MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies

Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, Sergey Levine
Department of Electrical Engineering and Computer Science
University of California, Berkeley
{xbpeng, mbchang, grace.zhang}@berkeley.edu
pabbeel@cs.berkeley.edu
svlevine@eecs.berkeley.edu

Abstract

Humans are able to perform a myriad of sophisticated tasks by drawing upon skills acquired through prior experience. For autonomous agents to have this capability, they must be able to extract reusable skills from past experience that can be recombined in new ways for subsequent tasks. Furthermore, when controlling complex high-dimensional morphologies, such as humanoid bodies, tasks often require coordination of multiple skills simultaneously. Learning discrete primitives for every combination of skills quickly becomes prohibitive. Composable primitives that can be recombined to create a large variety of behaviors can be more suitable for modeling this combinatorial explosion. In this work, we propose multiplicative compositional policies (MCP), a method for learning reusable motor skills that can be composed to produce a range of complex behaviors. Our method factorizes an agent's skills into a collection of primitives, where multiple primitives can be activated simultaneously via multiplicative composition. This flexibility allows the primitives to be transferred and recombined to elicit new behaviors as necessary for novel tasks. We demonstrate that MCP is able to extract composable skills for highly complex simulated characters from pre-training tasks, such as motion imitation, and then reuse these skills to solve challenging continuous control tasks, such as dribbling a soccer ball to a goal, and picking up an object and transporting it to a target location. (Video[1])

1 Introduction

Reinforcement learning is commonly applied to solve tasks from scratch. While *tabula rasa* learning can achieve state-of-the-art performance on a broad range of tasks [4, 13, 26, 28, 36], this approach can incur significant drawbacks in terms of sample efficiency and limits the complexity of skills that an agent can acquire. The ability to transfer and re-purpose skills learned from prior experiences to new domains is a hallmark of intelligent agents. Transferable skills can enable agents to solve tasks that would otherwise be prohibitively challenging to learn from scratch, by leveraging prior experiences to provide structured exploration and more effective representations. However, learning versatile and reusable skills that can be applied to a diverse set of tasks remains a challenging problem, particularly when controlling systems with large numbers of degrees-of-freedom.

In this work, we propose multiplicative compositional policies (MCP), a method for learning reusable motor primitives that can be composed to produce a continuous spectrum of skills. Once learned, the primitives can be transferred to new tasks and combined to yield different behaviors as necessary in the target domain. Standard hierarchical models [10, 38] often activate only a single primitive at each timestep, which can limit the diversity of behaviors that can be produced by the agent. MCP

---

[1] Supplementary video: xbpeng.github.io/projects/MCP/

composes primitives through a multiplicative model that enables multiple primitives to be activated at a given timestep, thereby providing the agent a more flexible range of skills. Our method can therefore be viewed as providing a means of composing skills in space, while standard hierarchical models compose skills in time by temporally sequencing the set of available skills. MCP can also be interpreted as a variant of latent space models, where the latent encoding specifies a particular composition of a discrete set of primitives.

The primary contribution of our work is a method for learning and composing transferable skills using multiplicative compositional policies. By pre-training the primitives to imitate a corpus of different motion clips, our method learns a set of primitives that can be composed to produce a flexible range of behaviors. While conceptually simple, MCP is able to solve a suite of challenging mobile manipulation tasks with complex simulated characters, significantly outperforming prior methods as task complexity grows. Our analysis shows that the primitives discover specializations that are reminiscent of previous manually-designed control structures, and produce coherent exploration strategies that are vital for high-dimensional long-horizon tasks. In our experiments, MCP substantially outperforms prior methods for skill transfer, with our method being the only approach that learns a successful policy on the most challenging task in our benchmark.

2 Preliminaries

We consider a multi-task RL framework for transfer learning, consisting of a set of pre-training tasks and transfer tasks. An agent is trained from scratch on the pre-training tasks, but it may then apply any skills learned during pre-training to the subsequent transfer tasks. The objective then is to leverage the pre-training tasks to acquire a set of reusable skills that enables the agent to be more effective at the later transfer tasks. Each task is represented by a state space $s_t \in \mathcal{S}$, an action space $a_t \in \mathcal{A}$, a dynamics model $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$, a goal space $g \in \mathcal{G}$, a goals distribution $g \sim p(g)$, and a reward function $r_t = r(s_t, a_t, g)$. The goal specifies task specific features, such as a motion clip to imitate, or the target location an object should be placed. All tasks share a common state space, action space, and dynamics model. However, the goal space, goal distribution, and reward function may differ between pre-training and transfer tasks. For each task, the agent's objective is to learn an optimal policy π^* that maximizes its expected return $J(\pi) = \mathbb{E}_{g \sim p(g), \tau \sim p_\pi(\tau|g)} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$ over the distribution of goals from the task, where $p_\pi(\tau|g) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t, g)$ denotes the distribution over trajectories τ induced by the policy π for a given goal g. T represents the time horizon, and $\gamma \in [0, 1]$ is the discount factor. Successful transfer cannot be expected for unrelated tasks. Therefore, we consider the setting where the pre-training tasks encourage the agent to learn relevant skills for the subsequent transfer tasks, but may not necessarily cover the full range of skills required to be effective at the transfer tasks.

Hierarchical policies are a common model for reusing and composing previously learned skills. One approach for constructing a hierarchical policy is by using a mixture-of-experts model [15, 19, 27, 29, 39], where the composite policy's action distribution $\pi(a|s, g)$ is represented by a weighted sum of distributions from a set of primitives $\pi_i(a|s, g)$ (i.e. low-level policies). A gating function determines the weights $w_i(s, g)$ that specify the probability of activating each primitive for a given s and g,

$$\pi(a|s, g) = \sum_{i=1}^{k} w_i(s, g)\pi_i(a|s, g), \quad \sum_{i=1}^{k} w_i(s, g) = 1, \quad w_i(s, g) \geq 0. \tag{1}$$

Here, k denotes the number of primitives. We will refer to this method of composing primitives as an *additive model*. To sample from the composite policy, a primitive π_i is first selected according to w, then an action is sampled from the primitive's distribution. Therefore, a limitation of the additive model is that only one primitive can be active at a particular timestep. While complex behaviors can be produced by sequencing the various primitives in time, the action taken at each timestep remains restricted to the behavior prescribed by a single primitive. Selecting from a discrete set of primitive skills can be effective for simple systems with a small number of actuated degrees-of-freedom, where an agent is only required to perform a small number of subtasks at the same time. But as the complexity of the system grows, an agent might need to perform more and more subtasks *simultaneously*. For example, a person can walk, speak, and carry an object all at the same time. Furthermore, these subtasks can be combined in any number of ways to produce a staggering array of diverse behaviors. This combinatorial explosion can be prohibitively challenging to model with policies that activate only one primitive at a time.

3 Multiplicative Compositional Policies

In this work, we propose multiplicative compositional policies (MCP), a method for composing primitives that addresses this combinatorial explosion by explicitly factoring the agent's behavior – not with respect to time, but with respect to the action space. Our model enables the agent to activate multiple primitives simultaneously, with each primitive specializing in different behaviors that can be composed to produce a continuous spectrum of skills. Our probabilistic formulation accomplishes this by treating each primitive as a distribution over actions, and the composite policy is obtained by a multiplicative composition of these distributions,

$$\pi(a|s,g) = \frac{1}{Z(s,g)} \prod_{i=1}^{k} \pi_i(a|s,g)^{w_i(s,g)}, \quad w_i(s,g) \geq 0. \tag{2}$$

Unlike an additive model, which activates only a single primitive per timestep, the *multiplicative model* allows multiple primitives to be activated simultaneously. The gating function specifies the weights $w_i(s,g)$ that determine the influence of each primitive on the composite action distribution, with a larger weight corresponding to a larger influence. The weights need not be normalized, but in the following experiments, the weights will be bounded $w_i(s,g) \in [0,1]$. $Z(s,g)$ is the partition function that ensures the composite distribution is normalized. While the additive model directly samples actions from the selected primitive's distribution, the multiplicative model first combines the primitives, and then samples actions from the resulting distribution.

