

A Notations

- \mathcal{S} : State space.
- \mathcal{A} : Action space.
- \mathcal{T}_i : The i -th task, defined as an MDP $(\mathcal{S}, \mathcal{A}, P_i, r_i, \gamma)$.
- $P_i(s_{t+1} | s_t, a_t)$: Transition kernel for task \mathcal{T}_i .
- $r_i(s, a)$: Reward function for task \mathcal{T}_i .
- $\gamma \in [0, 1)$: Discount factor.
- $p(\mathcal{T})$: Distribution over possible tasks.
- $p(\mathcal{T}_{\text{train}}), p(\mathcal{T}_{\text{test}})$: Training and test distributions (subsets of $p(\mathcal{T})$).
- $\mathcal{T}_i \sim p(\mathcal{T})$: Sampling the i -th task from the task distribution.
- $\mathcal{D}_i = \{(s_{i,k}, a_{i,k}, s_{i,k+1}, r_{i,k})\}_{k=1}^{m_i}$: Dataset of transitions for task \mathcal{T}_i , with m_i transitions.
- $\mathcal{D} = \bigcup_{i=1}^N \mathcal{D}_i$: Combined dataset across N tasks.
- θ : Parameters of the abstract representation model or policy, depending on context.
- \mathcal{X} : Abstract (latent) state space.
- $P_i(x_{t+1} | x_t, a_t)$: Transition kernel in the latent space for task \mathcal{T}_i .
- $\pi : \mathcal{X} \rightarrow \mathcal{A}$: A policy mapping from abstract states to actions.
- π_θ : A parameterized policy with parameters θ .
- $\psi(s_t, z)$: State encoder mapping raw state s_t and context z to abstract state $x_t \in \mathcal{X}$.
- $\phi(h_{i,:t})$: Context encoder that produces a task embedding $z \in \mathcal{Z}$ from a trajectory history.
- \mathcal{Z} : Context (task embedding) space.
- $\mathcal{L}(f\theta; \mathcal{T}_i)$: Loss of the ARM on task \mathcal{T}_i .
- $\mathcal{L}(\theta; \mathcal{T})$: Loss of the ARM over a set of tasks \mathcal{T} . If clear from context, we write $\mathcal{L}(\theta)$ to denote the expected loss over $p(\mathcal{T})$.
- $\hat{\mathcal{L}}(\theta; \mathcal{T})$: Empirical loss of the ARM over a set of tasks \mathcal{T} .
- \mathcal{F} : Function class.
- Θ : Parameter space.
- $\mathbb{H}(X)$: Entropy of a random variable X .

B Definitions

Definition 1 We define a task as a Markov Decision Process (MDP) via the following 5-tuple

$$\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P_i, r_i, \gamma),$$

where:

- \mathcal{S} denotes the state space,
- \mathcal{A} denotes the action space,
- $P_i(s_{t+1} | s_t, a_t)$ is the transition kernel for the task $\mathcal{T}_i \sim p(\mathcal{T})$,
- $r_i(s, a)$ is the reward function, i.e. $r_i : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$,
- $\gamma \in [0, 1]$ is the discount factor.

Definition 2 We define four parameterized model components:

$$\psi(s_t, z; \theta_\psi) \rightarrow x_t, \quad x_{t+1} = x_t + \tau(x_t, a_t; \theta_\tau), \quad \hat{r}_t = \rho(x_t, a_t; \theta_\rho), \quad z = \phi(h_t; \theta_\phi),$$

where: - $\psi : \mathcal{S} \times \mathcal{Z} \rightarrow \mathcal{X}$ is a contextual encoder mapping high-dimensional states $s \in \mathcal{S} \subseteq \mathbb{R}^n$ and task embeddings $z \in \mathcal{Z} \subseteq \mathbb{R}^k$ to abstract states $x \in \mathcal{X} \subseteq \mathbb{R}^d$, $\tau : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$ is a transition model predicting latent state deltas, - $\rho : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ predicts rewards in abstract space, and - $\phi : (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^{t-1} \times \mathcal{S} \rightarrow \mathcal{Z}$ is a context encoder mapping past transitions to a task embedding, where parameters $\theta = (\theta_\psi, \theta_\tau, \theta_\rho, \theta_\phi)$.

We draw N i.i.d. training tasks $\mathcal{T}_i \sim p(\mathcal{T})$. Each task \mathcal{T}_i provides a dataset $\{(s_{i,t}, a_{i,t}, s_{i,t+1}, r_{i,t})\}_{t=1}^{m_i}$. Denote the combined data by

$$\mathcal{D} = \left\{ (\mathcal{T}_i, s_{i,t}, a_{i,t}, s_{i,t+1}, r_{i,t}) \mid i = 1, \dots, N; t = 1, \dots, m_i \right\}.$$

Definition 3 For each task \mathcal{T}_i , define the transition and reward losses:

$$\mathcal{L}_{\text{transition}}(\theta; \mathcal{T}_i) = \mathbb{E}_{(s_t, a_t, s_{t+1}, r_t) \sim \mathcal{D}_i} \left[\left\| \psi(s_{t+1}; \theta_\psi) - (\psi(s_t; \theta_\psi) + \tau(\psi(s_t; \theta_\psi), a_t; \theta_\tau)) \right\|^2 \right],$$

$$\mathcal{L}_{\text{reward}}(\theta; \mathcal{T}_i) = \mathbb{E}_{(s_t, a_t, s_{t+1}, r_t) \sim \mathcal{D}_i} \left[\left\| r_t - \rho(\psi(s_t; \theta_\psi), a_t; \theta_\rho) \right\|^2 \right].$$

For a scalar $\lambda > 0$, the total task-specific loss is

$$\mathcal{L}(\theta; \mathcal{T}_i) = \mathcal{L}_{\text{transition}}(\theta; \mathcal{T}_i) + \lambda \mathcal{L}_{\text{reward}}(\theta; \mathcal{T}_i).$$

We collect the parameters $\theta = (\theta_\psi, \theta_\tau, \theta_\rho)$.

To simplify notation, we denote the expected loss over all tasks as $\mathcal{L}(\theta)$ rather than $\mathcal{L}(\theta; \mathcal{T})$.

Definition 4 Given a Markov Decision Process $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P_i, r_i, \gamma)$, a policy is a mapping $\pi^{\mathcal{T}_i} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. We have $\pi^{\mathcal{T}_i}(a_t | s_t)$, which defines the probability of selecting action a_t when in state s_t . A deterministic policy is a special case where the mapping reduces to $\pi^{\mathcal{T}_i} : \mathcal{S} \rightarrow \mathcal{A}$, meaning that for each state s_t , there exists a single action $a = \pi^{\mathcal{T}_i}(s_t)$. A stochastic policy assigns a probability distribution over actions, meaning that for any state s_t , the probabilities over actions sum to 1.

Assumption 1 (Existence of Distribution) We assume there is a distribution \mathcal{T} over possible tasks indexed by \mathcal{T}_i . Each task \mathcal{T}_i defines a stochastic mapping from (s_t, a_t) to (s_{t+1}, r_t) .

Definition 5 Given a Markov Decision Process (MDP) $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P_i, r_i, \gamma)$ and a policy π , we define the value function of policy π as a mapping $V_\pi^{\mathcal{T}_i} : \mathcal{S} \rightarrow \mathbb{R}$ such that:

$$V_\pi^{\mathcal{T}_i}(s_t) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_i(s_t, a_t) \mid s_0 = s \right].$$

Here, the expectation is taken over the trajectory induced by following $\pi(a_t \mid s_t)$, where the state transitions follow the transition dynamics P_i .

Definition 6 Given a Markov Decision Process (MDP) $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P_i, r_i, \gamma)$ and a policy π , we define the state-action value function (Q-function) of policy π as a mapping $Q^{\pi, \mathcal{T}_i} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that:

$$Q^{\pi, \mathcal{T}_i}(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_i(s_t, a_t) \mid s_0 = s, a_0 = a \right].$$

Here, the expectation is taken over the trajectory where the initial action is a in state s , and subsequent actions are selected according to $\pi(a_t \mid s_t)$, while state transitions follow P_i .

