Abstract—Deep reinforcement learning has shown its effectiveness in various applications and provides a promising direction for solving tasks with high complexity. In most reinforcement learning algorithms, however, two major issues need to be dealt with—the sample inefficiency and the interpretability of a policy. The former happens when the environment is sparsely rewarded and/or has a long-term credit assignment problem, while the latter becomes a problem when the learned policies are deployed at the customer side product. In this paper, we propose a novel hierarchical reinforcement learning algorithm that mitigates the aforementioned issues by decomposing the original task in a hierarchy and by compounding pretrained primitives with intents. We show how the proposed scheme can be employed in practice by solving a pick and place task with a 6 DoF manipulator.

Index Terms—Hierarchical reinforcement learning, deep reinforcement learning, symbolic artificial intelligence, hybrid artificial intelligence, interpretability.

I. INTRODUCTION

Deep reinforcement learning (deep RL) has shown its effectiveness when applied to tasks of various domains ranging from video games [1] to animations [2] and to the field of robotics within simulations [3] or in the real world [4]–[8]. Despite the performance enhancement benefited from deep RL, sample inefficiency remains a major challenge, especially for the environments with long-term credit assignment problems. It only exacerbates when the problem formulation becomes sparsely rewarded where we do not learn much from most of the interactions between the agent and the environment.

To tackle the aforementioned issues, various studies regarding deep RL from many perspectives have been actively conducted. Off-policy optimization algorithms using batch sampling from replay buffers allow multiple learnings to take place between roll-outs, but suffer from sample variances in turn [9]–[11]. Methods with hindsight alleviate sample variances by allowing the agent to learn from older data stored in the replay buffer by relabeling them [12], [13]. Apart from leverage on old data, exploiting auxiliary tasks or rewards to the unrewarded experiences also shows promising results on the speed of convergence. Employing unsupervised auxiliary tasks to maximize pseudo-reward by training separate policies, or regards controlled features as intrinsic motivation are tried as well [14]–[16]. Studies with approximate Bayes optimal planning provide a way to sample efficient learning by using Monte-Carlo tree search [17]. Although the aforementioned works show prominent progress on sample efficiency, there remain some limitations regarding the use of a single policy, e.g., lack of reusability, composability, or transparency.

To deal with such problems, meta-learning or hierarchical reinforcement learning (HRL) has emerged where these approaches provide options [18]–[22] or subgoals [23], [24] to the policy to abstract over sequences of primitive actions [23]. Another aspect of HRL chooses a sequence of actions among multiple policies and activate the selected one for the fixed/varying time window [25], [26]. These works share similarity with RL using behavior tree in a sense of learning a high-level agent that activates the proper low-level agent [27], [28]. HRL can be fused with other works of standard RL where [29] employs hindsight, [30] uses an off-policy method for optimizing, [31] applies auxiliary reward to train HRL. HRL is effective, but notorious for its necessity of task-specific state/reward manipulation and unstabilized optimization algorithm. In addition, most HRL schemes mainly focus on sequentially divided tasks such as the higher-level policy selecting lower policy correspondent to the piecewise task [1], but in practice, the vast portion of tasks are not sequentially dividable, but rather need to work together optimally orchestrated.

Another aspect for the deep RL to be considered is its model interpretability. The deep learning community has been addressing the importance of eXplainable Artificial Intelligence (XAI) where the related studies have emerged long before and still ongoing. As a matter of course, attaining interpretability in deep reinforcement learning which gives a proper reasoning of what the agent intends to users has also been widely discussed [32]–[36]. However, there exists some limitations. In terms of explainability of HRL, it is either using the physical property as a subgoal or using latent vector as an option which does not properly convince users on intentions of the model [13]. Hybrid artificial intelligence, an architecture which combines rule-based symbol manipulation with neural networks, tries to employ abstracted representations or patterns from data into a systematic framework on which user have prior knowledge [37].

This paper proposes a concept of Hierarchical Primitive Composition (HPC), where each of the known prede-
fined/pretrained primitives are assembled in a hierarchy to construct an overall compound policy. The contributions of this work are summarized as follows. First, the newly proposed HPC can successfully tackle a complex task by multiple hierarchies that build on a collection of primitives. Since HPC is not restricted to the dimensionality of the state and action spaces, any other previous works on RL can be applied to compose its primitives with pretrained policies, i.e., the increased level of reusability and modularization. Next, HPC can represent the current purpose of the agent by intents and subgoals associated to each of the primitives in a human-interpretable manner, i.e., XAI capability. 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). Lastly, since HPC composes modularized blocks of primitives, other works on recognition and logic using neural networks can be fused into this work and further structuralize the total artificial intelligence as a whole.

The paper is organized as follows. In Section II we illustrate background works that we build upon. In Sections III and IV we present our approach with technical details. Section V describes the comparative simulation setups and the results. Finally, Section VI provides discussions and concluding remarks along with our future works.