3.1 Gaussian Primitives

Gaussian policies are a staple for continuous control tasks, and modeling multiplicative primitives using Gaussian policies provides a particularly convenient form for the composite policy. Each primitive $\pi_i(a|s,g) = \mathcal{N}(\mu_i(s,g), \Sigma_i(s,g))$ will be modeled by a Gaussian with mean $\mu_i(s,g)$ and diagonal covariance matrix $\Sigma_i(s,g) = \mathrm{diag}\left(\sigma_i^1(s,g), \sigma_i^2(s,g), ..., \sigma_i^{|\mathcal{A}|}\right)$, where $\sigma_i^j(s,g)$ denotes the variance of the jth action parameter from primitive i, and $|\mathcal{A}|$ represents the dimensionality of the action space. A multiplicative composition of Gaussian primitives yields yet another Gaussian policy $\pi(a|s,g) = \mathcal{N}(\mu(s,g), \Sigma(s,g))$. Since the primitives model each action parameter with an independent Gaussian, the action parameters of the composite policy π will also assume the form of independent Gaussians with component-wise mean $\mu^j(s,g)$ and variance $\sigma^j(s,g)$,

$$\mu^j(s,g) = \frac{1}{\sum_{l=1}^{k} \frac{w_l(s,g)}{\sigma_l^j(s,g)}} \sum_{i=1}^{k} \frac{w_i(s,g)}{\sigma_i^j(s,g)} \mu_i^j(s,g), \qquad \sigma^j(s,g) = \left(\sum_{i=1}^{k} \frac{w_i(s,g)}{\sigma_i^j(s,g)} \right)^{-1}. \tag{3}$$

Note that while $w_i(s,g)$ determines a primitive's overall influence on the composite distribution, each primitive can also independently adjust its influence per action parameter through $\sigma_i^j(s,g)$. Once the parameters of the composite distribution have been determined, π can be treated as a regular Gaussian policy, and trained end-to-end using standard automatic differentiation tools.

3.2 Pre-Training and Transfer

The primitives are learned through a set of pre-training tasks. The same set of primitives is responsible for solving all pre-training tasks, which results in a collection of primitives that captures the range of behaviors needed for the set of tasks. Algorithm 1 illustrates the overall training process. $J_{pre}(\pi_{1:k}, w)$ denotes the objective for the pre-training tasks for a given set of primitives $\pi_{1:k}$ and gating function w, and $J_{tra}(\pi_{1:k}, \omega)$ denotes the objective for the transfer tasks. When transferring primitives to a new

---

Algorithm 1 MCP Pre-Training and Transfer

1: Pre-training:
2: $\pi_i \leftarrow$ random parameters for $i = 1, ..., k$
3: $w \leftarrow$ random parameters
4: $\pi_{1:k}^*, w^* = \arg\max_{\pi_{1:k}, w} J_{pre}(\pi_{1:k}, w)$
5: Transfer:
6: $\omega \leftarrow$ random parameters
7: $\omega^* = \arg\max_{\omega} J_{tra}(\pi_{1:k}^*, \omega)$

---

task, the parameters of the primitives are kept fixed, while a new policy is trained to specify weights for composing the primitives. Therefore, the primitives can be viewed as a set of nonlinear basis functions that defines a new action space for use in subsequent tasks. During pre-training, in order
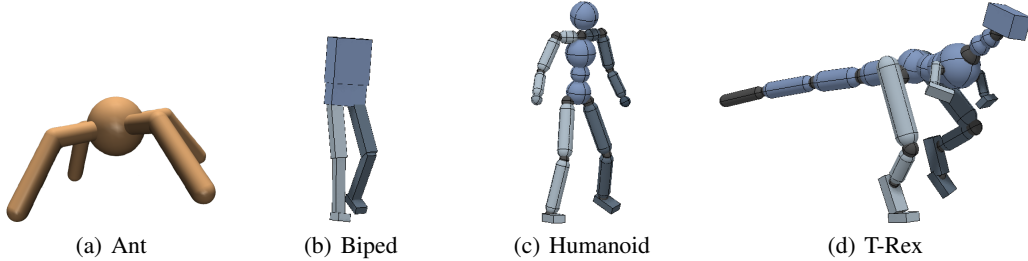
| (a) Ant | (b) Biped | (c) Humanoid | (d) T-Rex |

Figure 1: Our method is evaluated on complex 3D characters with different morphologies and large numbers of degrees-of-freedom.

to force the primitives to specialize in distinct skills, we use an asymmetric model, where only the gating function $w_i(s, g)$ observes the goal g, and the primitives have access only to the state s,

$$\pi(a|s, g) = \frac{1}{Z(s, g)} \prod_{i=1}^{k} \pi_i(a|s)^{w_i(s,g)}, \quad \pi_i(a|s) = \mathcal{N}\left(\mu_i(s), \Sigma_i(s)\right). \quad (4)$$

This asymmetric model prevents the degeneracy of a single primitive becoming responsible for all goals, and instead encourages the primitives to learn distinct skills that can then be composed by the gating function as needed for a given goal. Furthermore, since the primitives depend only on the state, they can be conveniently transferred to new tasks that share similar state spaces but may have different goal spaces. When transferring the primitives to new tasks, the parameters of the primitives $\pi_i(a|s)$ are kept fixed to prevent catastrophic forgetting, and a new gating function $\omega(w|s, g)$ is trained to specify the weights $w = (w_1, w_2, ...)$ for composing the primitives.

4   Related Work

Learning reusable representations that are transferable across multiple tasks has a long history in machine learning [1, 5, 40]. Finetuning remains a popular transfer learning technique when using neural network, where a model is first trained on a source domain, and then the learned features are reused in a target domain by finetuning via backpropagation [8, 18]. One of the drawbacks of this procedure is catastrophic forgetting, as backpropagation is prone to destroying previously learned features before the model is able to utilize them in the target domain [21, 31, 32].

Hierarchical Policies: A popular method for combining and reusing skills is by constructing hierarchical policies, where a collection of low-level controllers, which we will refer to as primitives, are integrated together with the aid of a gating function that selects a suitable primitive for a given scenario [2, 15, 38]. A common approach for building hierarchical policies is to first train a collection of primitives through a set of pre-training tasks, which encourages each primitive to specialize in distinct skills [6, 12, 23, 24, 29]. Once trained, the primitives can be integrated into a hierarchical policy and transferred to new tasks. End-to-end methods have also been proposed for training hierarchical policies [2, 7, 41]. However, since standard hierarchical policies only activate one primitive at a time, it is not as amenable for composition or interpolation of multiple primitives in order to produce new skills.

Latent Space Models: Our work falls under a class of methods that we will refer to broadly as latent space models. These methods specify controls through a latent representation that is then mapped to the controls (i.e. actions) of the underlying system [22]. Similar to hierarchical models, a latent representation can first be learned using a set of pre-training tasks, before transferring to downstream tasks [14, 17]. But unlike a standard hierarchical model, which activates a single primitive at a time, continuous latent variables can be used to enable more flexible interpolation of skills in the latent space. Various diversity-promoting pre-training techniques have been proposed for encouraging the latent space to model semantically distinct behaviors [9, 11, 16]. Demonstrations can also be incorporated during pre-training to acquire more complex skills [25]. In this work, we present a method for modeling latent skill representations as a composition of multiplicative primitives. We show that the additional structure introduced by the primitives enables our agents to tackle complex continuous control tasks, achieving competitive performance when compared to previous models, and significantly outperforming prior methods as task complexity grows.

