

A Theoretical Results

Theorem 1. *Given a positive integer L , ϕ_L is an optimal solution to \mathcal{J}_{PSD} if and only if it forms a regular $2L$ -gon of diameter L centered at the origin.*

Proof. We prove the claim by showing both directions of the equivalence.

(\Leftarrow) Suppose ϕ_L forms a regular $2L$ -gon of diameter L centered at the origin. Then, for all $(L, s_t, s_{t+1}, s_{t+L})$ in the replay buffer, we have $\|\phi_L(s_{t+L}) - \phi_L(s_t)\|_2 = L$ and $\phi_L(s_{t+L}) = -\phi_L(s_t)$, which implies $\|\phi_L(s_{t+L}) + \phi_L(s_t)\|_2 = 0$. Thus, the objective becomes $\mathcal{J}_{\text{PSD}} = \mathbb{E}[L - 0] = L$. Since Eq. (5) requires $\|\phi_L(s_{t+L}) - \phi_L(s_t)\|_2 \leq L$, any feasible solution must satisfy $\mathcal{J}_{\text{PSD}} \leq L$, and no higher value can be attained. Under this condition, the given regular $2L$ -gon satisfies both constraints in Eq. (5) and Eq. (6). Hence, ϕ_L is optimal.

(\Rightarrow) Suppose ϕ_L is optimal and achieves $\mathcal{J}_{\text{PSD}} = L$. Then $\|\phi_L(s_{t+L}) - \phi_L(s_t)\|_2 = L$ and $\|\phi_L(s_{t+L}) + \phi_L(s_t)\|_2 = 0$, implying $\phi_L(s_{t+L}) = -\phi_L(s_t)$ and $\phi_L(s_t)$ lies on a hypersphere of radius $L/2$ centered at the origin for all $(L, s_t, s_{t+1}, s_{t+L})$ in the replay buffer. From Eq. (6), we have $\|\phi_L(s_{t+1}) - \phi_L(s_t)\|_2 \leq L \sin(\pi/2L)$, which implies that the maximum angular distance between adjacent points is π/L . Under this condition, reaching the antipodal point $\phi_L(s_{t+L}) = -\phi_L(s_t)$ starting from $\phi_L(s_t)$ is only possible if the points are equally spaced along the circumference of a great circle on the hypersphere, with an angular distance of exactly π/L between adjacent points. Hence, ϕ_L forms a regular $2L$ -gon of diameter L centered at the origin. \square

B Experimental Details

B.1 Environments

MuJoCo locomotion environments We adopt MuJoCo environments including Ant, HalfCheetah, Humanoid, Hopper, and Walker2D [7, 43] to evaluate our method and baselines. Episode lengths are set to 200 timesteps for Ant and HalfCheetah, and 400 timesteps for Humanoid, Hopper, and Walker2D. For state-based observations, we follow the default Gym setting [7], which includes proprioceptive information in the observation space. However, since both CSD and METRA rely on global position information to construct their latent representations, we include the global position in the observation when applying these methods, following the setups described in their original papers. For the pixel-based experiments of PSD, we use $90 \times 90 \times 3$ RGB images captured from a tracking camera (view shown in Figure 7) as input to both the RL agent and the encoder ϕ , without incorporating any additional proprioceptive information.

Downstream tasks environments In the HalfCheetah-hurdle and Walker2D-hurdle, the high-level policy receives a reward whenever the agent successfully jumps over a hurdle. For HalfCheetah-hurdle, the hurdle positions are [2.5, 4.0, 7.0, 10.0, 15.0, 22.0, 30.0], with a height of 0.26, which is higher than the setting used in METRA [28]. For Walker2D-hurdle, the hurdle positions are [1.2, 2.7, 4.1, 5.8, 7.0, 9.2, 11.0, 12.8, 14.2], with a height of 0.11. In both environments, the hurdles are unevenly spaced, requiring multi-timescale coordination for successful locomotion. We provide the distance to the nearest hurdle as part of the task-specific input s_{task} to the high-level policy. The episode lengths are set to 300 timesteps for HalfCheetah-hurdle and 600 timesteps for Walker2D-hurdle.