C Propositions and Lemmas with Proofs

Proposition 1 (Entanglement Yields Lower Entropy) *Let \mathcal{T}_1 and \mathcal{T}_2 be two tasks such that for some state-action pair (s_t, a_t) , the transitions differ: $P_1(s_{t+1} | s_t, a_t) \neq P_2(s_{t+1} | s_t, a_t)$. Let $z = \phi(h_{i,:t}) \in \mathcal{Z}$ be a deterministic task embedding computed from transition history. Consider two model variants: (1) a task-state encoder, where $x_t = \psi(s_t, z)$, the transition prediction is given by $x_{t+1} \approx x_t + \tau(x_t, a_t)$ and the estimated reward $\hat{r}_t = \rho(x_t, a_t)$; and (2) a state encoder, where $x_t = \psi^*(s_t)$, $x_{t+1} \approx x_t + \tau^*(x_t, a_t, z)$ and the estimated reward $\hat{r}_t = \rho^*(x_t, a_t, z)$. Then, assuming ψ , ρ and τ as well as ψ^* , ρ^* and τ^* are deterministic and trained to perfectly fit the transition and reward losses, the total entropy of the representations satisfies:*

$$\mathbb{H}(\psi(s_t, z)) \leq \mathbb{H}((\psi^*(s_t), z)).$$

That is, the task-state version yields lower joint entropy due to resolving task ambiguity in the representation, resulting in simpler transitions and reward functions.

Proof 2 *Let s and z be random variables representing the state and task, respectively. Let $\psi : \mathcal{S} \times \mathcal{Z} \rightarrow \mathcal{X}$ be a task-aware encoder, and $\psi^* : \mathcal{S} \rightarrow \mathcal{X}$ a task-agnostic encoder.*

Assume there exists a deterministic function g such that

$$\psi(s, z) = g(\psi^*(s), z),$$

that is, the task-aware encoding can be obtained by transforming the joint variable $(\psi^(s), z)$. We refer to the left-hand side as the early-context encoding (direct mapping to the abstract state-task space) and to the right-hand side as the late-context encoding (post-hoc adjustment within abstract state space).*

Since g is a deterministic function applied to $(\psi^(s), z)$, the chain rule for entropy gives:*

$$\mathbb{H}(\psi^*(s), z) = \mathbb{H}(g(\psi^*(s), z)) + \mathbb{H}((\psi^*(s), z) | g(\psi^*(s), z)).$$

Because conditional entropy is nonnegative, we have

$$\mathbb{H}(g(\psi^*(s), z)) \leq \mathbb{H}(\psi^*(s), z).$$

Substituting $\psi(s, z) = g(\psi^(s), z)$, we obtain*

$$\mathbb{H}(\psi(s, z)) \leq \mathbb{H}(\psi^*(s), z).$$

Equality holds if and only if g is injective (i.e., information-preserving) on the support of $(\psi^(s), z)$. Otherwise, any many-to-one mapping g strictly reduces entropy.*

Hence, the early (task-state) representation cannot have higher entropy than the late (state + task) representation; it resolves task ambiguity directly in the encoded space, yielding lower representational complexity and simpler transition/reward dynamics. \square

Remark 1 (Conditions for strict inequality) *In practice, our architecture will often have a strictly lower entropy than downstream task-aliasing, because it operates directly in the state-task space, avoiding the need for downstream readjustment. To illustrate this, consider the following.*

We have

$$\mathbb{H}(\psi(s, z)) \leq \mathbb{H}(\psi^*(s), z).$$

Equality holds if and only if $\psi(s, z)$ and $(\psi^(s), z)$ are one-to-one, that is, if $\psi(s, z)$ can be perfectly recovered from $(\psi^*(s), z)$ and vice versa.*

As an example, consider two tasks \mathcal{T}_1 and \mathcal{T}_2 that differ in their transition dynamics from the same state-action pair $(s_1, a = \text{Right})$:

$$s_1 \xrightarrow{\mathcal{T}_1} s_2, \quad s_1 \xrightarrow{\mathcal{T}_2} s_3.$$

If the encoder ψ^* collapses the two follow-up states into the same abstract state, equality can still hold provided that the task embedding z can perfectly disambiguate this collapse.

However, we get a strict inequality once ψ^* induces a many-to-one mapping (state aliasing) and z cannot fully restore the lost information (e.g., due to overfitting on the training environments). In this case, the early task-aware representation $\psi(s, z)$ carries strictly less entropy, reflecting a more compact and disambiguated encoding of the task-conditioned dynamics.

Remark 2 (On Collapse and the Role of the Assumption and Regularizer) *The entropy inequality follows from subadditivity. However, it can be trivially satisfied if ψ collapses. However, the combination of the next-state encoder and the reward predictor will prevent this from happening. We additionally ensure that the representation encodes nontrivial information about the state.*

Practically, we enforce this via a regularizer:

$$\mathcal{L}(\theta)_{reg} = \exp \left(-C_d \left\| \psi(s^+, z; \theta_\psi) - \psi(s^-, z; \theta_\psi) \right\|^2 \right),$$

where s^+ and s^- are randomly sampled states. This encourages separation in latent space and mitigates representational collapse.

D Appendix D: Policy Algorithm

Algorithm 3 EMERALD Policy Learning

Require: Trained ARM f_θ , tasks $\mathcal{T}_i \sim p(\mathcal{T})$, learning rates α_π, α_V , batch size B , number of gradient updates n_{updates} , number of environment steps per rollout n_{rollout} , policy parameters θ_π and value/critic parameters θ_V , per-task replay buffers $\mathcal{D}_i \leftarrow \emptyset$, context horizon H

```

1: while not converged do                                     ▷ Policy learning loop
2:   for each task  $\mathcal{T}_i$  do                                     ▷ Data collection
3:     Reset temporary buffer:  $\mathcal{B} \leftarrow \emptyset$ 
4:     Reset context window:  $c^i \leftarrow \emptyset$ 
5:     for step = 1 to  $n_{\text{rollout}}$  do
6:       Get  $c^i$  from recent transitions; sample  $z \sim \phi(z \mid c^i; \theta_\phi)$ 
7:       Observe state  $s_t$ ; encode  $x_t = \psi(s_t, z; \theta_\psi)$ 
8:       Sample action  $a_t \sim \pi_{\theta_\pi}(\cdot \mid x_t)$ 
9:       Execute  $a_t$  in  $\mathcal{T}_i$ ; observe  $(s_{t+1}, r_t)$ 
10:      Encode next state  $x_{t+1}^{\text{true}} = \psi(s_{t+1}, z; \theta_\psi)$ 
11:      Store  $(x_t, a_t, r_t, x_{t+1}^{\text{true}})$  in  $\mathcal{B}$ 
12:      Update context:  $c^i \leftarrow c^i \cup \{(s_t, a_t, r_t, s_{t+1})\}$  until  $|c^i| \leq H$ 
13:    end for
14:    Update replay buffer:  $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \mathcal{B}$ 
15:  end for
16:  for update = 1 to  $n_{\text{updates}}$  do                               ▷ Optimization step
17:    for each task  $\mathcal{T}_i$  do
18:      Sample minibatch  $b^i = \{(s_t, a_t, r_t, s_{t+1})\}_{j=1}^B \sim \mathcal{D}_i$ 
19:      Compute task-specific policy loss  $L_{\text{policy}}^i$                 ▷ e.g. PPO, SAC, REINFORCE.
20:      Compute optional value loss  $L_{\pi\theta_{\text{value}}}^i$ 
21:    end for  $\theta_\pi \leftarrow \theta_\pi - \alpha_\pi \nabla_{\theta_\pi} \sum_i L_{\text{policy}}^i$ 
22:    Update value network:  $\theta_V \leftarrow \theta_V - \alpha_V \nabla_{\theta_V} \sum_i L_{\text{value}}^i$ 
23:  end for
24: end while
25: return Trained policy  $\pi_{\theta_\pi}$  and  $V_{\theta_V}$ 

```

E Policy Algorithm Details

The goal of reinforcement learning is to learn a policy that - on the basis of the current state - takes the action that maximizes the cumulative reward. In our setting, moreover, the “raw” input states are mapped to the learned latent space. Formally, the optimal policy satisfies:

$$\pi^* = \arg \max_{\pi} V_{\pi}(\psi(s_t; \theta_{\psi})), \quad \forall s \in \mathcal{S},$$

where $V_{\pi}(\psi(s_t; \theta_{\psi})) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r^{(e)}(x_t, a_t) \mid x_0 = x \right]$.