4

(a) Heading: Humanoid

(b) Carry: Biped

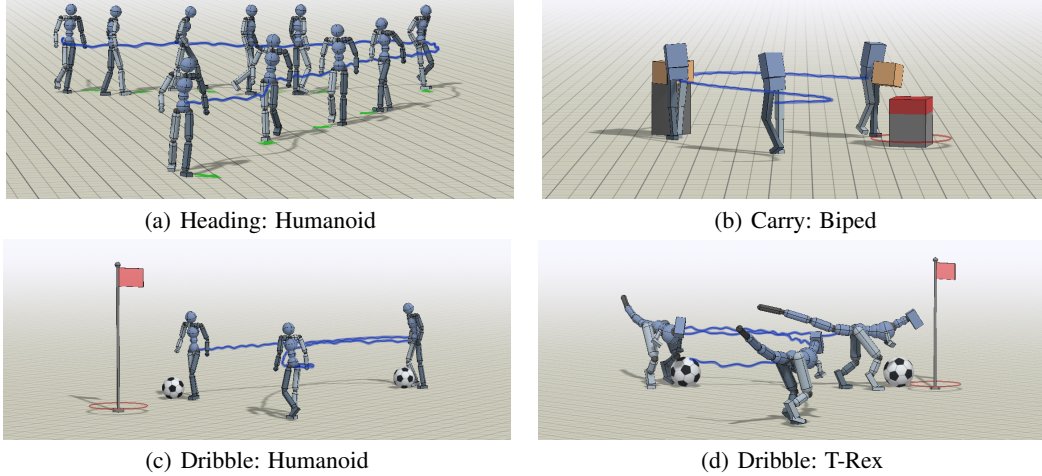(c) Dribble: Humanoid

(d) Dribble: T-Rex

Figure 2: The transfer tasks pose a challenging combination of locomotion and object manipulation, such as carrying an object to a target location and dribbling a ball to a goal, which requires coordination of multiple body parts and temporally extended behaviors.

5 Experiments

We evaluate the effectiveness of our method on controlling complex simulated characters, with large numbers of degrees-of-freedom (DoFs), to perform challenging long-horizon tasks. The tasks vary from simple locomotion tasks to difficult mobile manipulation tasks. The characters include a simple 14 DoF ant, a 23 DoF biped, a more complex 34 DoF humanoid, and a 55 DoF T-Rex. Illustrations of the characters are shown in Figure 1, and examples of transfer tasks are shown in Figure 2. Our experiments aim to study MCP's performance on complex temporally extended tasks, and examine the behaviors learned by the primitives. We also evaluate our method comparatively to determine the value of multiplicative primitives as compared to more standard additive mixture models, as well as to prior methods based on options and latent space embeddings. Behaviors learned by the policies are best seen in the supplementary video[2].

5.1 Experimental Setup

Pre-Training Tasks: The pre-training tasks in our experiments consist of motion imitation tasks, where the objective is for the character to mimic a corpus of different reference motions. Each reference motion specifies a sequence of target states $\{\hat{s}_0, \hat{s}_1, ..., \hat{s}_T\}$ that the character should track at each timestep. We use a motion imitation approach following Peng et al. [30]. But instead of training separate policies for each motion, a single policy, composed of multiple primitives, is trained to imitate a variety of motion clips. To imitate multiple motions, the goal $g_t = (\hat{s}_{t+1}, \hat{s}_{t+2})$ provides the policy with target states for the next two timesteps. A reference motion is selected randomly at the start of each episode. To encourage the primitives to learn to transition between different skills, the reference motion is also switched randomly to another motion within each episode. The corpus of motion clips is comprised of different walking and turning motions.

Transfer Tasks: We evaluate our method on a set of challenging continuous control tasks, involving locomotion and object manipulation using the various characters. Detailed descriptions of each task are available in the supplementary material.

Heading: First we consider a simple heading task, where the objective is for the character to move along a target heading direction $\hat{\theta}_t$. The goal $g_t = (\cos(\hat{\theta}_t), -\sin(\hat{\theta}_t))$ encodes the heading as a unit vector along the horizontal plane. The target heading varies randomly over the course of an episode.

Carry: To evaluate our method's performance on long horizon tasks, we consider a mobile manipulation task, where the objective is to move a box from a source location to a target location. The task can be decomposed into a sequence of subtasks, where the character must first pickup the box from the source location, before carrying it to the target location, and placing it on the table. The source and target are placed randomly each episode. Depending on the initial configuration, the task may require
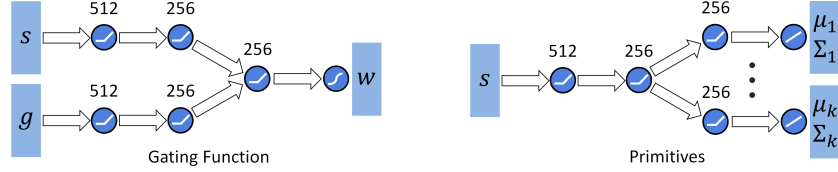
Figure 3: Schematic illustrations of the MCP architecture. The gating function receives both s and g as inputs, which are first encoded by separate networks, with 512 and 256 units. The resulting features are concatenated and processed with a layer of 256 units, followed by a sigmoid output layer to produce the weights $w(s, g)$. The primitives receive only s as input, which is first processed by a common network, with 512 and 256 units, before branching into separate layers of 256 units for each primitive, followed by a linear output layer that produces $\mu_i(s)$ and $\Sigma_i(s)$ for each primitive. ReLU activation is used for all hidden units.

Environment	Scratch	Finetune	Hierarchical	Option-Critic	MOE	Latent Space	MCP (Ours)
Heading: Biped	0.927 ± 0.032	0.970 ± 0.002	0.834 ± 0.001	0.952 ± 0.012	0.918 ± 0.002	0.970 ± 0.001	$\mathbf{0.976 \pm 0.002}$
Carry: Biped	0.027 ± 0.035	0.324 ± 0.014	0.001 ± 0.002	0.346 ± 0.011	0.013 ± 0.013	0.456 ± 0.031	$\mathbf{0.575 \pm 0.032}$
Dribble: Biped	0.072 ± 0.012	0.651 ± 0.025	0.546 ± 0.024	0.046 ± 0.008	0.073 ± 0.021	0.768 ± 0.012	$\mathbf{0.782 \pm 0.008}$
Dribble: Humanoid	0.076 ± 0.024	0.598 ± 0.030	0.198 ± 0.002	0.058 ± 0.007	0.043 ± 0.021	0.751 ± 0.006	$\mathbf{0.805 \pm 0.006}$
Dribble: T-Rex	0.065 ± 0.032	0.074 ± 0.011	$-$	0.098 ± 0.013	0.070 ± 0.017	0.115 ± 0.013	$\mathbf{0.781 \pm 0.021}$
Holdout: Ant	$\mathbf{0.951 \pm 0.093}$	0.885 ± 0.062	$-$	$-$	$-$	0.745 ± 0.060	0.812 ± 0.030

Table 1: Performance statistics of different models on transfer tasks. Additional experiments are available in the supplementary material. MCP outperforms other methods on a suite of challenging tasks with complex simulated characters.

thousands of timesteps to complete. The goal $g_t = (x_{tar}, q_{tar}, x_{src}, q_{src}, x_b, q_b, v_b, \omega_b)$ encodes the target's position x_{tar} and orientation q_{tar} represented as a quaternion, the source's position x_{src} and orientation q_{src}, and box's position x_b, orientation q_b, linear velocity v_b, and angular velocity ω_b.

Dribble: This task poses a challenging combination of locomotion and object manipulation, where the objective is to move a soccer ball to a target location. Since the policy does not have direct control over the ball, it must rely on complex contact dynamics in order to manipulate the movement of the ball while also maintaining balance. The location of the ball and target are randomly initialized each episode. The goal $g_t = (x_{tar}, x_b, q_b, v_b, \omega_b)$ encodes the target location x_{tar}, and ball's position x_b, orientation q_b, linear velocity v_b, and angular velocity ω_b.

Model Representation: All experiments use a similar network architecture for the policy, as illustrated in Figure 3. Each policy is composed of $k = 8$ primitives. The gating function and primitives are modeled by separate networks that output $w(s, g)$, $\mu_{i:k}(s)$, and $\Sigma_{i:k}(s)$, which are then composed according to Equation 2 to produce the composite policy. The state describes the configuration of the character's body, with features consisting of the relative positions of each link with respect to the root, their rotations represented by quaternions, and their linear and angular velocities. Actions from the policy specify target rotations for PD controllers positioned at each joint. Target rotations for 3D spherical joints are parameterized using exponential maps. The policies operate at 30Hz and are trained using proximal policy optimization (PPO) [34].