II. RELATED WORK

A. Hierarchical Reinforcement Learning

The studies of HRL are mainly classified into two different categories: selector-based and subgoal based methods [18–31]. Our work fuses subgoal and selector-based HRL methods, but does not leverage on the options framework where a single policy network learns and operates over different levels of temporal abstraction given options from the meta-policy [38]. Although options may abstract the action space, it is not interpretable by the human in a physical sense. Instead, we incorporate subgoals, either produced from the meta-policy or given by the user, which has been manually designated to certain primitive as a semantic state. Multiplicative compositional policies (MCP) forms a collection of Gaussian policies via multiplications, where \( k \) primitives are independently factored by weights \( w_i \) given a state and goals as follows [39].

\[
\pi(a|s, g) = \frac{1}{Z(s, g)} \prod_{i=1}^{k} \pi_i(a|s)^{w_i(s, g)}
\]  

Since each of \( \pi_i \) is assumed to be a Gaussian distribution, the overall composition is also induced to be a Gaussian distribution with a partition function \( Z(s, g) \) as a normalizer. Differently from additive models, MCP conducts simultaneous activations of primitives, while the gating function provides the number of influences of each action. We build upon the formulation of the multiplicative Gaussian policy and further improve it by structuring multiple levels of hierarchical primitives in a recursive fashion. In contrast to the original MCP paper, 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, reduce the constraints on selecting a proper state and action spaces. This plays an important role when it comes to the reusability. A policy trained with a robot of a fixed degree of freedoms (DoF) can be reused in a task which involves the robot with additional peripherals to produce more DoF. We verify this property in the experiments with tasks with and without the gripper angle.

B. Explainable Reinforcement Learning

The necessity of explainability behind the learned model has long been raised in the field of deep RL. This, however, faces a major challenge of performance-transparency trade-off; adverse characteristics of model performance decrease as complexity increases. [40], [41] propose a multi-task modular RL framework which reserves each task with human instructions while [32–34] exploit Q-values as a means of interpretability. Studies regarding hierarchical RL with sub-task decomposition also tackles the same problem where [35] produces subgoals for a low-level agent to maximize the reward and [36] further expands this work to provide an explanation with natural language encoding. [37] introduces hybrid artificial intelligence, which either leverages abstracted informations (or symbols) induced from raw data using neural networks to make decisions from rule-based framework, or inversely, define rules prior to select which neural networks to use. Our work exploits some of the precious characteristics from the preceding works such as assigning a known task to a module or arranging subgoals to low-level agents.

III. HIERARCHICAL PRIMITIVE COMPOSITION

In this section, we describe the framework of the hierarchical primitive composition and its training algorithm. We first define the MDP structure of the unit compound task and expand this concept as a hierarchical task formulation in a recursive manner. We then describe how the compound policy is constructed from the primitives according to the defined MDP and explain the data flow of the model structure. Next, we explain the finalized action generated by exploiting primitive actions and its hierarchical policy design. Lastly, we devise a training algorithm of HPC by reformulating the maximum entropy objectives along with the existing soft-Q and -V functions.

A. The Structure of HPC

Our model consists of a modularized composition of primitives. We assume that a single intricate task can be decomposed into a collection of sub-tasks. A primitive is designated to each sub-task to achieve the goal of the assigned tasks. These primitives then compose a compound policy that solves the overall goal the original task would like to accomplish. However, the difficulty of tasks is relative, and in some cases, a task once was considered to be complexed, might itself become a sub-task of even more complex problems. Therefore, we design our structure in a recursively hierarchical fashion and assign a level to each task as the module of composite
primitives stacks up. We denote the compound policy at tasks level \( l \), the meta-policy, and the corresponding primitive of sub-task index \( i \), by \( \Pi_l, \pi^\text{meta}_l \), and \( \pi_{li} \), respectively. Figure 1 shows how the primitives at task level \( l \) map themselves to each sub-tasks.

**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 \( \Pi^*_l \). We then introduce the meta-MDP, the surrogate MDP of the original problem, to solve the original MDP. It will later be proven that the optimal policy of this surrogate MDP \( \pi^\text{meta}_l \) is identical to the original optimal policy, \( \Pi^*_l \).

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 \( M_i = \{ S_i, A_i, \mathcal{P}_l, R_{li}, \gamma \} \) where each element within the tuple shares the same formulation with the standard MDP setting. \( M_i \) has its own optimal policy, \( \pi_{li} \), which we call a primitive. This primitive can either be a base policy or another compound policy at level \( l - 1 \), \( \Pi_l-1 \). 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 an MDP for task \( l \), \( M_l = \{ S_l, A_l, \mathcal{P}_l, R_l, \gamma \} \). The state and action spaces of the compound task \( l \) is defined to be a jointed space of its primitives: \( s_l = \bigcup_i S_i \) and \( a_l \in A_l = \bigcup_i A_{li} \), respectively. The transition probability \( \mathcal{P}_l = \mathcal{P}(s'_{li}|s_{li}, a_{li}) \), the reward function \( R_l = r(s_{li}, a_{li}) \), and the discount factor \( \gamma \) are the same.