PPO PPO is a policy gradient method that improves stability by constraining policy updates. It maximizes the clipped surrogate objective:

$$L_{\pi}(\theta_{\pi}) = \mathbb{E}_t [\min(r_t(\theta_{\pi}) A_t, \text{clip}(r_t(\theta_{\pi}), 1 - \epsilon, 1 + \epsilon) A_t)],$$

where $r_t(\theta_{\pi}) = \frac{\pi_{\theta}(\psi(s_t; \theta_{\psi}))}{\pi_{\theta_{\text{old}}}(\psi(s_t; \theta_{\psi}))}$ is the probability ratio and A_t is the advantage function estimated via Generalized Advantage Estimation (GAE). The value function $V_{\phi}(\psi(s_t; \theta_{\psi}))$ is updated using:

$$L_V(\phi) = \mathbb{E}_t \left[(V_{\phi}(\psi(s_t; \theta_{\psi})) - V_t^{\text{target}})^2 \right].$$

The final objective combines policy loss, value loss, and entropy regularization:

$$L(\theta_{\pi}, \phi_{\pi}) = L(\theta_{\pi}) - c_1 L_V(\phi) + c_2 H(\pi_{\theta}),$$

where $H(\pi_{\theta})$ encourages exploration.

SAC SAC is an off-policy actor-critic method that optimizes not only for high expected returns but also for high-entropy policies, encouraging broader exploration.

The maximum-entropy RL objective is:

$$\max_{\pi} \sum_{t=0}^{\infty} \mathbb{E}_{s_t, a_t \sim \rho_{\pi}} \left[r^{(e)}(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t)) \right],$$

where \mathcal{H} denotes the entropy of the policy, and α is a temperature parameter that balances exploration vs. exploitation. In practice, α can be fixed or automatically tuned.

SAC maintains two Q-functions, Q_{θ_1} and Q_{θ_2} , to reduce positive bias in the target updates. Each Q_{θ_i} is learned by minimizing the mean-squared Bellman error:

$$L_{Q_i}(\theta_i) = \mathbb{E}_{(s_t, a_t) \sim D} \left[(Q_{\theta_i}(\psi(s_t; \theta_{\psi}), a_t) - y_t)^2 \right],$$

where D is the replay buffer, and

$$y_t = r^{(e)}(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[\min_{i=1,2} Q_{\theta_i^{\text{target}}}(\psi(s_{t+1}; \theta_{\psi}), a_{t+1}) - \alpha \log \pi_{\phi}(a_{t+1} \mid \psi(s_{t+1}; \theta_{\psi})) \right].$$

Here, $Q_{\theta_i^{\text{target}}}$ are target networks (periodically updated copies of Q_{θ_i}), and π_{ϕ} is the current policy.

The policy π_{ϕ} is updated by minimizing the Kullback–Leibler divergence between π_{ϕ} and an exponential of the Q-function. In practice, the policy loss is often written as:

$$L_{\pi}(\phi) = \mathbb{E}_{s_t \sim D} \left[\mathbb{E}_{a_t \sim \pi_{\phi}(\cdot \mid \psi(s_t; \theta_{\psi}))} \left[\alpha \log \pi_{\phi}(a_t \mid \psi(s_t; \theta_{\psi})) - Q_{\theta_i}(\psi(s_t; \theta_{\psi}), a_t) \right] \right].$$

Because there are two Q-functions, the minimum of Q_{θ_1} and Q_{θ_2} is typically used in practice to reduce overestimation bias.

To automatically adjust the trade-off between exploration and exploitation, SAC can learn the temperature α . The corresponding loss is:

$$L(\alpha) = \mathbb{E}_{s_t \sim D, a_t \sim \pi_\phi} \left[-\alpha \log \pi_\phi(a_t \mid \psi(s_t; \theta_\psi)) - \alpha \bar{H} \right],$$

where \bar{H} is a target entropy. Decreasing α reduces the entropy bonus and focuses more on reward maximization, while increasing α encourages broader exploration.

SAC alternates between:

1. Minimizing the Q-function losses $L_{Q_1}(\theta_1)$ and $L_{Q_2}(\theta_2)$.
2. Minimizing the policy loss $L_\pi(\phi)$.
3. (Optionally) Adjusting α by minimizing $L(\alpha)$.

Thus, the SAC objective can be summarized as:

$$L_{\text{SAC}} = \sum_{i=1}^2 L_{Q_i}(\theta_i) + L_\pi(\phi) + \begin{cases} L(\alpha) & (\text{if } \alpha \text{ is learned}), \\ 0 & (\text{otherwise}). \end{cases}$$

This encourages policies that achieve high returns while maintaining sufficient exploration through entropy maximization, all in the learned latent space $\psi(\cdot; \theta_\psi)$.

F Comparative Table Methodologies

Feature	EMERALD	CaDM	PEARL
Context Conditioning	Implicit	Explicit	Explicit
Latent Encoding	State-space	Context	Context
Encoder model needed	No	Yes	Yes
Policy Input	Task-aware abstract state space \mathcal{X}	\mathcal{S} and \mathcal{Z}	\mathcal{S} and \mathcal{Z}
Encoder model needed		Yes	Yes
Model Based or Model Free	Hybrid	Hybrid	Model-free

Table 6: Feature Comparison Between Different Methods. \mathcal{X} denotes a compressed representation of \mathcal{S} .

G Abstract Representation Model Architecture

We implement an abstract representation model designed for meta-reinforcement learning across a distribution of tasks $\mathcal{T}_i \sim p(\mathcal{T})$. The model maps observations $s_t \in \mathcal{S}$ and actions $a_t \in \mathcal{A}$ to latent abstract representations $x_t \in \mathcal{X}$, and infers task embeddings $z_i \in \mathcal{Z}$. Each component of the model is parametrized by submodules of θ .

State Encoder $\psi_{\theta_\psi} : \mathcal{S} \times \mathcal{Z} \rightarrow \mathcal{X}$ The encoder ψ maps observations s_t and task embeddings z_i to abstract states x_t . It includes skip connections to promote identity preservation and stabilize representation learning.

Transition Function $\tau_{\theta_\tau} : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$ The transition model predicts the next abstract state x_{t+1} given the current abstract state x_t and action a_t . It is implemented as a deep MLP and is context-agnostic, relying on the encoded state x_t to contain relevant task information.

Reward Predictor $\rho_{\theta_\rho} : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ The reward model estimates the immediate reward r_t based on the abstract state and action.

Task Encoder $\phi_{\theta_\phi} : h_{i:t} \rightarrow \mathcal{Z}$ An LSTM-based encoder ϕ that produces a task embedding $z_i \in \mathcal{Z}$ from the interaction history $h_{i:t} = (s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_t)$. This embedding captures task-specific dynamics for each \mathcal{T}_i .

Terminal Classifier $\beta_\theta : \mathcal{X} \times \mathcal{A} \rightarrow \{0, 1\}$ A binary classifier that predicts episode termination given the abstract state and action. Note used in current setup, but available in the implementation.

Discount Decoder An auxiliary module that estimates the discount factor $\gamma_t \in [0, 1]$ from (x_t, a_t) . This can be used during value prediction in planning.