5.2 Comparisons

We compare MCP to a number of prior methods, including a baseline model trained from scratch for each transfer task, and a model first pre-trained to imitate a reference motion before being finetuned on the transfer tasks. To evaluate the effects of being able to activate and compose multiple primitives simultaneously, we compare MCP to models that activate only one primitive at a time, including a hierarchical model that sequences a set of pre-trained skills [23, 24], an option-critic model [2], and a mixture-of-experts model (MOE) analogous to Equation 1. Finally, we also include comparisons to a continuous latent space model with an architecture similar to Hausman et al. [16] and Merel et al. [25]. All models, except for the scratch model, are pre-trained with motion imitation [30]. Detailed descriptions of each method can be found in the supplementary material. Figure 4 illustrates learning curves for the various methods on the transfer tasks and Table 1 summarizes their performance. Each environment is denoted by "Task: Character". Performance is recorded as the average normalized return across approximately 100 episodes, with 0 being the minimum possible return per episode and 1 being the maximum. Three models initialized with different random seeds are trained for each environment and method.
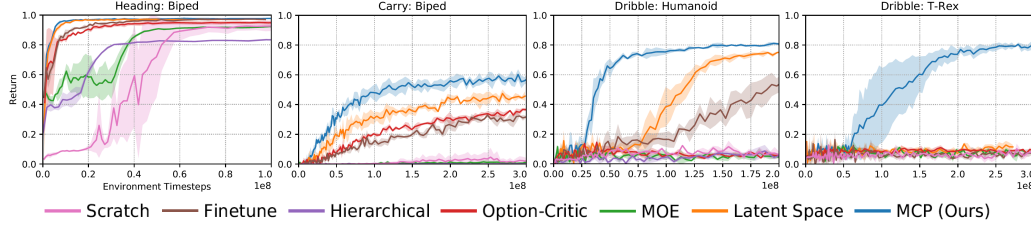
6

Figure 4: Learning curves of the various models when applied to transfer tasks. MCP substantially improves learning speed and performance on challenging tasks (e.g. carry and dribble), and is the only method that succeeds on the most difficult task (Dribble: T-Rex).
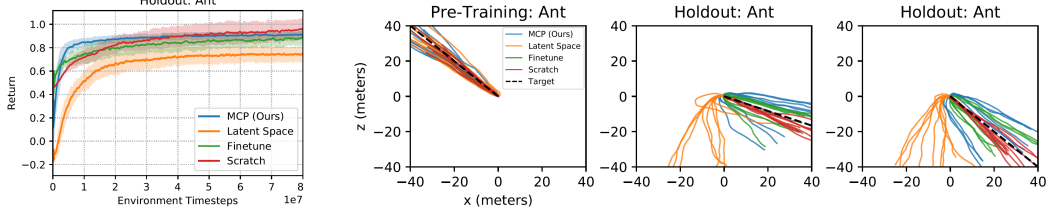

Figure 5: **Left:** Learning curves on holdout tasks in the Ant environment. **Right:** Trajectories produced by models with target directions from pre-training, and target directions from the holdout set after training on transfer tasks. The latent space model is prone to overfitting to the pre-training tasks, and can struggle to adapt to the holdout tasks.

Our experiments show that MCP performs well across the suite of tasks. For simple tasks such as heading, all models show similar performance. But as task complexity increases, MCP exhibits significant improvements to learning speed and asymptotic performance. Training from scratch is effective for the simple heading task, but is unable to solve the more challenging carry and dribble tasks. Finetuning proved to be a strong baseline, but struggles with the more complex morphologies. With higher dimensional action spaces, independent action noise is less likely to produce useful behaviors. Models that activate only a single primitive at a time, such as the hierarchical model, option-critic model, and MOE model, tend to converge to lower asymptotic performance due to their limited expressivity. MOE is analogous to MCP where only a single primitive is active at a time. Despite using a similar number of primitives as MCP, being able to activate only one primitive per timestep limits the variety of behaviors that can be produced by MOE. This suggests that the flexibility of MCP to compose multiple primitives is vital for more sophisticated tasks. The latent space model shows strong performance on most tasks. But when applied to characters with more complex morphologies, such as the humanoid and T-Rex, MCP consistently outperforms the latent space model, with MCP being the only model that solves the dribbling task with the T-Rex.

We hypothesize that the performance difference between MCP and the latent space model may be due to the process through which a latent code w is mapped to an action for the underlying system. With the latent space model, the pre-trained policy $\pi(a|s, w)$ acts as a decoder that maps w to a distribution over actions. We have observed that this decoder has a tendency to overfit to the pre-training behaviors, and can therefore limit the variety of behaviors that can be deployed on the transfer tasks. In the case of MCP, if σ_i^j is the same across all primitives, then we can roughly view w as specifying a convex combination of the primitive means $\mu_{i:k}$. Therefore, $\mu_{1:k}$ forms a convex hull in the original action space, and the transfer policy $\omega(w|s, g)$ can select any action within this set. As such, MCP may provide the transfer policy with a more flexible range of skills than the latent space model. To test this hypothesis, we evaluate the different models on transferring to out-of-distribution tasks using a simple setup. The environment is a variant of the standard Gym Ant environment [4], where the agent's objective is to run along a target direction $\hat{\theta}$. During pre-training, the policies are trained with directions $\hat{\theta} \in [0, 3/2\pi]$. During transfer, the directions are sampled from a holdout set $\hat{\theta} \in [3/2\pi, 2\pi]$. Figure 5 illustrates the learning curves on the transfer task, along with the trajectories produced by the models when commanded to follow different target directions from the pre-training and transfer tasks. Indeed we see that the latent space model is prone to overfitting to the directions from pre-training, and struggles to adapt to the holdout directions. MCP provides the transfer policy sufficient flexibility to adapt quickly to the transfer tasks. The scratch and finetune models also perform well on the transfer tasks, since they operate directly on the underlying action space.
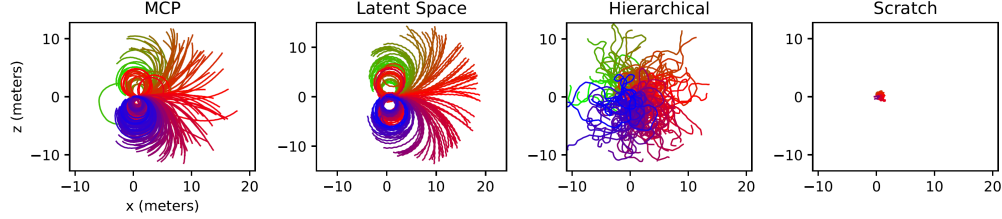
7

Figure 6: Trajectories of the humanoid's root along the horizontal plane visualizing the exploration behaviors of different models. MCP and other models that are pre-trained with motion imitation produce more structured exploration behaviors.
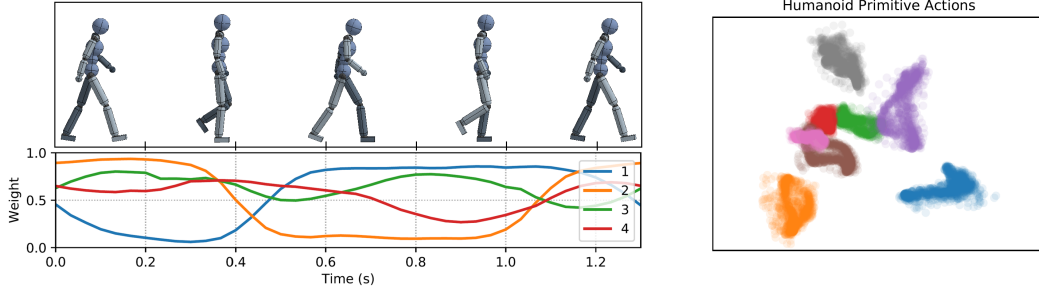

Figure 7: **Left:** Weights for primitives over the course of a walk cycle. Primitives develop distinct specializations, with some primitives becoming most active during the left stance phase, and others during right stance. **Right:** PCA embedding of actions from each primitive exhibits distinct clusters.

