Agnostic Architecture for Heterogeneous Multi-Environment Reinforcement Learning

Kukjin Kim
Korea University
kukjinkim@korea.ac.kr

Changhee Joo*
Korea University
changhee@korea.ac.kr

Abstract

In new environments, training a Reinforcement Learning (RL) agent from scratch can prove to be inefficient. The computational and temporal costs can be significantly reduced if the agent can learn across diverse environments and effectively perform transfer learning. However, achieving learning across multiple environments is challenging due to the varying state and action spaces inherent in different RL problems. Padding or naive parameter-sharing with environment-specific layers for different state-action spaces are possible solutions for multi-environment training. However, they can be less scalable when training for new environments. In this work, we present a flexible and environment-agnostic architecture designed for learning across multiple environments simultaneously without padding or environment-specific layers, while enabling transfer learning for new environments. We also propose training algorithms for this architecture to enable both online and offline RL. Our experiments demonstrate that multi-environment training with one agent is possible in heterogeneous environments and parameter-sharing with environment-specific layers is not effective in transfer learning.

1 Introduction

In recent years, foundation models in computer vision and natural language processing have been an irresistible trend. After pretraining a large-scale model on many datasets, this model can be adapted to downstream tasks by fine-tuning or zero-shot inference with contextual input (Bommasani et al., 2021). This paradigm is also extended actively to decision-making by utilizing pretrained representations for planning and reinforcement learning (RL) (Yang et al., 2023). However, unlike the vision and language domains, many RL problems are defined by their distinct Markov decision process (MDP). The components of MDP can be different because the state, observation, and action spaces depend on each problem. So, specialized techniques are required to train a large-scale neural network for multiple RL problems because of an incompatibility issue caused by fixed-sized input and output of the neural network. To address this issue, one can utilize padding and masking to data or use parameter sharing with the environment-specific encoders and decoders. In the continuous control domain, some related works (Huang et al., 2020; Kurin et al., 2020; Hong et al., 2021; Gupta et al., 2021; Trabucco et al., 2022; Furuta et al., 2022; Shridhar et al., 2023; Xiong et al., 2023) used graph neural networks, Transformer, and Hypernetwork architecture with padding or environment-specific layers to solve the incompatibility issue. Other works (Reed et al., 2022; Lee et al., 2022; Bousmalis et al., 2023) tried to pretrain large-scale decision transformer model (DT, Chen et al., 2021) in multiple environments by offline supervised learning with environment-specific embeddings and tokenization. Due to the excellent representational capability of the Transformer, it is being incorporated into many reinforcement learning works, and DT appears to be the suitable model for foundational RL agents. However, there are several issues with using the DT-based policy

*Corresponding author

Foundation Models for Decision Making Workshop at Neural Information Processing Systems, 2023

in the online RL setting.

(1) Costs of computation and memory. In the Transformer architecture (Vaswani et al., 2017), the computational complexity of self-attention operation increases quadratically with the length of the sequence $O(L^2)$. If the context length of DT for taking trajectory grows largely, the memory and computation cost of the DT-based policy will be prohibitive. The embedded system with limited network resources and low computational capacity will not be able to equip this model.

(2) Interaction with environment. For training the policy and value function in the online setting, the policy must collect data through interactions with the environment. When parametrizing the policy with a large-scale neural network, the speed of data collection and policy training slows down due to the long inference time from the issue (1), and this becomes severe as the number of environments increases. To avoid these problems, we can introduce small environment-specific encoders and decoders. But, whenever a new environment is added, these layers must be added and learned from scratch. The number of parameters will be increased linearly with respect to the number of environments, and the interaction with multiple environments can't be batch-processed.

(3) Data processing. For training the DT-based policies, tokenization and padding are required, and masking is applied for loss computation. Padding and masking increase memory usage and computational load during training. For instance, in the gym's CartPole and Humanoid environments, the states are 4-dimensional and 376-dimensional vectors, respectively. To simultaneously train on both environments, one could pad the state from CartPole with 372 zeros to make them of equal length or embed them to more high-dimensional vectors. However, this is inefficient in terms of memory usage. On the one hand, in the continuous control domain, discretizing continuous values and tokenizing them could lead to convergence to a suboptimal policy, potentially resulting in decreased performance.

In this work, we propose new policy and value architectures that combine standard neural network building blocks to avoid the above issues and decentralized distributed algorithms to train an agent in multiple environments. Learning heterogeneous problems that have entirely different state spaces and action spaces is possible without padding or environment-specific layers by using these architectures. Our architecture treats state and action as sequences, not requiring an additional environment wrapper for preprocessing observation and action. We parametrize the policy and value function by using a sequence-to-sequence model. It can accept various inputs and process continuous and discrete outputs by separating the decoders for each domain. This approach minimizes the usage of environment-specific layers, thereby not requiring the new layers when new environments come, and it also eliminates data processing related to padding and masking. We show that our architecture is useful in online RL and offline RL. It can learn heterogeneous multiple environments simultaneously. In some cases, the performances are close to an agent learned in a single environment and better than environment-specific architecture when conducting transfer learning to new environments.