Inference Procedure At each time step t , the model operates as follows:

1. Compute the task embedding: $z_i = \phi(h_{i:t})$
2. Encode current and next states: $x_t = \psi(s_t, z_i)$, $x_{t+1} = \psi(s_{t+1}, z_i)$
3. Predict the next abstract state: $\hat{x}_{t+1} = \tau(x_t, a_t)$
4. Predict the reward: $\hat{r}_t = \rho(x_t, a_t)$
5. Predict terminal status: $\hat{d}_t = \beta(x_t, a_t)$

The model is trained end-to-end using supervised objectives on transition, reward, and termination prediction. Training minimizes the expected loss across the task distribution:

$$\mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})} [\mathcal{L}_i(\theta)]$$

H Additional Experiments

Structured Representation Learning for Structurally Similar MDPs

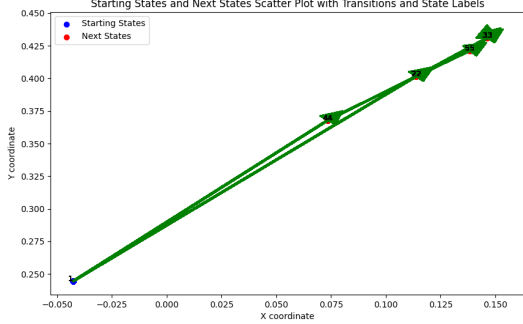


Figure 7: Reconstruction of an “arrow” from two MDPs.

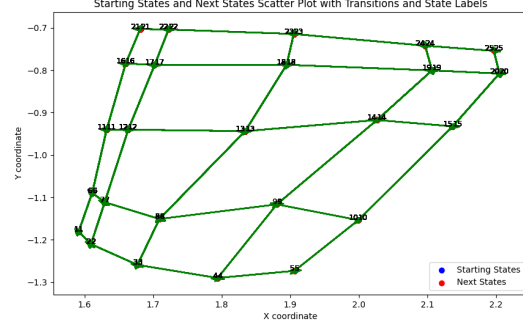


Figure 8: Reconstruction of a maze-like structure from three different structural MDPs

To evaluate EMERALD’s ability to produce meaningful and interpretable latent structures—a core property of abstract representation models—we adapt the visual experimental setups of [Van Driessel & François-Lavet \(2021\)](#) and [François-Lavet et al. \(2019\)](#) to a setting with multiple underlying MDPs (and thus multiple environments). Our objective is to assess how well EMERALD can learn a shared abstract representation across distinct environments.

We design two simple, interpretable MDP structures. In the first setting, we sample 300 transitions from two deterministic MDPs:

$$\mathcal{T}_1 : S_1 \rightarrow S_2 \rightarrow S_3, \quad \mathcal{T}_2 : S_1 \rightarrow S_4 \rightarrow S_2 \rightarrow S_5 \rightarrow S_3.$$

The transitions are embedded into a 2D latent space. As shown in Figure [7](#), EMERALD’s ARM effectively captures the shared structure between the two environments in this space.

In the second setting, we consider three distinct 5×5 maze-like environments. For each maze, we collect 10,000 transitions under a random policy, generating random walks. The maze layouts are generated randomly, with a fixed start state at $(0,0)$ and goal state at $(4,4)$. A visual overview of the layouts is provided in Appendix Figure [9](#). In the resulting 2D latent representation (Figure [8](#)), EMERALD successfully captures shared structural elements across the three mazes: similar states in different environments are closely aligned, illustrating the utility of the learned abstract representation.

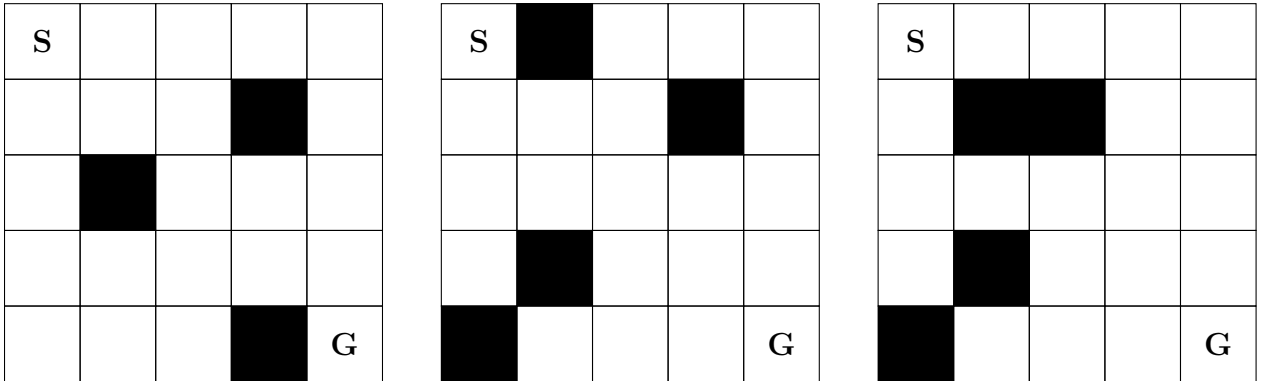


Figure 9: Three Maze Environments

Effect of Input Policy Type on Performance

For fairness, we pretrain the model with a random policy. However, to illustrate the effect when the offline data is of better quality (i.e. follows a non-random, but rather expert policy), we conducted a small ablation study with the PPO version of our model. The results are shown in Figure [7](#).

	Δ Improvement HalfCheetah (Vol.)	Δ Improvement HalfCheetah (Dir.)
EMERALD PPO	214.6 \pm 129.1	431.4 \pm 161.9

Table 7: Performance improvement (Δ Mean Return \pm Std.) of EMERALD PPO relative to baselines on HalfCheetah Volume and Direction tasks.

Additional Meta-Word Suite Experiments

Table 8: PEARL (SAC) baseline on Meta-World ML1 tasks.

Method	door-open	basketball	window-open	pick-place	button-topdown
PEARL (SAC)	0.0 ± 0.0	0.0 ± 0.0	36 ± 1.1	0.0 ± 0.0	43 ± 2.5

Table 9: PEARL (SAC) baseline on Meta-World ML10.

Method	Average (Train)	Average (Test)
PEARL (SAC)	23.1 ± 1.5	13.4 ± 1.5

I Abstract Model Hyperparameters

Table 10: Training Model Parameters

Parameter	HalfCheetah (Vol)	HalfCheetah (Dir)	Pendulum (Length)	CartPole (Length)
Weight Decay	0.00001	0.00001	0.00001	0.00001
Hidden Dim	128	128	32	32
Latent Dim	10	10	2	8
Model Learning Rate	0.0001	0.0001	0.0001	0.0001
Hidden Dim ρ	128	128	8	64
Output Dim ρ	1	1	1	1
Context Dim	4	4	12	4
Environment ID	HalfCheetah	HalfCheetah	Pendulum	CartPole
# Envs	5	2	11	5

Table 11: Training Agent Parameters

Parameter	HalfCheetah (Vol)	HalfCheetah (Dir)	Pendulum (Length)	CartPole (Length)
Policy Network LR	0.0001	0.0001	0.0001	0.0003
Policy Hidden Dim	64	64	64	64
Gamma	0.95	0.95	0.99	0.99
Reward Weight	1	1	1	1
τ Loss Weight	1	1	1	1
Regularization Weight	0.01	0.01	0.01	0.01
Train Epochs	300	300	3000	50
Max Iterations	1000	1000	1000	200
History Length	100	100	100	100
Max Episode Steps	1000	1000	200	200
Policy Iterations Learn	5,000,000	5,000,000	2,000,00	2,000,00
Samples per Task (ARM Training)	100,000	100,000	\sim 4,500	10,000
Batch Size	256	256	256	256

J Training Details for the Different Environments

General Details We follow the experimental setup of Lee et al. (2020), reproducing their results with the same environment modifications using the code base available at <https://github.com/younggyoseo/CaDM>. For each environment, we train on N training configurations and evaluate on several out-of-distribution (OOD) test environments. Results are averaged over 8 runs with different random seeds. The setup is illustrated in Figure 10. While the conceptual design is the same, we modified the implementation to be compatible with OpenAI Gymnasium v5, which has built-in support for MuJoCo-based environments.