5.3 Exploration Behaviors

To analyze the exploration behaviors produced by the primitives, we visualize the trajectories obtained by random combinations of the primitives, where the weights are sampled from a Gaussian and held fixed over the course of a trajectory. Figure 6 illustrates the trajectories of the humanoid's root produced by various models. Similar to MCP, the trajectories from the latent space model are also produced by sampling w from a Gaussian. The trajectories from the hierarchical model are generated by randomly sequencing the set of primitives. The model trained from scratch simply applies Gaussian noise to the actions, which leads to a fall after only few timesteps. Models that are pre-trained with motion imitation produce more structured behaviors that travel in different directions.

5.4 Primitive Specializations

To analyze the specializations of the primitives, we record the weight of each primitive over the course of a walk cycle. Figure 7 illustrates the weights during pre-training, when the humanoid is trained to imitate walking motions. The activations of the primitives show a strong correlation to the phase of a walk cycle, with primitive 1 becoming most active during left stance and becoming less active during right stance, while primitive 2 exhibits precisely the opposite behavior. The primitives appear to have developed a decomposition of a walking gait that is commonly incorporated into the design of locomotion controllers [42]. Furthermore, these specializations consistently appear across multiple training runs. Next, we visualize the actions proposed by each primitive. Figure 7 shows a PCA embedding of the mean action from each primitive. The actions from each primitive form distinct clusters, which suggests that the primitives are indeed specializing in different behaviors.

6 Conclusion

We presented multiplicative compositional policies (MCP), a method for learning and composing skills using multiplicative primitives. Despite its simplicity, our method is able to learn sophisticated behaviors that can be transferred to solve challenging continuous control tasks with complex simulated agents. Once trained, the primitives form a new action space that enables more structured exploration and provides the agent with the flexibility to combine the primitives in novel ways in order to elicit new behaviors for a task. Our experiments show that MCP can be effective for long horizon tasks and outperforms prior methods as task complexity grows. While MCP provides a form of spatial abstraction, we believe that incorporating temporal abstractions is an important direction. During pre-training, some care is required to select an expressive corpus of reference motions. In future work, we wish to investigate methods for recovering sophisticated primitive skills without this supervision.

8

Acknowledgements

References

[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 41–48. MIT Press, 2007. URL `http://papers.nips.cc/paper/3143-multi-task-feature-learning.pdf`.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2016.

[3] Glen Berseth, Cheng Xie, Paul Cernek, and Michiel van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. In *Proc. International Conference on Learning Representations*, 2018.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

[5] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

[6] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Robust task-based control policies for physics-based characters. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 28(5):Article 170, 2009.

[7] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 271–278, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-274-7. URL `http://dl.acm.org/citation.cfm?id=645753.668239`.

[8] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 647–655, Bejing, China, 22–24 Jun 2014. PMLR. URL `http://proceedings.mlr.press/v32/donahue14.html`.

[9] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=SJx63jRqFm`.

[10] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 251–260, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383287. URL `http://doi.acm.org/10.1145/383259.383287`.

[11] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[12] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *CoRR*, abs/1710.09767, 2017. URL `http://arxiv.org/abs/1710.09767`.

[13] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *CoRR*, abs/1610.00633, 2016. URL `http://arxiv.org/abs/1610.00633`.

[14] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1846–1855, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18a.html`.

[15] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016. URL `http://www.cs.utexas.edu/users/ai-lab/?hausknecht:iclr16`.

[16] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rk07ZXZRb`.

[17] Nicolas Heess, Gregory Wayne, Yuval Tassa, Timothy P. Lillicrap, Martin A. Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *CoRR*, abs/1610.05182, 2016. URL `http://arxiv.org/abs/1610.05182`.

[18] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 0036-8075. doi: 10.1126/science.1127647. URL `http://science.sciencemag.org/content/313/5786/504`.

[19] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, March 1991. ISSN 0899-7667. doi: 10.1162/neco.1991.3.1.79. URL `http://dx.doi.org/10.1162/neco.1991.3.1.79`.

[20] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. URL `http://dblp.uni-trier.de/db/journals/corr/corr1312.html#KingmaW13`.

[21] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

[22] J. Zico Kolter and Andrew Y. Ng. Learning omnidirectional path following using dimensionality reduction. In *in Proceedings of Robotics: Science and Systems*, 2007.

[23] Libin Liu and Jessica Hodgins. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics*, 36(3), 2017.

[24] Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BJfYvo09Y7`.

[25] Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BJl6TjRcY7`.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

[27] Gerhard Neumann, Wolfgang Maass, and Jan Peters. Learning complex motions by sequencing simpler motion templates. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 753–760, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553471. URL `http://doi.acm.org/10.1145/1553374.1553471`.

[28] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, 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. *CoRR*, abs/1808.00177, 2018. URL `http://arxiv.org/abs/1808.00177`.

[29] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Trans. Graph.*, 35(4):81:1–81:12, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925881. URL `http://doi.acm.org/10.1145/2897824.2925881`.

[30] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201311. URL `http://doi.acm.org/10.1145/3197517.3201311`.

[31] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL `http://arxiv.org/abs/1606.04671`.

[32] Andrei A. Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 262–270. PMLR, 13–15 Nov 2017. URL `http://proceedings.mlr.press/v78/rusu17a.html`.

[33] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2016.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[35] SFU. Sfu motion capture database. http://mocap.cs.sfu.ca/.

[36] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017. URL `http://dx.doi.org/10.1038/nature24270`.

[37] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[38] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, August 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1. URL `http://dx.doi.org/10.1016/S0004-3702(99)00052-1`.

[39] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, 2016.

[40] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, pages 640–646. The MIT Press, 1996.

[41] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 3540–3549. JMLR.org, 2017. URL `http://dl.acm.org/citation.cfm?id=3305890.3306047`.

[42] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):Article 105, 2007.

Supplementary Material

7   Additional Experiments

A comprehensive set of learning curves for all transfer tasks are available in Figure 8 and Table 2 summarizes the performance of the final policies. Note that the hierarchical model selects a new primitive at the start of each walk cycle, approximately 30 environment timesteps, and as such operates at a lower frequency than the other models. Instead of recording the number of policy steps, we record the number of environment timestep. This corresponds to the amount of physical interactions that the agent requires to learn a policy, which is often the bottleneck for simulated and real world domains.