To induce the solution for \( M_l \) we employ the surrogate MDP of \( M_i \), a meta-MDP \( M^\text{meta}_l = \{ S^\text{meta}_l, \Omega_l \oplus G^l, \mathcal{P}^\text{meta}_l, R^\text{meta}_l, \gamma \} \) \( \oplus \) denotes a concatenation operator which links two vectors together when used with vectors (i.e., for \( a \in \mathbb{R}^n \) and \( b \in \mathbb{R}^m \), \( a \oplus b = c \in \mathbb{R}^{n+m} \)), or works as a set construction operator which outputs a set with elements concatenated from the elements of two set operands when used with sets (i.e., for \( A = \{ a \mid a \in \mathbb{R}^n \} \) and \( B = \{ b \mid b \in \mathbb{R}^m \} \), \( A \oplus B = C = \{ c \mid c = a \oplus b \} \) for any \( a \) and \( b \).

1) **State:** The state space \( S^\text{meta}_l \) is identical to the \( \omega_s \) under the assumption that the same information is required to induce the same solution.

2) **Intent:** \( [\omega_{l1}, \cdots, \omega_{li}, \cdots] = \omega_l \in \Omega_l \) denotes a collection of intents which represents the measure of the influence of each primitive \( \pi_{li} \) on \( \Pi_l \). Since the users have the knowledge on to which task the primitives are designated, the intent indicates the purpose of the current action. The intent is the output from the meta-policy given state. Although the intent \( \omega_l \) theoretically needs not be bounded [39], we restrict its lower bound to be 0, and the sum of all intents 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.

3) **Subgoal:** A subgoal is a property which can replace the element of the state given to each primitives that works as a target. For example, consider a primitive designated to a task of a humanoid robot walking at a target velocity. Since the robot needs to know at which speed it should operate, the target velocity will be included in the state. The subgoal can then be the target velocity.

There are two types of subgoal. First, \( g_{m1}^{\pi} \oplus \cdots \oplus g_{mn}^{\pi} \) denotes a subgoal produced from the meta-policy. It is a concatenation of all subgoals from \( \pi^\text{meta} \) distributed to any primitives which requires one. As in intent, \( g^\pi_l \) is likewise produced from the meta-policy given a state. It can be a null vector if no primitive leverages on subgoals from the meta-policy (i.e., \( \forall i, g^\pi_l = \emptyset \)). Along with \( g^\pi_l \), \( g^\omega_l \) denotes a subgoal given from the environment. \( g^\omega_l \) can be manually provided from the user, hard-coded within the environment, or produced from other function approximators as a feature that the primitive should be directed to.

4) **Intent and Subgoal:** The concatenation of intents and subgoals \( \Omega_l \oplus G^l \) is analogous to the actions of conventional MDPs since intents and subgoals lead to the next state. The only difference from the original setup is that the intents and subgoals are no longer stochastic and are deterministically chosen from each state. Note here that \( g^\omega_l \) is not included, since the subgoal from the environment is regarded as a state. A detailed explanation of how intents and subgoals compose actions of \( \Pi_l \) will be explained in Sections IIIC and IIID.

5) **Other Components of the meta-MDP:** \( \mathcal{P}^\text{meta}_l = \mathcal{P}(s^\text{meta}_l|s_l, \omega_l \oplus g^\omega_l) \) is a state transition probability of the next state \( s^\text{meta}_l \) given \( s_l \) and \( \omega_l \oplus g^\omega_l \). \( R^\text{meta}_l = r^\text{meta}_l(s_l, \omega_l \oplus g^\omega_l) \) is a reward function of a state-intent-subgoal pair. We define the reward function \( r^\text{meta}_l(s_l, \omega_l \oplus g^\omega_l) \) to be equal to the expectation of the reward of the compound MDP over actions of \( \Pi_l \) given same states, intents and subgoals (Definition 1).

The discount factor \( \gamma \) is the same.

**C. Model Construction**

In this section, we describe how the compound policy \( \Pi_l \) is constructed, how the data flows, and how the model calculates the primitive actions. Since the vast portion of the descriptions is heavily dependent on what’s stated in the previous section, we mark those with the superscript \((*)\) to notify readers.
Fig. 2: Sequential data flow through the structure of HPC. (a) Compound state through the meta-policy, (b) The state distributed to each primitive via sieving layer, (c) Intents and primitive actions calculation.

The compound policy is composed of two key structures: the meta-policy and primitives. First, the meta-policy receives a state $s_l$ from the space identical to that of the compound MDP\(^{(s)}\). Note that the state includes $g_l$ from the environment. Once given the state, it produces intents $\omega_l$ and subgoals $g_{m_l}$. Although $\omega_l$ are given to all primitives, $g_{m_l}$ are provided only for the primitives which require one \((\ast)\) (Fig. 2a). Subgoals are seen as a directing state, e.g., it can be a goal pose of the reaching primitive, a position of a certain object to be grasped, or a target velocity of a mobile robot to be achieved. As long as the primitives require these targets as a state, the subgoal can replace these features. Simultaneously, the same state $s_l$ 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 \((\ast)\), this layer sieves and distributes the given state to the primitives accordingly, denoted as $s_{li}$. The distributed state $s_{li}$ is then modified with the given subgoal $g_{mi}^{s}$ to form a subgoal-aware state. The dimension of the state which indicates the target of the primitive is replaced with the subgoal (Fig. 2b). Given the modified states for each primitive, we calculate the actions from the primitives (Fig. 2c). These actions are then combined with the intents to form an overall action for the compound policy.