We implemented the EMERALD abstract representation model in PyTorch (Python 3.9). RL agents were implemented using Stable Baselines3 (SB3) (Raffin et al., 2021) and CleanRL (Huang et al., 2022). Experiments were conducted on an Apple M4 processor with 10 CPU cores and 10 GPU cores. All results are reported as the mean over 4 runs with different random seeds.

K Implementational Considerations

OOD Comparative Experiments We sample offline data using a random policy to ensure that our model is not positively biased by policy optimization. This setup arguably places our model at a disadvantage compared to methods that benefit from task- or goal-directed data collection. We sample at most 1M training data points, divided by the number of maximum steps per episode and the number of available training environments. Full training details are provided in Table 1. For baselines, we re-ran the publicly available code from <https://github.com/younggyoseo/CaDM> without any modifications.

ID Comparative Experiments We compare in-distribution performance against existing methods. For VariBAD and PEARL, we report values as recorded in the original papers (Zintgraf et al., 2021).

Task-Diversity Experiment For each environment, we train models (ARMs) under two configurations, averaging results over five runs, while keeping the total number of environment interactions fixed: 200K for CartPole and 3M for HalfCheetah. In Configuration 1, each model is trained on a single task using the full sample budget. In Configuration 2, the same total number of samples is distributed across five tasks—i.e., 200K / 5 for CartPole and 3M / 5 for HalfCheetah—reducing per-task data while increasing task diversity.

Ablation Study We evaluate the model under four different configurations:

1. 100K samples with a context encoder on CartPole (Volume).
2. 10K samples without a context encoder on CartPole.
3. 100K samples with a context encoder on HalfCheetah (Volume).
4. 10K samples without a context encoder on CartPole.

We trained each configuration for up to 100 epochs to analyze the effect of model training on downstream performance. After pretraining, we evaluate by fine-tuning the policy on a single target environment: CartPole with pole length 1, or HalfCheetah (Volume) with scaling set to 1. We fine-tune for 10K steps in CartPole and 100K steps in HalfCheetah. We deliberately keep fine-tuning short to avoid masking the effects of model pretraining—since longer training might allow some seeds to recover good performance regardless of initialization, thus obscuring the contribution of the model.

Latent Trajectory Analysis For each environment, we extract the first 50 latent transitions from both training and test episodes. In CartPole and HalfCheetah (Volume), this yields 7 distinct clusters (5 from training environments and 2 from test environments), while in Pendulum (Length), we obtain 13 clusters (11 training, 2 test). By design, the initial training transitions are collected from largely untrained policies, resulting in greater variability within the corresponding latent clusters. In contrast, test clusters reflect more stable behavior, as they are derived from trained policies. To visualize the latent space structure, we apply principal component analysis (PCA) to reduce the dimensionality and plot the trajectories in 3D.

This form of latent space analysis has been used in prior work to examine representation quality and clustering behavior in RL and meta-RL models (e.g., Rakelly et al., 2019; Zintgraf et al., 2021).

L Environment-Variations

We follow the general structure of [Lee et al. \(2020\)](#).

CartPole The CartPole task involves balancing a pole on a moving cart by applying discrete forces to the cart.

- **Observation:** $(x_t, \dot{x}_t, \theta_t, \dot{\theta}_t)$, where x is the cart’s position and θ is the angle of the pole from the vertical.
- **Action:** $\{0, 1\}$, where 0 pushes the cart left and 1 pushes it right.
- **Reward:**

$$r_t = \mathbb{1}_{\{|x_{t+1}| < 2.4 \wedge |\theta_{t+1}| < \frac{14\pi}{360}\}}$$

A reward of 1 is given if the cart remains within ± 2.4 and the pole within $\pm 14^\circ$; otherwise 0.

- **Modifications:** Push force f and pole length l are modifiable.

Pendulum The Pendulum task aims to swing a rod upright using continuous torque.

- **Observation:** $(\cos \theta, \sin \theta, \dot{\theta})$, where $\theta \in [-\pi, \pi]$, with $\theta = 0$ being upright.
- **Action:** $a \in [-2.0, 2.0]$, representing continuous torque.
- **Reward:**

$$r_t = -(\theta_t^2 + 0.1\dot{\theta}_t^2 + 0.001a_t^2)$$

Penalizes deviation from the upright, angular velocity, and torque magnitude.

- **Modifications:** Pendulum mass m and length l can be changed.

HalfCheetah (Volume) The HalfCheetah is a planar robot designed to learn fast and energy-efficient running.

- **Observation:** A 20-dimensional vector including joint positions/velocities, root joint state (excluding x), and torso center-of-mass velocity.
- **Action:** $a \in [-1.0, 1.0]^6$, torques applied at six joints.
- **Reward:**

$$r_t = \dot{x}_{\text{torso},t} - 0.05\|a_t\|^2$$

Rewards forward velocity and penalizes control effort.

- **Modifications:** Scale the mass of every rigid link by a fixed factor m and the damping of every joint by a fixed factor d .

HalfCheetah (Direction) HalfCheetah is a planar robot with 9 links and 6 actuators, designed to move along the x-axis. The agent is trained to move either forward or backward depending on the task direction.

- **Observation:** A 17-dimensional vector containing positional and velocity information of the joints and torso.
- **Action:** $a \in [-1, 1]^6$, representing the torques applied at each joint.

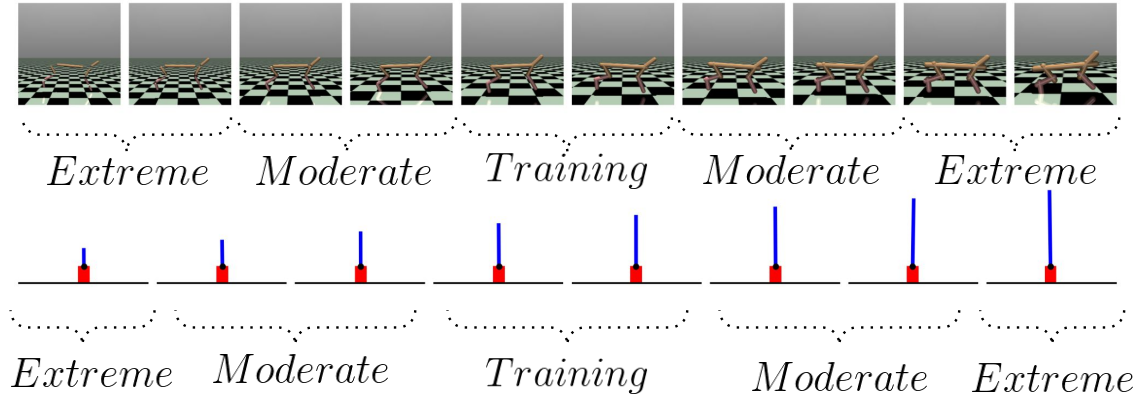


Figure 10: Illustration of the Experimental Setup. We train on N intermediate training environments. At test time, we test the model’s performance in a test regime that is moderately different and one that differs extremely from the training environment.

- **Reward:**

$$r_t = d \cdot \dot{x}_t - 0.05 \|a_t\|^2$$

where $d \in \{-1, 1\}$ specifies the target direction (backward or forward). The reward encourages movement in the desired direction and penalizes large control inputs.

- **Modifications:**

1. Scale the mass of each link by a factor m
2. Scale joint damping by a factor d

Table 12: Environment Modifications used for Comparative our Experiments (following Lee et al. (2020)).