To analyze the effects of the number of primitives used, we trained MCP models with $k = 4, 8, 16, 32$ primitives. Figure 4 illustrates the learning curves with varying numbers of primitives. We do not observe a noticeable performance difference between 4 and 8 primitives. But as the number of primitives increases, learning efficiency appears to decrease. In the case of 32 primitives, the dimensionality of w is larger than the dimensionality of the original action space for the humanoid (28D), which diminishes some of the benefits of the dimensionality reduction provided by the primitives.
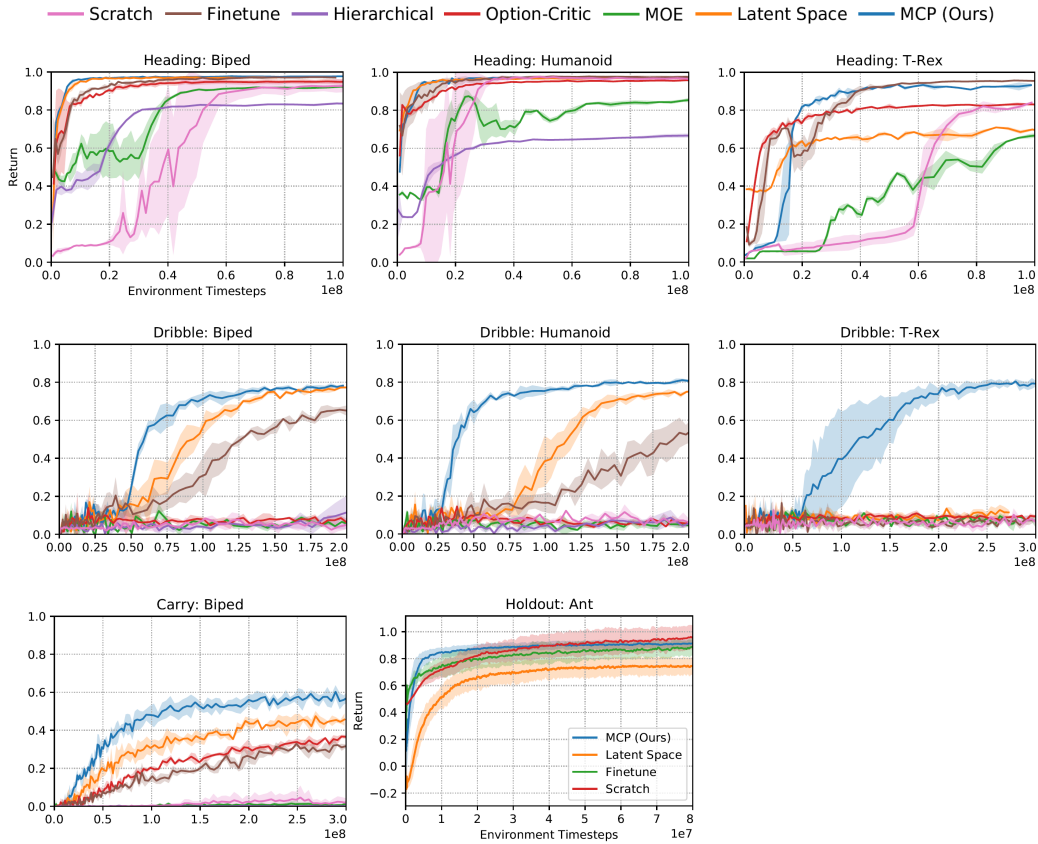

Figure 8: Learning curves of the various models when applied to transfer tasks. MCP improves learning speed and performance on challenging tasks (e.g. carry and dribble), and is the only method that succeeds on the most difficult task (Dribble: T-Rex).

Environment	Scratch	Finetune	Hierarchical	Option-Critic	MOE	Latent Space	MCP (Ours)
Heading: Biped	0.927 ± 0.032	0.970 ± 0.002	0.834 ± 0.001	0.952 ± 0.012	0.918 ± 0.002	0.970 ± 0.001	$\mathbf{0.976 \pm 0.002}$
Heading: Humanoid	0.965 ± 0.010	$\mathbf{0.975 \pm 0.008}$	0.681 ± 0.006	0.958 ± 0.001	0.857 ± 0.018	0.969 ± 0.002	0.970 ± 0.003
Heading: T-Rex	0.840 ± 0.003	$\mathbf{0.953 \pm 0.004}$	–	0.830 ± 0.004	0.672 ± 0.011	0.686 ± 0.003	0.932 ± 0.007
Carry: Biped	0.027 ± 0.035	0.324 ± 0.014	0.001 ± 0.002	0.346 ± 0.011	0.013 ± 0.013	0.456 ± 0.031	$\mathbf{0.575 \pm 0.032}$
Dribble: Biped	0.072 ± 0.012	0.651 ± 0.025	0.546 ± 0.024	0.046 ± 0.008	0.073 ± 0.021	0.768 ± 0.012	$\mathbf{0.782 \pm 0.008}$
Dribble: Humanoid	0.076 ± 0.024	0.598 ± 0.030	0.198 ± 0.002	0.058 ± 0.007	0.043 ± 0.021	0.751 ± 0.006	$\mathbf{0.805 \pm 0.006}$
Dribble: T-Rex	0.065 ± 0.032	0.074 ± 0.011	–	0.098 ± 0.013	0.070 ± 0.017	0.115 ± 0.013	$\mathbf{0.781 \pm 0.021}$
Holdout: Ant	$\mathbf{0.951 \pm 0.093}$	0.885 ± 0.062	–	–	–	0.745 ± 0.060	0.812 ± 0.030

Table 2: Performance statistics of different models on transfer tasks.

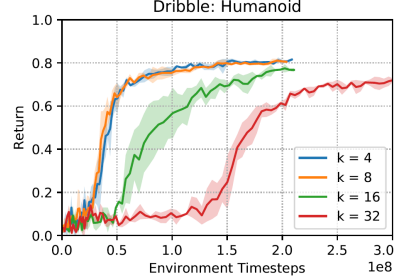| Property	Biped	Humanoid	T-Rex
Links	12	13	20
Total Mass (kg)	42	45	54.5
Height (m)	1.34	1.62	1.66
Degrees-of-Freedom	23	34	55
State Features	105	196	261
Action Parameters	17	28	49

Table 3: Properties of the characters.

Table 4: Learning curves of MCP with different numbers of primitives k.

8 Reference Motions

During pre-training, the primitives are trained by imitating a corpus of reference motions. The biped and humanoid share the same set of reference motions, consisting of mocap clips of walking and turning motions collected from a publicly available database [35]. In total, 230 seconds of motion data is used to train the biped and humanoid. To retarget the humanoid reference motions to the biped, we simply removed extraneous joints in the upper body (e.g. arms and head). The reference motions for the T-Rex consist of artist generated keyframe animations. Due to the cost of manually authored animations, the T-Rex is trained with substantially less motion data than the other characters. In total, 11 seconds of motion data is used to train the T-Rex. The T-Rex motions include 1 forward walk, 2 left turns, and 2 right turns. Despite having access to only a small corpus of reference motions, MCP is nonetheless able to learn a flexible set of primitives that enables the complex T-Rex character to perform challenging tasks.

9 Transfer Tasks

Heading: First we consider a simple heading task, where the objective is for the character to move in a target heading direction $\hat{\theta}_t$. The heading is changed every timestep by applying a random perturbation $\hat{\theta}_t = \hat{\theta}_{t-1} + \nabla\theta_t$ sampled from a uniform distribution $\nabla\theta_t \sim \mathrm{Uniform}(-0.15\mathrm{rad}, 0.15\mathrm{rad})$. The goal $g_t = (\cos(\hat{\theta}_t), -\sin(\hat{\theta}_t))$ encodes the heading as a unit vector along the horizontal plane. The reward r_t encourages the character to follow the target heading, and is computed according to

$$r_t = \exp\left(-4\ (\hat{u} \cdot v_{com} - \hat{v})^2\right).$$

Here, $(\cdot)$ denotes the dot product, v_{com} represents the character's center-of-mass (COM) velocity along the horizontal plane, $\hat{v} = 1m/s$ represents the target speed that the character should travel in along the target direction $\hat{u} = (\cos(\hat{\theta}_t), -\sin(\hat{\theta}_t))$.

Carry: To evaluate our method's performance on long horizon tasks, we consider a mobile manipulation task, where the goal is for the character to move a box from a source location to a target location. The task can be decomposed into a sequence of subtasks, where the character must first pickup the object from the source location, before carrying it to the target location and placing it on the table. To enable the character to carry the box, when the character makes contact with the box at the source location with a specific link (e.g. torso), a virtual joint is created that attaches the box to the character. Once the box has been placed at the target location, the joint is detached.