D. Action Composition

Each primitive $\pi_{li}(a_{li}|s_{li})$ is regarded as a Gaussian distribution $\mathcal{N}(\mu_{li}(s_{li}), \Sigma_{li}(s_{li}))$, where $\mu_{li}(s_{li}) \in \mathbb{R}^{|A_{li}|}$ and $\Sigma_{li}(s_{li}) = \text{diag}([(\sigma_{li,j,i}^2(s_{li}))_{j=1}^{|A_{li}|}]$ denote a mean vector and a diagonal covariance matrix given the state $s_{li}$, respectively. We formulate the compound policy at level $l$, $\Pi_l$, to be the multiplication of primitives powered by their respective normalized intents. It is worth noting that the action dimension of each primitive can differ, as the types f actions required to solve different tasks are not necessarily identical. Hence, we weigh the mean of each action dimension with the summation of the intents of primitives to which they activate on. The probability distribution of the $j$-th action parameter of $\Pi_l$, $\Pi_{l,j}$ can be formalized as follows.

$$
\Pi_{l,j}(a_{lj}|s_l) = \frac{1}{Z(s_l)} \prod_{i=1}^{P} \mathcal{N}_j\left(\mu_{li,j}(s_{li}), \sigma_{li,j}^2(s_{li})\right)^{\omega_i}
$$

where $C_j$ denotes a set of primitive indices in which includes the $j$-th parameter within their action space, $P$ the total number of primitives of task $l$, and $\omega_i = \omega_i(s_l)/\sum_{p=1}^{P} \omega_p(s_l)$ the normalized intent over the sum. Since the multiplication of Gaussian distributions powered by some factor yields another
Gaussian distribution \( \omega \), we can derive the component-wise parameters for the composed Gaussian distribution as follows.

\[
\begin{align*}
\mu_{i,j} &= \frac{\sum_{c \in C} \omega_c \sum_{p=1}^{P} \omega_p}{\sum_{p=1}^{P} \omega_p/\sigma_{l,p,j}^2} \sum_{i=1}^{P} \omega_i \mu_{l,i,j}, \\
\sigma_{l,j}^2 &= \frac{\sum_{c \in C} \omega_c \sum_{p=1}^{P} \omega_p}{\sum_{p=1}^{P} \omega_p/\sigma_{l,p,j}^2} \sum_{i=1}^{P} \omega_i \sigma_{l,i,j}^2
\end{align*}
\]

where \( \mu_{i,j}, \sigma_{l,j}^2, \omega_c, \omega_p, \omega_i \) are functions of \( s_t \) and \( \mu_{l,i,j}, \sigma_{l,i,j}, \omega_{l,p,j} \) are functions of \( s_{l,t} \). Intuitively, the mean of the compound policy indicates the sum of mean of each primitive factored by their intents over variance, and re-weighted by the summed intents which are relevant to the respective action dimension. The compound variance is the reciprocal of the summation of intents over variances and again, re-weighted by the relevant intents.

IV. TRAINING META-POLICY WITH THE MAXIMUM ENTROPY OBJECTIVE

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. Next, we devise the learning algorithm of the meta-MDP by defining the state and state-action value functions of the meta-policy, by addressing the policy evaluation and the policy improvement.

A. Optimal Policy Equivalence

The objective of the maximum entropy reinforcement learning corresponds to the accumulated sum of discounted rewards and entropies as follows \([42]\).

\[
J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}(s_t,a_t) \sim P \left[ \sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{s_k \sim P,a_k \sim \pi}[r(s_k, a_k)] + \alpha H(\pi(.|s_k))|s_t, a_t] \right]
\]

where \( H(X) \) indicates the differential entropy of the continuous random variable \( X \) with the probability distribution \( P(X) \), defined as: \( H(X) = -\mathbb{E} \log(P(X)) \). To formulate the objective of our MDP, we substitute the standard policy \( \pi \) with the compound policy, \( \Pi \). In addition, since the entropy term is used for incentivizing the uncertainty of the policy distribution, we replace the standard entropy of the policy with the entropy of the primitive intents. We omit the subscript \( l \) hereafter for simplicity and rewrite the objective as follows.

\[
J(\Pi) = \sum_{t=0}^{\infty} \mathbb{E}(s_t,a_t) \sim P \left[ \sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{s_k \sim P,a_k \sim \Pi}[r(s_k, a_k)] + \alpha H(\omega_k)|s_t, a_t] \right]
\]

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

Prior to express \( J(\Pi) \) 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 \( s \in S \), intent \( \omega = \pi^{meta}(s) \in \Omega \), and the compound policy defined by the intents \( \Pi(\omega) \), the reward function of the meta-MDP is defined by

\[
r^{meta}(s, \omega) := \mathbb{E}_{a \sim \Pi(\omega)}[r(s, a)|s] \]

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

\[
J(\pi^{meta}) = \sum_{t=0}^{\infty} \mathbb{E}(s_t) \sim P \left[ \sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{s_k \sim P}(r^{meta}(s_k, \omega_k)] + \alpha H(\omega_k)|s_t] \right]
\]

where, \( \omega_k = \pi^{meta}(s_k) \)

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.

\[
J(\pi^{meta}) = J(\Pi) = J(\pi^{*})
\]

**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 \( \Pi^{*} \) such that

\[
J(\Pi^{*}) = J(\pi^{*})
\]

where \( \pi^{*} \) 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, \( \pi^{*} \).