Environment	Training	Test (Moderate)	Test (Extreme)	Episode Length
CartPole	$f \in \{5.0, 6.0, 7.0, 8.0, 9.0, 10.0\}$			200
	$11.0, 12.0, 13.0, 14.0, 15.0\}$	$f \in \{3.0, 3.5, 16.5, 17.0\}$	$f \in \{2.0, 2.5, 17.5, 18.0\}$	
	$l \in \{0.40, 0.45, 0.50, 0.55, 0.60\}$	$l \in \{0.25, 0.30, 0.70, 0.75\}$	$l \in \{0.15, 0.20, 0.80, 0.85\}$	
Pendulum	$m \in \{0.75, 0.80, 0.85, 0.90, 0.95, 1.00, 1.05, 1.10, 1.15, 1.20, 1.25\}$	$m \in \{0.50, 0.70, 1.30, 1.50\}$	$m \in \{0.20, 0.40, 1.60, 1.80\}$	200
	$l \in \{0.75, 0.80, 0.85, 0.90, 0.95, 1.0, 1.05, 1.10, 1.15, 1.20, 1.25\}$			
		$l \in \{0.50, 0.70, 1.30, 1.50\}$	$l \in \{0.20, 0.40, 1.60, 1.80\}$	
HalfCheetah (Volume)	$m \in \{0.75, 0.85, 1.00, 1.15, 1.25\}$	$m \in \{0.40, 0.50, 1.50, 1.60\}$	$m \in \{0.20, 0.30, 1.70, 1.80\}$	1000
	$d \in \{0.75, 0.85, 1.00, 1.15, 1.25\}$	$d \in \{0.40, 0.50, 1.50, 1.60\}$	$d \in \{0.20, 0.30, 1.70, 1.80\}$	
HalfCheetah (Direction)	$d \in \{-1.0, 1.0\}$	$d \in \{-1.0, 1.0\}$	$d \in \{-1.0, 1.0\}$	1000

M Budget Allocation

Environment	Total Number of Transitions	ARM Training	Policy Training
HalfCheetah	5M	500K	4.5M
Pendulum	500K	~50K	~450K
Cartpole	500K	50K	450K

Table 13: Sample breakdown across training phases for each environment.

N Additional Considerations regarding Baselines

Hyperparameter Tuning: We aimed to reproduce the baselines as faithfully as possible by relying directly on the authors’ publicly available implementations. For each method, we used the original source code for the networks and policies whenever feasible. To ensure fairness in comparison, we applied the same hyperparameter tuning protocol to both our method and the baselines. Specifically, for every method we selected the best-performing discount factor $\lambda \in 0.9, 0.95, 0.99$ and learning rate $\in 10^{-3}, 10^{-4}, 10^{-5}$.

Observed Discrepancies: We also verified the extent to which our experimental results deviate from existing literature. For Setting 1, our findings largely align with prior work (most notably Lee et al. (2020)). The same holds for the HalfCheetah (Direction) experiment in Table 2, although we were only able to validate the VariBAD (PPO) results using Zintgraf et al. (2021) and could not find any reported performance for VariBAD (SAC) in this setting. In Settings 2 and 3, the PPO-based results are generally consistent with the values reported in the original papers (e.g., Beck et al. (2023); Ni et al. (2021)). A notable exception is the VI-HN (PPO) result from Beck et al. (2023), which in our experiments performs substantially better than RNN-HN. Finally, most hypernetwork-based SAC approaches perform significantly worse than their PPO counterparts in the MetaWorld settings. We could not find any prior literature demonstrating strong SAC performance in these settings, which may indicate that others have attempted SAC without success.

Additional Considerations: We further highlight the following implementational challenges: (1) It was not always possible to run the source code *as is*; in several cases we had to modify or debug the implementations to make them functional. This was particularly true for RMF-RL Ni et al. (2021) and RNN-HN Beck et al. (2023). (2) Another difficulty and potential source of variation stems from the need to adapt the baselines to our Gymnasium environment version (1.1.1). Most original implementations rely on earlier versions of Gym and/or MuJoCo. Nevertheless, our VI-HN (PPO) and RNN-HN (PPO) results closely match those reported in Beck et al. (2023) (see Section 8). (3) Finally, we maintain a strict distinction between PPO-based and SAC-based methods. However, many official repositories provide a complete implementation for only one of these families (usually only the on-policy PPO). For example, the original VariBAD codebase (<https://github.com/lmzintgraf/varibad>) includes PPO and A2C implementations but no SAC version. For this reason, we implemented most of the SAC variants ourselves. While we managed to do so within our available time and resources, we emphasize that integrating SAC into PPO-oriented codebases could likely be improved, though doing so would require methodological design changes beyond the scope of this work. Still, we did test whether SAC-based approaches such as VI-HN (SAC) worked in single-setting problems and found that we obtained reasonable results there, indicating that there are almost certainly no large bugs in the code and that breakdowns only occur when the method needs to generalize to the meta-RL setting.

O Averaged Performance for SAC and PPO

The table below show the avrage performance over *both* PPO and SAC.

Table 14: Averaged performance between PPO and SAC versions (Mean Return).

	CartPole (Lengths)		Pendulum (Lengths)		HalfCheetah (Volume)	
	Moderate	Extreme	Moderate	Extreme	Moderate	Extreme
VariBAD (Avg.)	196.9	190.8	-659.9	-918.5	2359.8	1521.6
RMF-RL (Avg.)	198.35	194.55	-637.45	-978.9	3361.3	1589.6
VI-HN (Avg.)	198.45	195.35	-539.25	-814.95	3123.75	1956.1
RNN-HN (Avg.)	197.5	194.2	-591.65	-744.55	2410.3	1761.75
EMERALD (Avg.)	196.85	197.75	-255.35	-531.55	4021.8	2336.6

P Learning Curves

Performance on CartPole Train

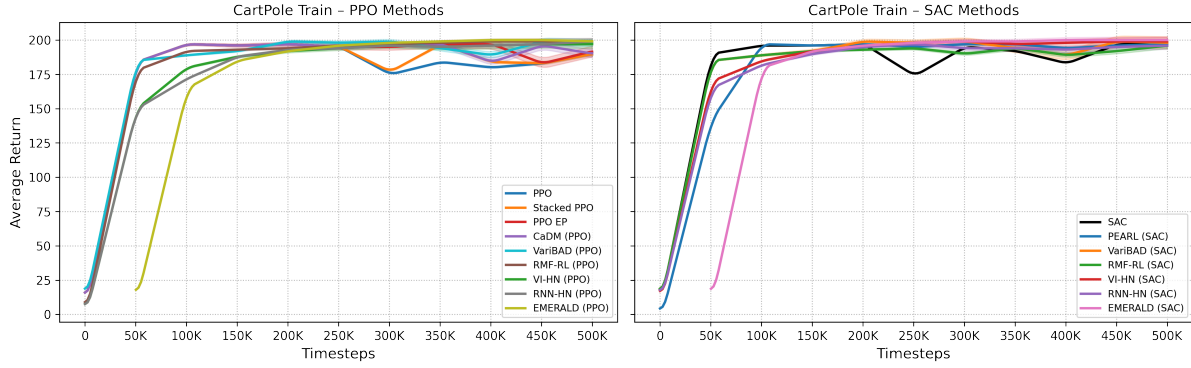


Figure 11: CartPole: Training (rolling averages over 8 independent runs)

Performance on CartPole Moderate

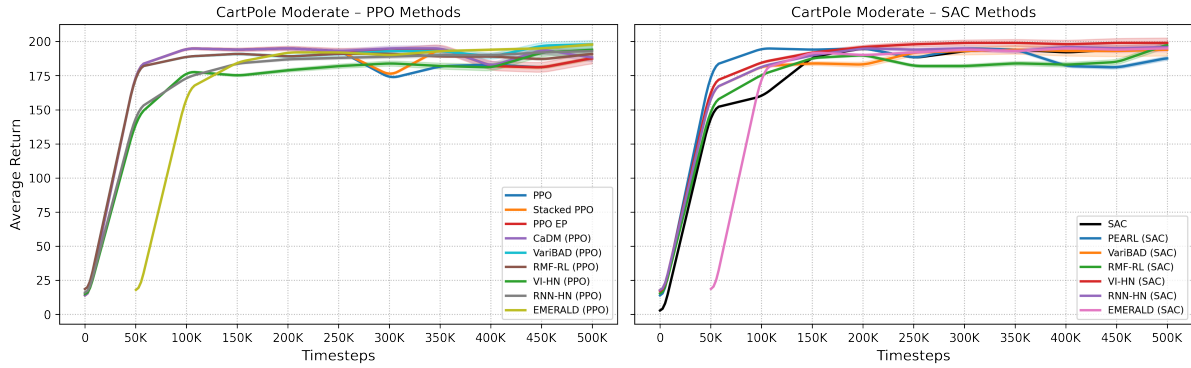


Figure 12: CartPole: Moderate (rolling averages over 8 independent runs)