In the HalfCheetah-friction and Walker2D-friction, the agent is rewarded for maintaining forward velocity, which encourages robust locomotion while avoiding falls under changing friction conditions. For implementation simplicity, we do not modify the ground friction directly. Instead, we sequentially change the friction parameters of the agent’s feet in the XML file every 100 timesteps, cycling through the values [0.5, 1.5, 2.0], given that the default friction parameter in MuJoCo is 1.0. Additionally, the current friction coefficient is provided as task-specific input s_{task} to the high-level policy, enabling it to adapt to the changing frictions. The episode lengths are set to 500 for both HalfCheetah-friction and Walker2D-friction.

B.2 Implementation Details

We implement PSD on top of the publicly available PyTorch SAC implementation¹. For fair comparison, we implement all baseline methods within the same codebase as PSD to ensure consistency in training procedures and infrastructure. To train the high-level policy for downstream tasks, we use PPO implemented in a public PyTorch repository². All experiments are conducted on an NVIDIA A6000 GPU, and training for each task typically completes within 24 hours.

Training of PSD For training PSD, we uniformly sample four discrete values of the period variable L from the range $[L_{\min}, L_{\max}]$, including both bounds, to ensure coverage of the range while maintaining training efficiency. As shown in Appendix C, we found that this sampling strategy is sufficient, as the model is able to generalize to intermediate L values via interpolation.

Additionally, since L is a scalar value, directly feeding it into the encoder ϕ_L , the policy π , and the Q-function may limit the representational capacity of these networks. To address this, we apply sinusoidal positional embeddings—commonly used in transformers [45] and diffusion models [17]—to project L into a higher-dimensional space. As an example, rather than using L directly in the policy in the form of $\pi(a | s, L)$, we use $\pi(a | s, \text{Embed}(L))$, where $\text{Embed}(L)$ denotes the embedded representation of L as follows:

$$\text{Embed}(L) = [e_0, e_1, \dots, e_{D-1}], \quad \text{where } e_i = \begin{cases} \sin(L \cdot \omega_i) & \text{if } i \bmod 2 = 0, \\ \cos(L \cdot \omega_i) & \text{if } i \bmod 2 = 1, \end{cases}$$

where the frequency term ω_i is defined as $\omega_i = 10000^{-2 \cdot \lfloor i/2 \rfloor / D}$. We apply this sinusoidal embedding to the period variable L whenever it is used as input to the network, enabling the model to better distinguish and generalize across different temporal scales. In addition, we found that using fixed values for λ_1 and λ_2 works well in practice, when optimizing \mathcal{J}_{PSD} via dual gradient descent method. The full set of hyperparameters is summarized in Table 2.

Training of baseline methods For baseline methods, we closely followed the implementation details described in their original papers. For METRA, we use a 2-dimensional continuous skill vector $z \in \mathbb{R}^2$ for Ant and Humanoid, and 16-dimensional discrete skills for other environments. In CSD, a 16-dimensional discrete skill vector is used for all environments. For both METRA and CSD, continuous skills are sampled from a standard Gaussian distribution and normalized to have unit norm, and discrete skills are designed to be zero-centered one-hot vectors. For DADS, we use 2-dimensional continuous skills sampled from the uniform range $[-1, 1]^2$ for Ant and Humanoid, and 16-dimensional one-hot vectors for the remaining environments. In DIAYN, we use 16-dimensional one-hot vectors across all environments. For training the low-level policy for downstream tasks, we use 16-dimensional discrete skills for all baseline methods.

Adaptive sampling method For the adaptive sampling method, we evaluate the periodic skill policy conditioned on the current boundary periods every 1K episodes. To measure performance, we roll out 5 episodes and compute the average cumulative sum of r_{PSD} , defined as $\bar{R}_L = \mathbb{E}_{p(\tau, L)}[\sum_{t=0}^{T-1} r_{\text{PSD}}]$. For the threshold coefficients α and β , we found that setting $\alpha = 0.9$ and $\beta = 0.4$ works well in practice. To avoid abrupt narrowing of the radius range in the early stages of training, each bound is allowed to shrink only after it has first been expanded, i.e., after $\bar{R}_L > 0.9T$ has been satisfied at least once. The full algorithm is described in Algorithm 2, and a complete list of hyperparameters is provided in Table 2.