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 \( \pi^{meta*} \) which optimizes the objective \( 9 \),

\[
\pi^{meta*} = \pi^{*}
\]

**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.

\[
J(\pi^{meta*}) = J(\Pi^{*}) = J(\pi^{*})
\]

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

B. Meta Policy Iteration via Soft Actor-Critic

We describe the meta policy iteration via alternations of iterative policy evaluations and improvements using a meta soft-action-value (soft-Q) function and a meta soft-value (soft-V) function [43]. In order to optimize (6), we first define a meta soft-Q function as follows.

Definition 3 (Meta soft-Q function). The meta soft-Q function is defined by

\[
Q^{\text{meta}}(s_t, \omega_t) := r^{\text{meta}}(s_t, \omega_t) + E_{\tau \sim p}[\sum_{k=1}^{\infty} \gamma^k(r^{\text{meta}}(s_{t+k}, \omega_{t+k}) + \alpha \mathcal{H}(\omega_{t+k})] (11)
\]

where \(\tau = (s_{t+1}, \omega_{t+1}, s_{t+2}, \omega_{t+2}, \cdots)\) denotes the trajectory originating from \((s_{t+1}, \omega_{t+1})\) under the state transition probability \(p\).

The meta soft-Q represents the state-intent value which rates how good the intent is given the state under the current meta-policy, analogous to the state-action value of a standard policy.

Next, we can derive the meta soft-V function from the Definition 3 since the value function becomes the expectation of the Q value over possible actions from the policy distribution. However, the meta-policy outputs intent in a deterministic fashion, and thus the meta soft-V function is newly defined as follows.

Definition 4 (Meta soft-V Function). The meta soft-value function is defined by

\[
V^{\text{meta}}(s_t) := Q^{\text{meta}}(s_t, \omega_t) + \alpha \mathcal{H}(\omega_t) (12)
\]

We omit the superscript meta of Q, V, and \(r\) hereafter and regard them as meta functions whenever receive intents \(\omega\) instead of actions \(a\) as an input. From Definitions 3 and 4, we can express the meta soft-Q function with the meta soft-V function as follows.

\[
Q(s_t, \omega_t) = r(s_t, \omega_t) + \gamma E_{s_{t+1} \sim p}[Q(s_{t+1}, \omega_{t+1}) + \alpha \mathcal{H}(\omega_{t+1})] (13)
\]

By inserting (12) into (13), the soft Bellman equation of the meta soft-Q function is derived by:

\[
Q(s_t, \omega_t) = r(s_t, \omega_t) + \gamma E_{s_{t+1} \sim p}[Q(s_{t+1}, \omega_{t+1}) + \alpha \mathcal{H}(\omega_{t+1})] (14)
\]

For a fixed policy, we can obtain the soft-Q function by repeatedly applying (14). However, the empirical results have shown that using both Q and V stabilizes the learning in practice [11]. Furthermore, it is possible to obtain the soft-Q function for any policy from repeated updates using (11), which leads to the policy evaluation [42].

In this work, we use neural networks as a function approximator to parameterize the soft-Q and the soft-V with \(\theta\) and \(\psi\) and denote them \(Q_{\theta}\) and \(V_{\psi}\), respectively. We train our \(Q_{\theta}\) and \(V_{\psi}\) by reducing the soft Bellman residual given below [42].

\[
J_{Q}(\theta) = E_{(s_t, \omega_t) \sim \mathcal{D}} \left[ \frac{1}{2} (Q_{\theta}(s_t, \omega_t) - (r(s_t, \omega_t) + \gamma E_{s_{t+1} \sim p}[V_{\psi}(s_{t+1})]))^2 \right] (15)
\]

\[
J_{V}(\psi) = E_{(s_t, \omega_t) \sim \mathcal{D}} \left[ \frac{1}{2} (V_{\psi}(s_t) - (Q_{\theta}(s_t, \omega_t) + \alpha \mathcal{H}(\omega_t)))^2 \right] (16)
\]

where \(\mathcal{D}\) denotes the replay buffer containing previously sampled transitions. Instead of collecting transitions of a state-action pair during rollouts, we store state-intent pairs to calculate the gradient estimators of the meta functions. The gradients of (15) and (16) can be estimated as follows.