Performance on CartPole Extreme

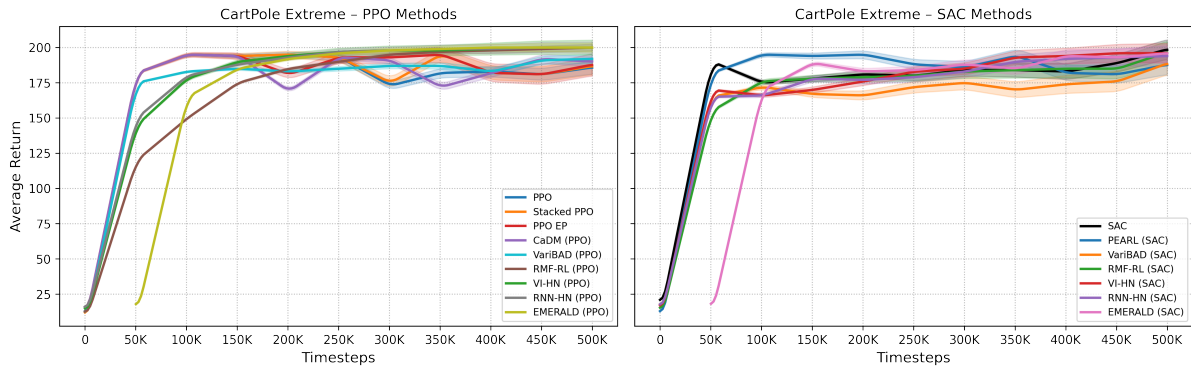


Figure 13: CartPole: Extreme (rolling averages over 8 independent runs)

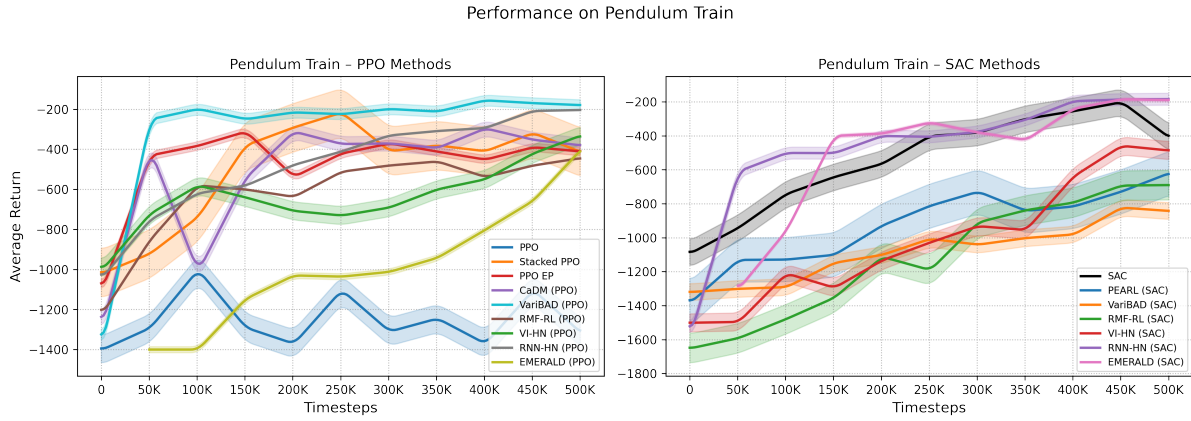


Figure 14: Pendulum: Training (rolling averages over 8 independent runs)

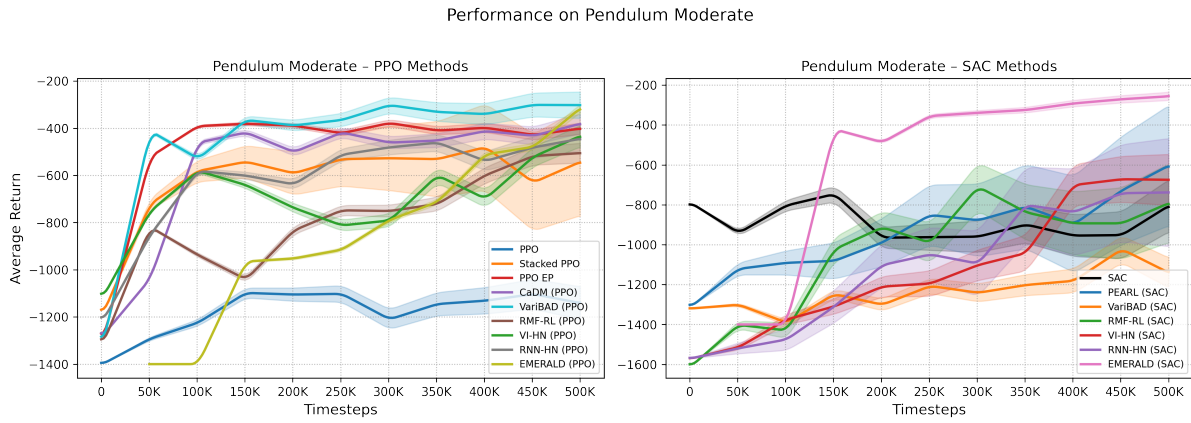


Figure 15: Pendulum: Moderate (rolling averages over 8 independent runs)

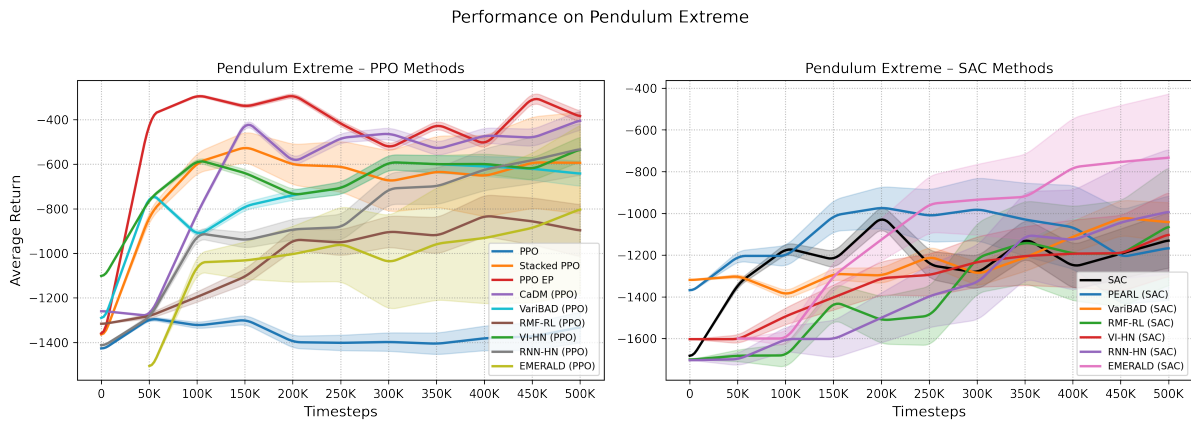


Figure 16: Pendulum: Extreme (rolling averages over 8 independent runs)