Training of PSD with pixel-based observations For experiments using pixel-based observations, we use a CNN-based encoder [22] to process visual inputs. To capture temporal continuity, we concatenate consecutive frames as input. We also apply random cropping as a form of data augmentation, following CURL [20]. We found that action repeat was not necessary to achieve stable training in our setup. A complete list of hyperparameters is provided in Table 3.

Task-specific reward Since PSD is designed to enrich the agent’s behavior with additional diversity while still achieving the primary task, we optionally combine the velocity-based external reward r_{ext} with the intrinsic reward r_{PSD} .

¹<https://github.com/pranz24/pytorch-soft-actor-critic>

²<https://github.com/nikhilbarhate99/PPO-PyTorch>

Given that r_{PSD} is bounded in the range $(0, 1]$, we design the external reward to also have an upper bound of 1, ensuring a balanced contribution of both rewards when the agent reaches optimal performance, as follows:

$$r_{\text{ext}}(v_x) = \begin{cases} 1 & \text{if } v_x \geq v_x^* \\ v_x/v_x^* & \text{otherwise} \end{cases}$$

This reward function assigns $r_{\text{ext}} = 1$ when the agent’s forward velocity v_x exceeds the threshold v_x^* , and increases linearly as v_x approaches the threshold from below. We set $v_x^* = 0.5$ for Ant and HalfCheetah, and $v_x^* = 1.0$ for Humanoid, Hopper, and Walker2D.

Training high-level policy for downstream tasks For downstream tasks, we train the high-level policy using PPO [35] with a task-specific reward and additional observation s_{task} described in Appendix B.1, while keeping the low-level skill policies frozen. The high-level policy $\pi^h(L | s_{\text{task}}, s)$ (or $\pi^h(z | s_{\text{task}}, s)$ for baseline methods) selects a skill every H steps, and the selected skill is provided as input to the low-level policy, which is then executed for H steps. A complete list of hyperparameters is provided in Table 4.

Training of PSD Combined with METRA METRA [28] learns an encoder ϕ_m and a skill policy $\pi(a | s, z)$ that encourages transitions to deviate maximally along latent directions z , while constraining the one-step latent distance to capture temporal coherence. As described in the original paper [28], the constrained objective in Eq. (9) is optimized by maximizing the following components:

$$\mathcal{J}_{\text{METRA}, \phi_m} = \mathbb{E}_{(s, s', z) \sim \mathcal{D}} [(\phi_m(s') - \phi_m(s))^\top z + \lambda_m \cdot \min(\epsilon, 1 - \|\phi_m(s') - \phi_m(s)\|_2^2)], \quad (11)$$

$$\mathcal{J}_{\text{METRA}, \lambda_m} = -\lambda_m \cdot \mathbb{E}_{(s, s', z) \sim \mathcal{D}} [\min(\epsilon, 1 - \|\phi_m(s') - \phi_m(s)\|_2^2)], \quad (12)$$

where λ_m is a Lagrange multiplier, updated during training via the dual gradient method to enforce the constraint.

A naïve combination of PSD and METRA—training their encoders *independently* and simply summing their intrinsic rewards—fails in practice. As explained in section 2, the METRA objective strongly favors skills with the shortest possible period. This is because shorter periods typically correspond to faster motions, which lead to larger per-step deviations in the latent space and thus yield higher values of $(\phi_m(s') - \phi_m(s))^\top z$. Consequently, all discovered skills collapse into a single short-periodic behavior, undermining the diversity of the learned skill of PSD.

To address this issue, we condition each encoder on the other method’s skill variable by incorporating it as an additional input to the state. Specifically, we augment the input to ϕ_L with the skill vector z from METRA, and the input to ϕ_m with the period variable L from PSD, resulting in:

$$\phi_L(s) \longrightarrow \phi_L(s, z), \quad \phi_m(s) \longrightarrow \phi_m(s, L).$$

This mutual conditioning allows each encoder to account for the temporal properties imposed by the other method, thereby regularizing their joint optimization and preventing skill collapse. For example, from the perspective of training $\phi_m(s, L)$, the METRA objective in Eqs. (11) and (12) encourages latent representations that exhibit large per-step deviations in the latent space while satisfying the periodicity determined by L .