\[
\hat{\nabla}_{\theta} J_{Q}(\theta) = \nabla_{\theta} Q_{\theta}(s_t, \omega_t)(Q_{\theta}(s_t, \omega_t) - (r(s_t, \omega_t) - \gamma V_{\psi}(s_{t+1}))) (17)
\]

\[
\hat{\nabla}_{\psi} J_{V}(\psi) = \nabla_{\psi} V_{\psi}(s_t)(V_{\psi}(s_t) - (Q_{\theta}(s_t, \omega_t) + \alpha \mathcal{H}(\omega_t))) (18)
\]

Finally, we again use neural networks to approximate the meta-policy with parameters \(\phi\), and denote it \(\pi_{\phi}^{\text{meta}}\). Please note here that the superscript meta is not dropped to emphasize that the policy we are trying to optimize is the meta-policy. The objective function of the parameters of the meta-policy can be derived with the definition of meta soft-Q, using (11).

\[
J_{\pi^{\text{meta}}} (\phi) = -E_{(s_t, \omega_t) \sim \mathcal{D}}[Q_{\theta}(s_t, \omega_t) + \alpha \mathcal{H}(\omega_t)] (19)
\]

We directly optimize the objective by taking the gradients of (19) with respect to the parameters of the meta-policy as follows.

\[
\hat{\nabla}_{\phi} J_{\pi^{\text{meta}}} (\phi) = - (\nabla_{\omega_t} Q_{\theta}(s_t, \omega_t) + \nabla_{\omega_t} \alpha \mathcal{H}(\omega_t)) \nabla_{\phi} \pi_{\phi}^{\text{meta}}(\omega_t) (20)
\]

We also leverage on the target networks \(\tilde{\psi}\) to stabilize training, and the double Q trick as in the original Soft-Acitor-Critic (SAC) paper to mitigate the overestimation of the state-action value in the policy improvement step [44–46]. The overall algorithm is described in Algorithm 1 from Appendix B.

V. EXPERIMENT

In this section, we show the proof of concept of HPC by solving the well-known robotics problem, a pick-and-place. The pick-and-place task is one of the fundamental skills required in both industrial and household applications since the vast majority of robotic tasks incorporate object handling. The pick-and-place can be considered elementary in some cases where a robot only needs to reach a designated position and opens/closes its gripper in a binary fashion to complete the task. However, the task becomes trickier if the robot needs to behave as we humans do, as it is regarded as an idiosyncrasy.
if we abruptly pause and resume our movements between gaits. HPC alleviates this problem by eliminating the need for sequential decision-making of gait selection, and directly train the policy which smoothly interchanges between these gaits. We also show the effectiveness of HPC from the reward comparison with other algorithms as a benchmark from this experiment.

A. Experiment Setup

We conduct our experiment within the MuJoCo physics simulator [47]. For the manipulator, the 6-DoF Jaco2 model from Kinova Robotics is used. To control the end-effector, we used operational space controller (OSC) within the abr_control package provided from Applied Brain Research [48], [49]. OSC receives the target pose of the end-effector and calculates required torques to be applied to each joint of the manipulator.

We decompose the pick-and-place task with a picking gait and a placing gait. These gaits can once again be decomposed into simpler primitives; reaching and grasping for the picking, reaching, and releasing for the placing. Figure 3 schematizes the overall formulation of the task. The training is sequentially done from the lowest level of primitives and iteratively stacking up the learned policies as the level increases. The state and action formulation of the overall environment along with the goal and key features of each task are described in Supplementary Material [1].

B. Training and Results

In this section, we describe the features associated with training and important aspects on the results of each task. The details on the hyperparameter settings of each task are in Supplementary Material [11].

1) Level 1 Tasks: All of the tasks at level 1 are mapped to the base policies. Therefore, the tasks do not depend on the surrogate MDP and regarded as a conventional reinforcement learning problem. With the state/action and reward formulation described in the previous sections, we use SAC to train the policies for level 1 tasks.

Different from reaching and releasing, the grasping task failed to learn from scratch. While others obtain more rewards as they simply produce actions of opposite signs from the relative pose to the target, the complexity of manipulations for the grasping an object is much higher. If the end-effector only tries to get close to the object as in reaching and releasing, the outer part of the gripper finger will most likely contact the object first and pushes it away. In order to grasp the object, the end-effector needs to head itself towards the object prior to approaching it. The reward function introduced in [11] tries to incentivize such motion by employing a term inversely proportional to $||q_{EE} - q_{o}||_2$. Still, the trained policy ends up in the sub-optimality where the end-effector hovers above the object without letting the outer side of the finger touching it. To tackle this problem, we let a human expert manually conducts the grasping task and collect trajectories of the state and action pairs. From the collected trajectories, we then warmstart the soft-Q function to propose a good initial gradient of the actor policy. We address the details on the soft-Q warmstart in Supplementary Material [IV].

Figures 4a, 4b, and 4c illustrates the training curves of level 1 tasks. Both reaching and releasing tasks successfully converge to the optimal policy without any modifications added to the vanilla SAC. Return plot of the grasping shows that the learning without the soft-Q warmstart falls into the local optima, whereas the learning with the warmstart manages to encounter successful episodes from the early stages and converges to the optimal policy.

