitting and generalization in continuous reinforcement learning," 2018, *arXiv:1806.07937*.

[11] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 2169–2176.

[12] C. Scheiderer, M. Mosbach, A. F. Posada-Moreno, and T. Meisen, "Transfer of hierarchical reinforcement learning structures for robotic manipulation tasks," in *Proc. Int. Conf. Comput. Sci. Comput. Intell.*, 2020, pp. 504–509.

[13] A. A. Rusu *et al.*, "Progressive neural networks," 2016, *arXiv:1606.04671*.

[14] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 166–175.

[15] A. Das, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, "Neural modular control for embodied question answering," in *Proc. Conf. Robot Learn.*, 2018, pp. 53–62.

[16] Z. Xu, H. Chang, C. Tang, C. Liu, and M. Tomizuka, "Toward modularization of neural network autonomous driving policy using parallel attribute networks," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2019, pp. 1400–1407.

[17] B. Beyret, A. Shafti, and A. A. Faisal, "Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 5014–5019.

[18] Y. Lee, J. Yang, and J. J. Lim, "Learning to coordinate manipulation skills via skill behavior diversification," in *Proc. Int. Conf. Learn. Representations*, 2019.

[19] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "MCP: Learning composable hierarchical control with multiplicative compositional policies," *Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 3681–3692, 2019.

[20] L. Hasenclever, F. Pardo, R. Hadsell, N. Heess, and J. Merel, "Comic: Complementary task learning & mimicry for reusable skills," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 4105–4115.

[21] A. Pore and G. Aragon-Camarasa, "On simple reactive neural networks for behaviour-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 7477–7483.

[22] Y. Lee, J. J. Lim, A. Anandkumar, and Y. Zhu, "Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization," in *Proc. Conf. Robot Learn.*, 2022, pp. 406–416.

[23] M. Dalal, D. Pathak, and R. R. Salakhutdinov, "Accelerating robotic reinforcement learning via parameterized action primitives," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, vol. 34, pp. 21847–21859.

[24] F. Alet, T. Lozano-Pérez, and L. P. Kaelbling, "Modular meta-learning," in *Proc. Conf. Robot Learn.*, 2018, pp. 856–868.

[25] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[26] H. Sahni, S. Kumar, F. Tejani, and C. Isbell, "Learning to compose skills," *NIPS Deep RL Symp.*, 2017.

[27] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," in *Proc. Int. Conf. Learn. Representations*, 2017.

[28] J. Merel *et al.*, "Hierarchical visuomotor control of humanoids," in *Proc. Int. Conf. Learn. Representations*, 2018.

[29] L. Marzari, A. Pore, D. Dall'Alba, G. Aragon-Camarasa, A. Farinelli, and P. Fiorini, "Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks," in *Proc. 20th Int. Conf. Adv. Robot.*, 2021, pp. 640–645.

[30] A. Clegg, W. Yu, J. Tan, C. K. Liu, and G. Turk, "Learning to dress: Synthesizing human dressing motion via deep reinforcement learning," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 1–10, 2018.

[31] A. H. Qureshi, J. J. Johnson, Y. Qin, T. Henderson, B. Boots, and M. C. Yip, "Composing task-agnostic policies with deep reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2020.

[32] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[34] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," in *Proc. Int. Conf. Learn. Representations*, 2017.

[35] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 3303–3313.

[36] T. Haarnoja, "Acquiring diverse robot skills via maximum entropy deep reinforcement learning," Ph.D. dissertation, Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, 2018.

[37] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.

[38] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robot. Autom.*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

[39] T. DeWolf, P. Jaworski, D. Rasmussen, and T. Bekolay, "ABR control," 2017. [Online]. Available: https://github.com/abr/abr_control

[40] A. Goyal, S. Sodhani, J. Binas, X. B. Peng, S. Levine, and Y. Bengio, "Reinforcement learning with competitive ensembles of information-constrained primitives," in *Proc. Int. Conf. Learn. Representations*, 2020.