By jointly optimizing both encoders with this conditioning, we obtain a policy $\pi(a | s, z, L)$ that can independently modulate both the temporal direction (i.e., the skill variable z) and the temporal length (i.e., the period variable L) in a fully unsupervised manner. The full algorithm is described in Algorithm 3 and a complete list of hyperparameters is provided in Table 5.

B.3 Visualizations

PCA visualization for latent space As described in Section 3.2, the circular latent space of PSD is not necessarily 2-dimensional. Given this formulation, we map states s to latent vectors $\phi_m(s)$ with 3 or more dimensions in practice to better capture periodicity. To visualize this circular latent space, as shown in Figure 5 and [our video](#), we apply Principal Component Analysis (PCA) to obtain a 2-dimensional projection that represents the underlying circular structure.

Algorithm 2 Adaptive Sampling Method

```
1: Initialize: policy  $\pi$ , encoder  $\phi$ , current sampling bound  $L_{\min}, L_{\max}$ 
2:  $updated\_once\_min \leftarrow \text{False}$ 
3:  $updated\_once\_max \leftarrow \text{False}$ 
   // Evaluate current bounds at  $L_{\min}$  and  $L_{\max}$ 
4: for each evaluation episode do
5:   for  $L \in \{L_{\min}, L_{\max}\}$  do
6:     Execute  $\pi(a|s, L)$  for the entire episode
7:     Compute cumulative reward  $\sum_{t=0}^{T-1} r_{\text{PSD}}(L)$ 
8:   end for
9: end for
10: Compute average reward  $\bar{R}_{L_{\min}}, \bar{R}_{L_{\max}}$ 
   // Update current bounds
11: if  $\bar{R}_{L_{\min}} > \alpha T$  then
12:    $L_{\min} \leftarrow L_{\min} - N$ 
13:    $updated\_once\_min \leftarrow \text{True}$ 
14: end if
15: if  $\bar{R}_{L_{\max}} > \alpha T$  then
16:    $L_{\max} \leftarrow L_{\max} + N$ 
17:    $updated\_once\_max \leftarrow \text{True}$ 
18: end if
19: if  $\bar{R}_{L_{\min}} < \beta T$  and  $updated\_once\_min == \text{True}$  then
20:    $L_{\min} \leftarrow L_{\min} + N$ 
21: end if
22: if  $\bar{R}_{L_{\max}} < \beta T$  and  $updated\_once\_max == \text{True}$  then
23:    $L_{\max} \leftarrow L_{\max} - N$ 
24: end if
25: return  $L_{\min}, L_{\max}$ 
```

Algorithm 3 PSD Combined with METRA

```
1: Initialize: policy  $\pi$ , PSD encoder  $\phi$ , METRA encoder  $\phi_m$ , sampling bound  $L_{\min, \max}$ , replay
   buffer  $\mathcal{D}$ , Lagrange multiplier  $\lambda_{1,2,m}$ 
2: for each training epoch do
3:   Update  $L_{\min}, L_{\max}$  if AdaptiveSampling is enabled
   // Environment interaction
4:   for each episode in the epoch do
5:     Sample  $L \sim p(L)$  where  $L \in [L_{\min}, L_{\max}]$ 
6:     Sample  $z \sim p(z)$  where  $p(z)$  is the skill prior of METRA
7:     Execute  $\pi(a|s, z, L)$  for the entire episode, and store transitions  $(z, L, s_t, a_t, s_{t+1})$  in  $\mathcal{D}$ 
8:   end for
   // Update encoder and RL network
9:   Update  $\phi_L(s, z)$  by maximizing  $\mathcal{J}_{\text{PSD}, \phi}$  using samples  $(z, L, s_t, s_{t+1})$  from  $\mathcal{D}$ 
10:  Update  $\phi_m(s, L)$  by maximizing  $\mathcal{J}_{\text{METRA}, \phi_m, \lambda_m}$  using samples  $(z, L, s_t, s_{t+1}, s_{t+L})$  from  $\mathcal{D}$ 
11:  Compute intrinsic reward  $r_{\text{PSD}}$ 
12:  Compute intrinsic reward  $r_{\text{METRA}}$ 
13:  Update  $\pi(a|s, z, L)$  with  $\alpha_{\text{PSD}} \cdot r_{\text{PSD}} + r_{\text{METRA}}$  using SAC
14: end for
```

Table 2: Hyperparameters for training PSD.