2) Level 2 Tasks: Tasks of level 2 and greater forms a hierarchical structure, and can employ the hierarchical reinforcement learning. To examine the effects of HRL when solving the complex task, we compare HPC to the vanilla SAC denoted as Flat [11]. To evaluate the effects of pretrained primitives in terms of the performance, we compare HPC to Hierarchical Reinforcement learning with Off-policy correction (HIRO) [30]. Figures 4d and 4e shows the training curves of two level 2 tasks. The results show that HPC outperforms the baselines by far, while both Flat and HIRO do not experience a successful episode.

We further examine the interpretability of HPC from...
Fig. 4: Training curves of each tasks. The x-axis shows the number of training steps and y-axis represents the episode return. (a-c) Training curve of level 1 tasks: reaching, releasing, and grasping with and without Q-warmstart, respectively. (d-e) Training curve of level 2 tasks: picking and placing, respectively. (f) Training curve of the pick and place task.

Figs. 5a and 5b. Both figures illustrate the intents distributed to each primitives at the test time. For the picking task (Fig. 5a), the reaching primitive becomes dominant at first as it needs to approach the target object (1). Once the end-effector arrives near the initial state distribution of the grasping primitive, the meta-policy slowly reduces the intent given to the reaching and increases the usage of the grasping primitive (2, 3). It is worth emphasizing that the intent is not binarily biased to either one of the primitives, but rather to take advantages of both. The grasping primitive becomes dominant when the end-effector is fully within its initial state boundary and proceeds until the policy achieves the goal (4 - 6). For the placing task (Fig. 5b), the trained meta-policy shows different behavior from what is expected. It was presumed that the reaching primitive would be employed first to drive the end-effector near the destination point, where the initial state distribution of the releasing primitive lies. However, the meta-policy has learned through out the training steps that the releasing primitive can fully conduct the placing task without the needs of the reaching primitive (1 - 6). The result shows that the meta-policy is capable of deciding the intents effectively, regardless of the skill set.

3) Level 3 Task: The training curve of the pick and place task is shown in Fig. 4f. The figure shows that HPC outperforms both Flat and HIRO. Since both algorithms did not manage to solve the lower level tasks, it is obvious that the level 3 pick and place task is also impossible to be solved. HPC, however, abstracted its scope from the {pose, gripper angle} selection to the intent selection problem, and efficiently managed to solve the task. The intents distributed to each primitive is shown in the top figure of the Fig. 6. The intents of the level 2 and level 1 tasks are plotted in the upper and the lower part of the intent plot, respectively. At 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, 3). This addresses that the meta-policy takes advantage of the primitives within the skill set regardless of their role. Once the end-effector is properly oriented, the picking primitive is fully used until the end-effector successfully grasp the object and lift it above the table (4 - 9). With the object in the gripper, the meta-policy now leverages on the placing primitive, and completes the overall task (10 - 12).

VI. DISCUSSION AND CONCLUSION

In this paper, a concept of hierarchical primitive composition is proposed, where each of the known predefined/pretrained primitives are assembled in hierarchy to construct a compound overall policy. HPC greatly reduces the learning time of the complicated robotic tasks by abstracting the scope of the action leveraging on the surrogate MDP. It can represent the intention of the agent by intents and subgoals distributed to each of the primitives in a human-interpretable manner, while reusing a pretrained primitives enhance sample efficiency and guarantees stability at 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. Lastly, [50], [51] proposes that the human brain functional networks have a hierarchical modular organization, where medial occipital, lateral occipital, central, parieto-frontal, and fronto-temporal region of the brain takes each of the intrinsic role. HPC adapts the proposed formulation of the human brain in terms of the hierarchical structure, and thus, other
Fig. 5: (a) Intent examination of the picking task. **Top:** Weight plot of the reaching and grasping task. **Bottom:** The visualization of the simulation at the corresponding timestep. (b) Intent examination of the placing task. **Top:** Weight plot of the reaching and releasing task. **Bottom:** The visualization of the simulation at the corresponding timestep. Videos of the picking and placing tasks can be found in [https://youtu.be/WlrEhIesWJU](https://youtu.be/WlrEhIesWJU) and [https://youtu.be/kiTZreZ2WQc](https://youtu.be/kiTZreZ2WQc) respectively.
Fig. 6: Intent examination of the pick and place task. **Top:** Weight 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](https://youtu.be/_MlKP80sVtE).

works on recognition and logic using neural networks can be fused into HPC and further structuralize the total artificial intelligence as a whole.

Although HPC proposes the interpretability of the compound policy and reduces the learning time from temporal abstractions, there still remain future works to do. Firstly, Assumption 1 limits the applicability 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 space, the compound policy should bias its intent signal on a single primitive activated at the moment to outputs a proper action. This phenomenon can be seen from the below experiment. With the priorly trained primitive for the humanoid robot walking and running as shown in Fig. 7, we...
The running primitive.

Fig. 7: The walking and running primitives of the humanoid robot.

Fig. 8: The state space of the walking and running primitives. The region of the state space required for the jogging is between that of the walking and the running primitive.