The box has a mass of 5kg and is initialized to a random source location at a distance of $[0\text{m}, 10\text{m}]$ from the character. The target is initialized to a distance $[0\text{m}, 10\text{m}]$ from the source. The goal $g_t = (x_{tar}, q_{tar}, x_{src}, q_{src}, x_b, q_b, v_b, \omega_b)$ encodes the target table's position x_{tar} and orientation q_{tar} as represented as a quaternion, the source table's position x_{src} and orientation q_{src}, and box's position x_b, orientation q_b, linear velocity v_b, and angular velocity ω_b. The reward function consists of terms that encourage the character to move towards the box, as well as to move the box towards the target,

$$r_t = w^{cv} r_t^{cv} + w^{cp} r_t^{cp} + w^{bv} r_t^{bv} + w^{bp} r_t^{bp},$$

r_t^{cv} encourages the character to move towards the box, while r_t^{cp} encourages the character to stay near the box,

$$r_t^{cv} = \exp\left(-1.5 \min\left(0, u_b \cdot v_{com} - \hat{v}\right)^2\right)$$

$$r_t^{cp} = \exp\left(-0.25 \left\|x_{com} - x_b\right\|^2\right).$$

u_b represents the unit vector pointing in the direction of the box with respect to the character's COM, v_{com} is the COM velocity of the character, $\hat{v} = 1m/s$ is the target speed, x_{com} is the COM position, and x_b is the box's position. All quantities are expressed along the horizontal plane. Similarly, r_t^{bv} and r_t^{bp} encourages the character to move the box towards the target,

$$r_t^{bv} = \exp\left(-1 \min\left(0, u_{tar} \cdot v_b - \hat{v}\right)^2\right)$$

$$r_t^{bp} = \exp\left(-0.5 \left\|x_b - x_{tar}\right\|^2\right).$$

u_{tar} represents the unit vector pointing in the direction of the target with respect to the box, v_b is the velocity of the box, and x_{tar} is the target location. The weights for the reward terms are specified according to $(w^{cv}, w^{cp}, w^{bv}, w^{bp}) = (0.1, 0.2, 0.3, 0.4)$.

Dribble: This task poses a challenging combination of locomotion and object manipulation, where the goal is for the character to move a soccer ball to a target location. Since the policy does not have direct control over the ball, it must rely on complex contact dynamics in order to manipulate the movement of the ball while also maintaining balance. The ball is randomly initialized at a distance of $[0\text{m}, 10\text{m}]$ from the character, and the target is initialized to a distance of $[0\text{m}, 10\text{m}]$ from the ball. The goal $g_t = (x_{tar}, x_b, q_b, v_b, \omega_b)$ encodes the target location x_{tar}, and ball's position x_b, orientation q_b, linear velocity v_b, and angular velocity ω_b. The reward function for this task follows a similar structure as the reward for the carry task, consisting of terms that encourage the character to move towards the ball, as well as to move the ball towards the target,

$$r_t = w^{cv} r_t^{cv} + w^{cp} r_t^{cp} + w^{bv} r_t^{bv} + w^{bp} r_t^{bp},$$

r_t^{cv} encourages the character to move towards the ball, while r_t^{cp} encourages the character to stay near the ball,

$$r_t^{cv} = \exp\left(-1.5 \min\left(0, u_b \cdot v_{com} - \hat{v}\right)^2\right)$$

$$r_t^{cp} = \exp\left(-0.5 \left\|x_{com} - x_b\right\|^2\right).$$

u_b represents the unit vector pointing in the direction of the ball with respect to the character's COM, v_{com} is the character's COM velocity, $\hat{v} = 1m/s$ is the target speed, x_{com} is the COM position, and x_b is the ball's position. Similarly, r_t^{bv} and r_t^{bp} encourages the character to move the ball towards the target,

$$r_t^{bv} = \exp\left(-1 \min\left(0, u_{tar} \cdot v_b - \hat{v}\right)^2\right)$$

$$r_t^{bp} = \exp\left(-0.5 \left\|x_b - x_{tar}\right\|^2\right).$$

u_{tar} represents the unit vector pointing in the direction of the target with respect to the ball, v_b is the velocity of the ball, and x_{tar} is the target location. The weights for the reward terms are specified according to $(w^{cv}, w^{cp}, w^{bv}, w^{bp}) = (0.1, 0.1, 0.3, 0.5)$.

14

Holdout: The holdout task is based on the standard Gym `Ant-v3` environment. The goal $g_t = \left(\cos(\hat\theta), \sin(\hat\theta) \right)$ specifies a two-dimensional vector that represents the target direction $\hat\theta$ that the character should travel in. The reward function is similar to that of the standard `Ant-v3` environment:

$$r_t = w^{\text{forward}} r_t^{\text{forward}} + w^{\text{healthy}} r_t^{\text{healthy}} + w^{\text{control}} r_t^{\text{control}} + w^{\text{contact}} r_t^{\text{contact}},$$

but the forward reward $r_t^{forward}$ is modified to reflect the target direction $\hat u = \left(\cos(\hat\theta), \sin(\hat\theta) \right)$:

$$r_t^{\text{forward}} = \hat u \cdot v_{com}$$

where v_{com} represents the character's COM velocity along the horizontal plane. The weights of the reward terms are specified according to $\left(w^{\text{forward}}, w^{\text{healthy}}, w^{\text{control}}, w^{\text{contact}} \right) = (1.0, 1.0, 0.5, 0.0005)$. During pre-training, the policies are trained with directions $\hat\theta \in [0, 3/2\pi]$. During transfer, the policies are trained with directions sampled from a holdout set $\hat\theta \in [3/2\pi, 2\pi]$.

10 Model Setup

All models are trained using proximal policy optimization (PPO) [34], except for the option-critic model, which follows the update rules proposed by Bacon et al. [2]. A discount factor of $\gamma = 0.95$ is used during pre-training, and $\gamma = 0.99$ is used for the transfer tasks. The value functions for all models are trained using multi-step returns with TD(λ) [37]. The advantages for policy gradient calculations are computed using the generalized advantage estimator GAE(λ) [33]. We detail the hyperparmater settings for each model in the following sections.

10.1 MCP

The MCP model follows the architecture detailed in Figure 3. The value function $V(s, g)$ is modeled with a fully-connected network with 1024 and 512 hidden units, followed by a linear output unit. Hyperparameter settings are available in Table 5.

Parameter	Biped	Humanoid	T-Rex
k Primitives	8	8	8
π Stepsize (Pre-Train)	2×10^{-5}	1×10^{-5}	1×10^{-5}
π Stepsize (Transfer)	5×10^{-5}	5×10^{-5}	5×10^{-5}
V Stepsize	1×10^{-2}	1×10^{-2}	1×10^{-2}
Batch Size	4096	4096	4096
Minibatch Size	256	256	256
SGD Momentum	0.9	0.9	0.9
TD(λ)	0.95	0.95	0.95
GAE(λ)	0.95	0.95	0.95
PPO Clip Threshold	0.02	0.02	0.02

Table 5: MCP model hyperparamters.

10.2 Scratch

As a baseline, we train a model from scratch for each transfer task. The policy network consists of two fully-connected layers with 1024 and 512 ReLU units, followed by a linear output layer that outputs the mean of a Gaussian distribution $\mu(s, g)$. The covariance matrix is represented by a fixed diagonal matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, ...)$ with manually specified values for σ_i. The value function follows a similar architecture, but with a single linear output unit. Hyperparameter settings are available in Table 6.

Parameter	Biped	Humanoid	T-Rex
π Stepsize	2.5×10^{-6}	2.5×10^{-6}	1×10^{-6}
V Stepsize	1×10^{-2}	1×10^{-2}	1×10^{-2}
Batch Size	4096	4096	4096
Minibatch Size	256	256	256
SGD Momentum	0.9	0.9	0.9
TD(λ)	0.95	0.95	0.95
GAE(λ)	0.95	0.95	0.95
PPO Clip Threshold	0.02	0.02	0.02

Table 6: Scratch model hyperparamters.

10.3 Finetuning

The finetuning model is first pre-trained to imitate a reference motion, and then finetuned on the transfer tasks. The network architecture is identical to the scratch model. Pre-training is done using the motion imitation approach proposed by Peng et al. [30]. When transferring to tasks with additional goal inputs g that are not present during training, the networks are augmented with additional inputs using the *input injection* method from Berseth et al. [3], which adds additional inputs to the network without modifying the initial behavior of the model. Hyperparameter settings are available in Table 7.

Parameter	Biped	Humanoid	T-Rex
π Stepsize	2.5×10^{-6}	2.5×10^{-6}	1×10^{-6}
V Stepsize	1×10^{-2}	1×10^{-2}	1×10^{-2}
Batch Size	4096	4096	4096
Minibatch Size	256	256	256
SGD Momentum	0.9	0.9	0.9
TD(λ)	0.95	0.95	0.95
GAE(λ)	0.95	0.95	0.95
PPO Clip Threshold	0.02	0.02	0.02

Table 7: Finetune model hyperparamters.