Parameter	Value
Learning rate	1×10^{-4}
Discount factor γ	0.99
Optimizer	Adam
N of episodes per epoch	8
N of gradient steps per epoch	64
Replay buffer size	5×10^5
Minibatch size	1024 (ϕ_L), 256 (for others)
Target smoothing coefficient	0.995
Entropy coefficient	Auto-tuned
Circular latent dimension d	{3, 6}
Output dimension of the positional encoding D	8
$\mathcal{J}_{\text{PSD}} \epsilon$	10^{-5}
Initial λ_1	5 (Ant, HalfCheetah), 10 (Humanoid, Hopper, Walker2D)
Initial λ_2	5 (Ant, HalfCheetah), 10 (Humanoid, Hopper, Walker2D)
N of hidden layers	2
N of hidden units per layer	1024
Step size of adaptive sampling N	1
Adaptive sampling interval	1000 episodes
N of evaluation episodes for adaptive sampling	5
Thresholds (α, β) for adaptive sampling	(0.9, 0.4)

Table 3: Hyperparameters for training PSD with Pixel-based observation (others same as Table 2).

Parameter	Value
Replay buffer size	3×10^4
Minibatch size	512 (ϕ_L), 256 (for others)
Circular latent dimension d	{3, 6}
Output dimension of the positional encoding D	128
Initial λ_1	5 (Ant), 3 (HalfCheetah)
Initial λ_2	5 (Ant), 3 (HalfCheetah)
Encoder	CNN [22]
Random Crop	$90 \times 90 \rightarrow 84 \times 84$
N of stacked frames	3
N of action repeat	1

Table 4: Hyperparameters for training the high-level policy using PPO.

Parameter	Value
Learning rate	3×10^{-4} (actor), 1×10^{-3} (critic)
Discount factor γ	0.99
N of episodes per epoch	4
N of gradient steps per epoch	80
Batch size	256
PPO clipping parameter ϵ	0.2
N of hidden layers	2
N of hidden units per layer	256

Table 5: Hyperparameters for training PSD Combined with METRA.

Parameter	Value
Learning rate	1×10^{-4}
Discount factor γ	0.99
Optimizer	Adam
N of episodes per epoch	8
N of gradient steps per epoch	64
Reward coefficient α_{PSD}	1.0
Replay buffer size	5×10^5
Minibatch size	1024 (ϕ_L), 256 (for others)
Target smoothing coefficient	0.995
Entropy coefficient	Auto-tuned
Skill dimension of METRA	2-D cont. (Ant), 1-D cont. (Walker2D)
Circular latent dimension d	6 (Ant), 3 (Walker2D)
Output dimension of the positional encoding D	6
$\mathcal{J}_{\text{PSD}} \epsilon$	10^{-5}
$\mathcal{J}_{\text{METRA}} \epsilon$	10^{-3}
Initial λ_1	5 (Ant), 10 (Walker2D)
Initial λ_2	5 (Ant), 10 (Walker2D)
Initial λ_m	30
N of hidden layers	2
N of hidden units per layer	512 (Ant), 1024 (Walker2D)