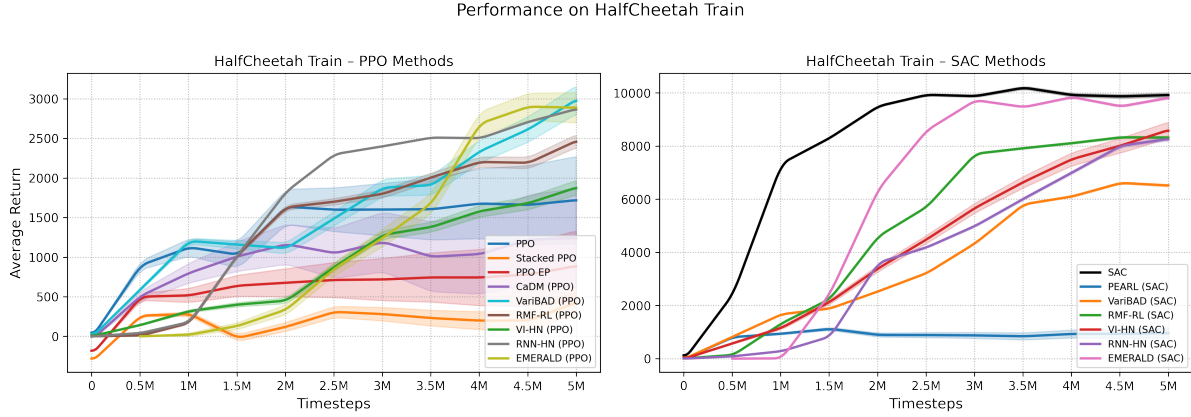


Figure 17: HalfCheetah (Volume): Training (rolling averages over 8 independent runs)

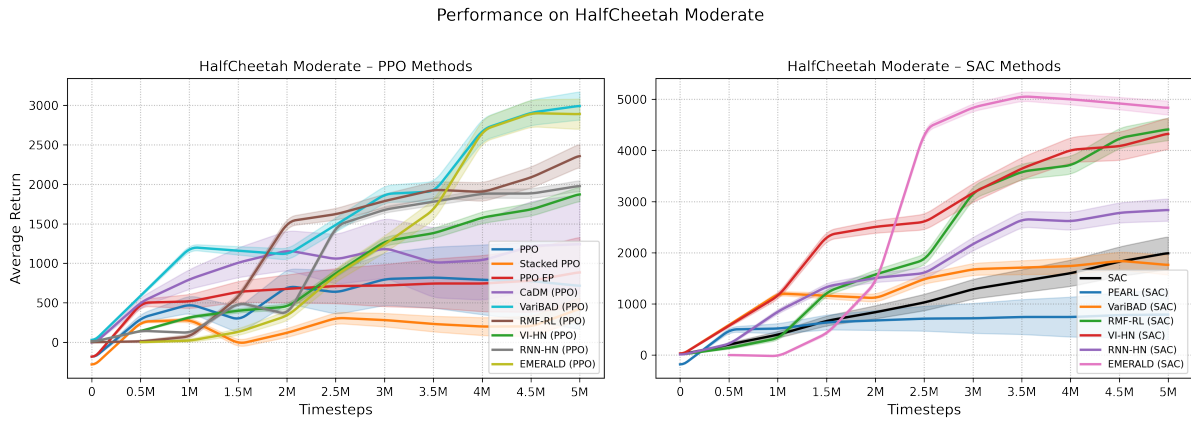


Figure 18: HalfCheetah (Volume): Moderate (rolling averages over 8 independent runs)

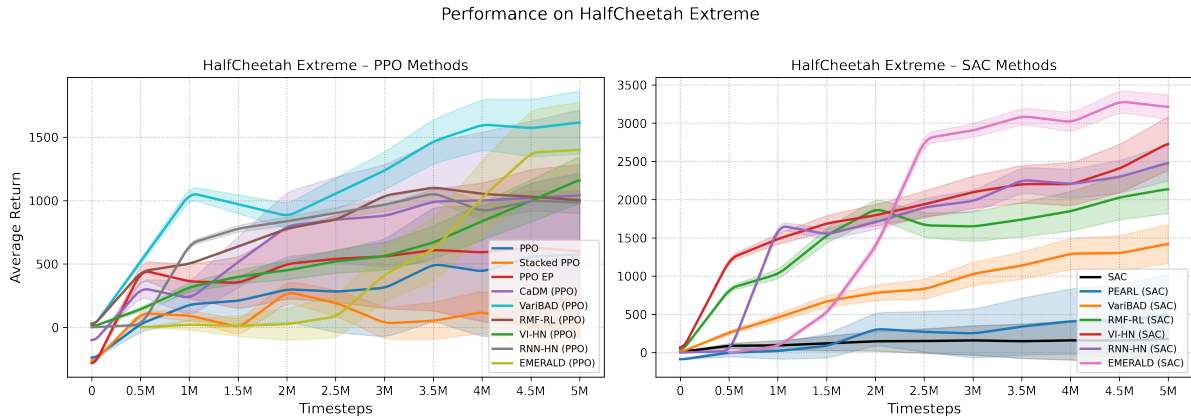


Figure 19: HalfCheetah (Volume): Extreme (rolling averages over 8 independent runs)

Q Latent Space Visualizations

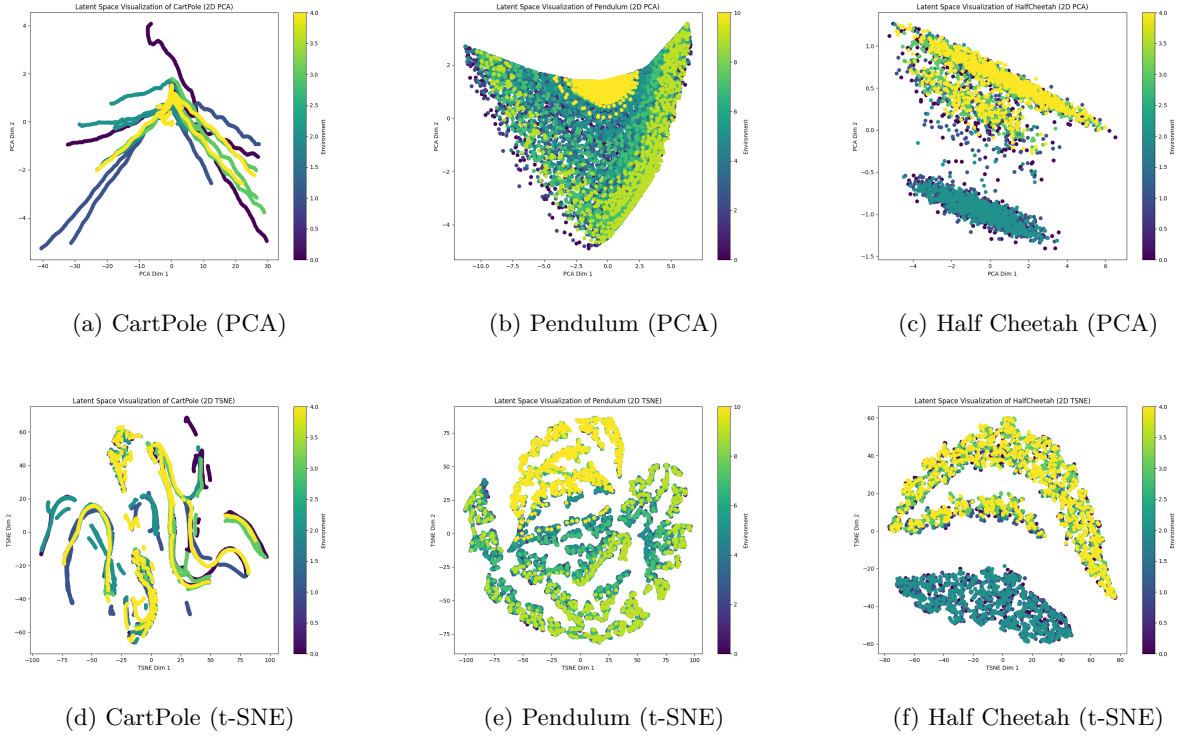


Figure 20: Latent space visualizations across environments using PCA (top row) and t-SNE (bottom row).