10.4 Hierarchical

The hierarchical model consists of a gating function $w(s, g)$ that specifies the probability of activating a particular low-level primitive $\pi_i(a|s)$ from a discrete set of primitives. To enable the primitives to be transferable between tasks with different goal representations, the hierarchical model follows a similar asymmetric architecture, where the primitives have access only to the state. During pre-training, each primitive is trained to imitate a different reference motion. All experiments use the same set of 7 primitives, including 1 primitive trained to walk forwards, 3 primitives trained to turn right at different rates, and 3 primitives trained to turn left at different rates. Once the primitives have been trained, their parameters are kept fixed, while a gating function is trained to sequence the primitives for each transfer task. The gating function selects a new primitive every walk cycle, which has a duration of approximately 1 second, the equivalent of about 30 timesteps. Each primitive is modeled using a separate network with a similar network architecture as the scratch model. The gating function is modeled with two fully-connected layers consisting of 1024 and 512 ReLU units, followed by a

softmax output layer that specifies the probability of activating each primitive. The gating function is also trained with PPO. Hyperparameter settings are available in Table 8.

Parameter	Biped	Humanoid
k Primitives	7	7
π Stepsize	1×10^{-3}	1×10^{-3}
V Stepsize	1×10^{-2}	1×10^{-2}
Batch Size	4096	4096
Minibatch Size	256	256
SGD Momentum	0.9	0.9
TD(λ)	0.95	0.95
GAE(λ)	0.95	0.95
PPO Clip Threshold	0.02	0.02

Table 8: Hierarchical model hyperparamters.

10.5 Option-Critic

The option-critic model adapts the original implementation from Bacon et al. [2] to continuous action spaces. During pre-training, the model is trained end-to-end with the motion imitation tasks. Unlike the hierarchical model, the options (i.e. primitives) are not assigned to a particular skills, and instead specialization is left to emerge automatically from the options framework. To enable transfer of options between different tasks, we also use an asymmetric architecture, where the intra-option policies $\pi_\omega(a|s)$ and termination functions $\beta_\omega(s)$ receive only the state as input. The policy over options $\pi_\Omega(\omega|s, g)$, as defined by the option value function $Q_\Omega(s, g, \omega)$, has access to both the state and goal. When transferring the options to new tasks, the parameters of π_ω and β_ω are kept fixed, and a new option value function Q_Ω is trained for the new task. We have also experimented with finetuning π_ω and β_ω on the transfer tasks, but did not observe noticeable performance improvements. Furthermore, joint finetuning often results in catastrophic, where the options degrade to producing highly unnatural behaviours. Therefore, all experiments will have π_ω and β_ω fixed when training on the transfer tasks. Hyperparameter settings are available in Table 9.

Parameter	Biped	Humanoid	T-Rex
k Options	8	8	8
π Stepsize	2.5×10^{-6}	2.5×10^{-6}	1×10^{-6}
β_ω Stepsize	2.5×10^{-6}	2.5×10^{-6}	1×10^{-6}
Q_Ω Stepsize	1×10^{-2}	1×10^{-2}	1×10^{-2}
Batch Size	256	256	256
SGD Momentum	0.9	0.9	0.9
ξ Termination Cost	0.01	0.01	0.01

Table 9: Option-critic model hyperparamters.

10.6 Mixture-of-Experts

The mixture-of-experts (MOE) model is implemented according to Equation 1. The policy consists of a set of primitives $\pi_i(a|s)$ and gating function $w(s, g)$ that specifies the probability of activating each primitive. To facilitate transfer, the primitives only receives the state as input, while the gating function receives both the state and the goal. The primitives are first pre-trained with the motion imitation task, and when transferring to new tasks, the parameters of the primitives are kept fixed, while a new gating function is trained for each transfer task. Therefore, MOE is analogous to MCP where only a single primitive is activated at each timestep. The gating function and the primitives are modeled by separate networks. The network for the gating function consists of 1024 and 512 ReLU units, followed by a softmax output layer that specifies $w_i(s, g)$ for each primitive. The primitives are modeled jointly by a single network consisting of 1024 and 512 ReLU units, followed separate linear output layers for each primitives that specifies the parameters of a Gaussian. As such, the MOE model's action distribution is modeled as a Gaussian mixture model. Hyperparameter settings are available in Table 10.

Parameter	Biped	Humanoid	T-Rex
k Primitives	8	8	8
π Stepsize	1×10^{-5}	5×10^{-6}	2×10^{-6}
V Stepsize	1×10^{-2}	1×10^{-2}	1×10^{-2}
Batch Size	4096	4096	4096
Minibatch Size	256	256	256
SGD Momentum	0.9	0.9	0.9
TD(λ)	0.95	0.95	0.95
GAE(λ)	0.95	0.95	0.95
PPO Clip Threshold	0.02	0.02	0.02

Table 10: Mixture-of-experts model hyperparamters.

10.7 Latent Space

The latent space model follows a similar architecture as Merel et al. [25], where an encoder $q(w_t|g_t)$ first maps the goal g_t to a distribution over latent variables w_t. w_t is then sampled from the latent distribution and provided to the policy as input $\pi(a_t|s_t, w_t)$. The latent distribution is modeled as an IID Gaussian $q(w_t|g_t) = \mathcal{N}(\mu_q(g_t), \Sigma_q(g_t))$ with mean $\mu_q(w_t)$ and diagonal covariance matrix $\Sigma_q(g_t)$. Similar to VAEs, we include a term in the objective that regularizes the latent distribution against a standard Gaussian prior $p_0(w_t) = \mathcal{N}(0, I)$,

$$\arg \max_{\pi, q} \quad \mathbb{E}_{\tau \sim p_{\pi,q}(\tau)} \left[\sum_{t=0}^{T} \gamma^t r_t \right] + \beta \, \mathbb{E}_{g_t \sim p(g_t)} \left[\mathrm{D}_{KL} \left[q(w_t|g_t) || p_0(w_t) \right] \right] \tag{5}$$

Here, β is a manually specified coefficient for the KL regularizer. The encoder and policy are trained end-to-end using the reparameterization trick [20].

The latent space model follows a similar pre-training procedure as the MCP model, where the model is trained to imitate a corpus of reference motions with the goal $g_t = (\hat{s}_{t+1}, \hat{s}_{t+2})$ specifying the target states for the next two timesteps. The encoder is therefore trained to embed short motion clips into the latent space. After pre-training, the parameters of π are frozen, and a new encoder $q'(w_t|s_t, g_t)$ is trained for each transfer task. Following the architectures from previous work [16, 25], the encoder used during pre-training only receives the goal g_t as input, while the encoder used in the transfer tasks receives both the state s_t and goal g_t as input, since additional information from the state may be necessary when performing the new tasks.

The policy network follows a similar architecture as the ones used by the finetuning model, consisting of two hidden with 1024 and 512 ReLU units followed by a linear output layer. The encoder used during pre-training consists of 256 and 128 hidden units, followed by a linear output layer for $\mu_q(g_t)$ and $\Sigma_q(g_t)$. The size of the encoding is set to be 8D, the same dimensionality as the weights of the gating function from the MCP model. The encoder used in the transfer tasks is modeled by a larger network with 1024 and 512 hidden units. Hyperparameter settings are available in Table 11.

Parameter	Biped	Humanoid	T-Rex
w Latent Size	8	8	8
π Stepsize (Pre-Train)	5×10^{-6}	2.5×10^{-6}	1×10^{-6}
π Stepsize (Transfer)	5×10^{-5}	5×10^{-5}	5×10^{-5}
V Stepsize	1×10^{-2}	1×10^{-2}	1×10^{-2}
Batch Size	4096	4096	4096
Minibatch Size	256	256	256
SGD Momentum	0.9	0.9	0.9
TD(λ)	0.95	0.95	0.95
GAE(λ)	0.95	0.95	0.95
PPO Clip Threshold	0.02	0.02	0.02
β KL Regularizer	1×10^{-4}	1×10^{-4}	1×10^{-4}

Table 11: Latent space model hyperparamters.