574 C Additional Experimental Results

575 C.1 Evolution of Learned Bounds via Adaptive Sampling

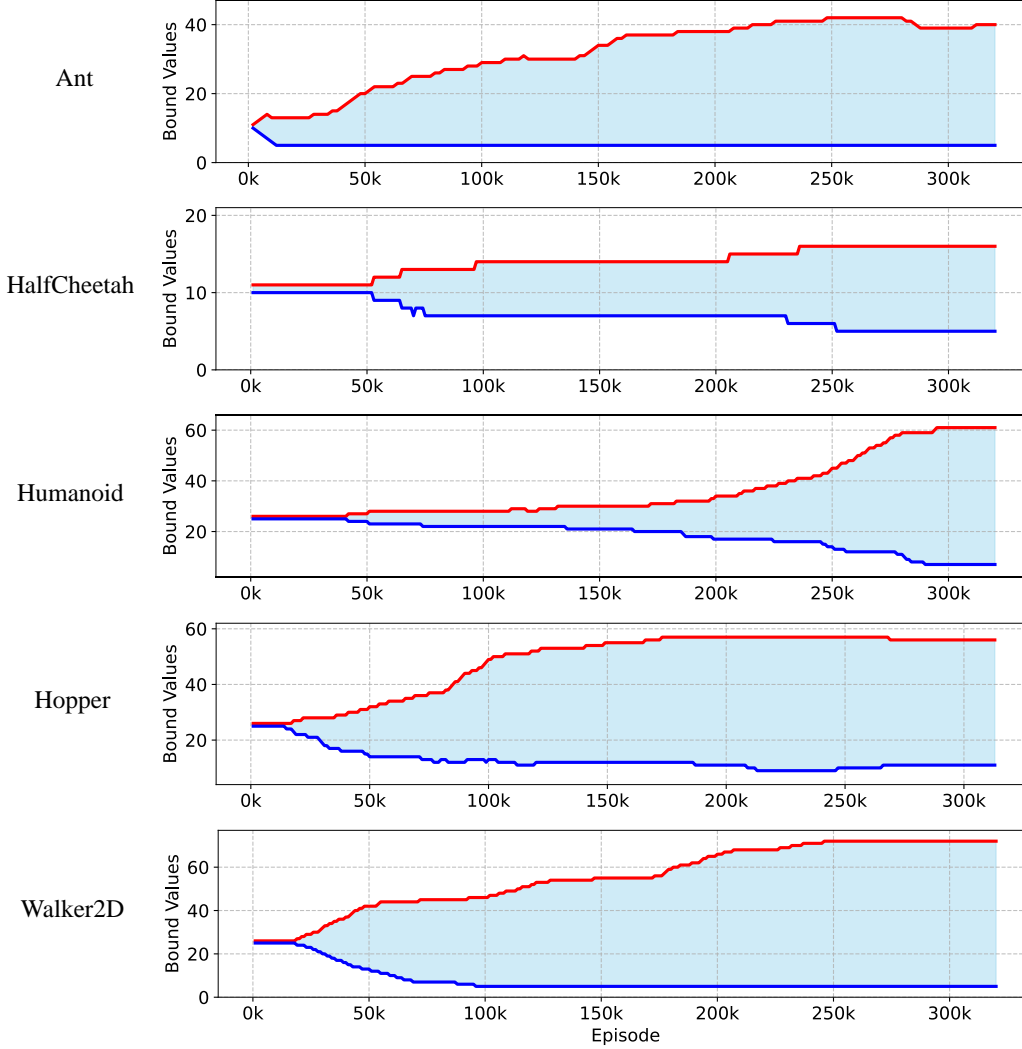


Figure 9: **Evolution of the L_{\min} , L_{\max} during training.** The figure shows how the period variable L_{\min} , L_{\max} evolves over training episodes with the adaptive sampling method applied to the Ant, HalfCheetah, Humanoid, Hopper, and Walker2D environments. As training progresses, increasingly challenging periods are proposed to the agent based on the average cumulative sum of r_{PSD} , enabling the discovery of a wider range of periodic behaviors.

576 In Figure 9, we visualize the evolution of the sampling range of L during training with the adaptive
 577 sampling method. To prevent the period variable L_{\min} , which must be a positive integer, from
 578 becoming too small, we set the minimum value $L_{\min} = 5$.

579 Although training begins with a single period value, the adaptive sampling method gradually proposes
 580 more challenging periods, enabling the agent to acquire skills across a broad range of dynamically
 581 feasible period lengths. Moreover, since training is conducted with the combined reward of r_{ext} and
 582 r_{PSD} (as described in Appendix B.2), the agent learns to maintain a velocity above the target v_x^* while
 583 acquiring maximally diverse skills to optimize the overall reward. A notable property of this method
 584 is that even if a proposed period is initially rejected due to low performance, the agent may later learn
 585 to handle it as training progresses. Overall, this method enables PSD to autonomously discover a
 586 wide range of periodic behaviors without requiring prior knowledge of agent-specific period scales.