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 space of the joint velocity differed. Moreover, the region of the velocity space required for the jogging is between the two primitives as in Fig. 8 the meta-policy was not able to seek for the optimal action. Therefore, additional work is required to properly match the operational 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.

\[ J(\pi^{\text{meta}}) = J(\Pi) \]  

Proof. We first rewrite the compound objective as follows.

By the linearity of the expectation, (21) becomes

\[
\sum_{t=0}^{\infty} \mathbb{E}_{(s_t, a_t) \sim \rho_1} \left[ \sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{a_k \sim \mathcal{P}, a_k \sim \Pi} [r(s_k, a_k)|s_t, a_t] \right.
\] 

\[ + \gamma^{k-t} \mathbb{E}_{a_k \sim \mathcal{P}} [\alpha \mathcal{H}(\omega_k)|s_t] \]  

\[ \left. + \mathbb{E}_{a_k \sim \mathcal{P}} [\alpha \mathcal{H}(\omega_k)|s_t] \right] \)  

Note here that the entropy of a categorical distribution of the intents \( \mathcal{H}(\omega_k) \) is independent of the actions, and thus the expectation over the action can be omitted. From Definition 1 (22) now becomes

\[
\sum_{t=0}^{\infty} \mathbb{E}_{(s_t, a_t) \sim \rho_1} \left[ \sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{a_k \sim \mathcal{P}} [r^\text{meta}(s_k, \omega_k)|s_t] \right.
\] 

\[ + \gamma^{k-t} \mathbb{E}_{a_k \sim \mathcal{P}} [\alpha \mathcal{H}(\omega_k)|s_t] \] 

\[ = \sum_{t=0}^{\infty} \mathbb{E}_{(s_t, a_t) \sim \rho_1} \left[ \sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{a_k \sim \mathcal{P}} [r^\text{meta}(s_k, \omega_k) \mathcal{H}(\omega_k)|s_t] \left. \right] \)  

\[ = J(\pi^{\text{meta}}) \]  

\[ \square \]

APPENDIX B

META POLICY ITERATION

Algorithm 1 Meta Policy Iteration

Initialize parameters \( \theta, \psi, \tilde{\psi}, \) and \( \phi \).

for each iteration do

\( s_0 \sim \rho_1 \)

for each environment step do

\( \omega_t = \pi^{\text{meta}}(s_t) \)

\( s_{t+1} \sim \mathcal{D}^\text{meta}(s_{t+1}|s_t, \omega_t) \)

\( D \leftarrow D \bigcup \{s_t, \omega_t, r(s_t, \omega_t), s_{t+1}\} \)

end for

for each gradient step do

\( \psi \leftarrow \psi - \lambda_\psi \nabla \psi J(\psi) \)

\( \theta \leftarrow \theta - \lambda_\theta \nabla \theta J(\psi) \) for \( i \in \{1, 2\} \)

\( \phi \leftarrow \phi - \lambda_\phi \nabla \phi J(\text{meta}(\phi)) \)

\( \tilde{\psi} \leftarrow \tilde{\psi} + (1 - \tau) \tilde{\psi} \)

end for

end for
Jeong-Hoon Lee received the B.S. and the M.S. degrees in mechanical engineering from Yonsei University, Seoul, South Korea in 2019 and 2021, respectively. His research interests include robotics, hierarchical reinforcement learning, and interpretable AI, with applications on manipulators and mobile robots.

Jongeun Choi received the B.S. degree in mechanical design and production engineering from Yonsei University, Seoul, South Korea, in 1998, and the M.S. and Ph.D. degrees in mechanical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 2002 and 2006, respectively. He is currently the Director with the Machine Learning and Control Systems Laboratory and a Professor with the School of Mechanical Engineering, Yonsei University. From 2020 to 2021, he has served as the Graduate Program Chair with School of Mechanical Engineering, Yonsei University. Since 2019, he has been the Chairperson with the Department of Vehicle Convergence Engineering, Yonsei University, funded by Hyundai Motor Company. Prior to joining Yonsei University, he worked for ten years as an Associate Professor from 2012 to 2016 and an Assistant Professor from 2006 to 2012, with the Department of Mechanical Engineering, and also with the Department of Electrical and Computer Engineering, Michigan State University. His current research interests include machine learning, reinforcement learning, systems and control, system identification, and Bayesian methods, with applications to autonomous robots, self-driving vehicles, mobile sensor networks, (physical) human and robot interaction, and medical machine learning. Dr. Choi co-organized and co-edited special issues on Stochastic Models, Control, and Algorithms in Robotics in Journal of Dynamic Systems, Measurement and Control (JDSMC), from 2014 to 2015. He was the recipient of the Best Conference Paper Award at the 12th International Conference on Ubiquitous Robots and Ambient Intelligence, in 2015, and an NSF CAREER Award, in 2009. His articles were finalists for the Best Student Paper Award at the 24th American Control Conference, in 2005, and the Dynamic System and Control Conference, in 2011 and 2012. He served as an Associate Editor for the IEEE ROBOTICS AND AUTOMATION LETTERS, JDSMC, and International Journal of Precision Engineering and Manufacturing. He serves as a Senior Editor for Ubiquitous Robots, in 2020. He is a Member of ASME and IEEE.