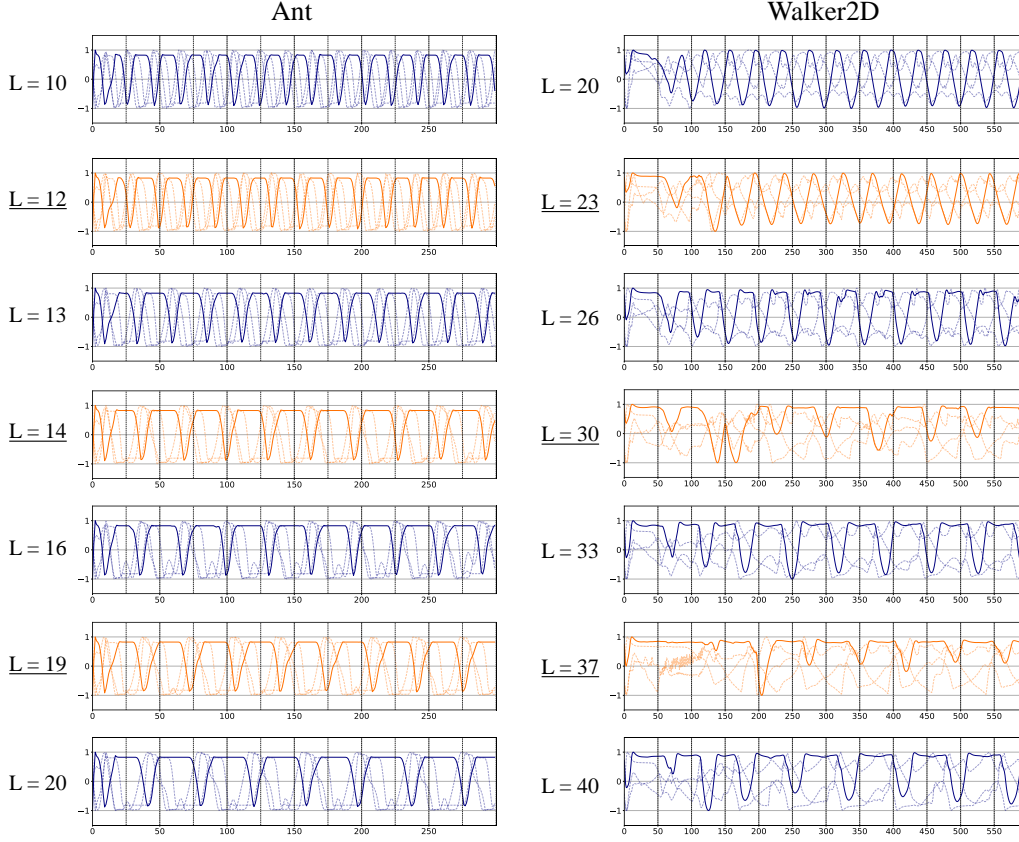


Figure 10: **Trajectories of the skill policy $\pi(a | s, L)$ under different values of L .** The figure shows representative joint trajectories of Ant (*left*) and Walker2D (*right*) generated by the skill policy $\pi(a | s, L)$ under different values of L . The blue trajectories are rollouts of the policy conditioned on the final sampling candidates after convergence, while the orange ones are generated using intermediate integer values between these candidates. Although the orange trajectory does not perfectly satisfy the $2L$ periodicity, it still generalizes well, indicating that our sampling strategy is effective and the circular representation of PSD generalizes across diverse periods.

588 C.3 Analysis of Pixel-based Observation Experiments

589 Comparing Figure 3 and Figure 7, we observe that the pixel-based HalfCheetah exhibits narrower
 590 and higher-frequency periodic behaviors than its state-based counterpart, despite having identical
 591 robot dynamics. We hypothesize that this is due to the inability to differentiate periodic variations
 592 in the vertical direction from pixel observations. As shown in our [video](#), it is difficult to perceive
 593 z -axis variations from raw images, and this issue is exacerbated by random cropping. In contrast, in
 594 state-based settings, the z -coordinate is explicitly provided as the first dimension of the observation
 595 vector, making it easier for the network to recognize vertical changes. This suggests that the ability
 596 to represent vertical height enables PSD to learn longer-period behaviors (e.g., jumping) in the
 597 state-based setting, as also visually confirmed in the video. On the other hand, in the Ant environment,
 598 both pixel-based and state-based observations provide similar levels of information, and thus result in
 599 similar walking behaviors.