2 Preliminaries

Multi-Environment Reinforcement Learning. Consider the set of N decision-making problems $\{\mathcal{M}_i\}_{i=1}^N$. Each problem can be defined as Markov decision process (MDP) $\mathcal{M}_i = \{\mathcal{S}_i, \mathcal{A}_i, R_i, P_i, p_i(s_0)\}$ where $\mathcal{S}_i, \mathcal{A}_i$ are sets of states and actions, $R_i : \mathcal{S}_i \times \mathcal{A}_i \to \mathbb{R}$ is the reward function, $P_i : \mathcal{S}_i \times \mathcal{A}_i \times \mathcal{S}_i \to \mathbb{R}$ is the transition probability distribution, and $p_i(s_0) : \mathcal{S}_i \to \mathbb{R}$ is the initial state distribution. The i-th environment determines each i-th component. Multi-environment RL aims to find the optimal policy π^* that maximizes the expected discounted return in all environments. The objective to be optimized in N environments is defined as Equation (1).

$$J(\pi) := \sum_{i}^{N} J_i(\pi), \quad J_i(\pi) = \mathbb{E}_{s_0 \sim p_i(s_0)}\big[\sum_{t=0}^{T} \gamma^t R_i(s_t, a_t)\big] \tag{1}$$

Note that the policy π is shared for multiple environments. In multi-task and meta RL settings, the task structure is shared; the state and action spaces are the same across tasks. However, multi-environment RL problems have heterogeneous spaces of state and action. So if $\{\mathcal{M}_i\}_{i=1}^N$ contains the discrete and continuous action spaces, the agent is required to be parametrized with multi-modal policy and value functions. The optimization of the shared policy in the multi-environment setting can also be viewed as multi-objective optimization, and the solution of the policy will be converged to Pareto optimal (Sener and Koltun, 2018).

Promixal Policy Optimization. (PPO) PPO (Schulman et al., 2017) is a standard technique to find optimal policy π^* with on-policy training. Consider the parametrized policy π_{θ_π} and value function

V_{θ_V} and π_{old} is the policy used for collecting trajectories. $r_t(\theta_\pi) = \frac{\pi_{\theta_\pi}(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ is the probability ratio to be clipped by range ϵ. $\hat{A}_t = R_t - \hat{V}_t$ at timestep t is the advantage estimator computed from collected experiences. The ratio clipping PPO objective is defined as follows. This can be extended easily for multi-task settings (MT-PPO, Yu et al., 2020)

$$J(\pi) = \hat{\mathbb{E}}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\quad \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\Big)\Big] \tag{2}$$

.

Implicit Q Learning. (IQL) IQL (Kostrikov et al., 2021) is a simple and powerful offline RL method that avoids evaluating actions out of a given dataset. This algorithm treats the state value function as the random variable for implicitly approximating the policy improvement step. It alternates between fitting the upper expectile value function by Equation (3) and backing it up into an action-value function by Equation (4). The policy is made by advantage-weighted behavioral cloning by Equation (5) without querying out-of-sample actions.

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))]. \tag{3}$$

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2]. \tag{4}$$

$$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[\exp(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s)))\log\pi_\phi(a|s)]. \tag{5}$$

Structured State Space Sequence Model. (S4) S4 (Gu et al., 2021a) is a general-purpose sequence model that can deal with long-range dependencies. This model can solve long sequence problems that neither RNNs nor Transformers can address and offer faster inference speeds. They formalize the long-range sequence modeling as an online function approximation problem. S4 is based on the state-space model (SSM), which maps function-to-function for continuous input signals. The Equations of the SSM are defined as follows $\dot{x} = \boldsymbol{A}x(t) + \boldsymbol{B}u(t),\ y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$. For dealing with discrete sequences, it should be discretized with step size Δ, which means a resolution of the input. The Equations of discrete-time SSMs are as follows, assuming $\boldsymbol{D}u(t) = 0$ because it can be viewed as a skip connection.

$$\begin{aligned} \overline{\boldsymbol{A}} &= (\boldsymbol{I} - \Delta/2\cdot\boldsymbol{A})^{-1}(\boldsymbol{I} + \Delta/2\cdot\boldsymbol{A}) & x_k &= \overline{\boldsymbol{A}}x_{k-1} + \overline{\boldsymbol{B}}u_k \\ \overline{\boldsymbol{B}} &= (\boldsymbol{I} - \Delta/2\cdot\boldsymbol{A})^{-1}\Delta\boldsymbol{B}, \quad \overline{\boldsymbol{C}} = \boldsymbol{C} & y_k &= \overline{\boldsymbol{C}}x_k \end{aligned} \tag{6}$$

By initializing the internal state matrix $\overline{\boldsymbol{A}}$ as a structured HiPPO matrix, it can capture the long-range dependencies (Gu et al., 2020). The hidden state vector x_k is a compressed representation of the cumulative long-range history. It can be computed autoregressively like the RNN, allowing memory-efficient inference. From the linear state space layer (LSSL, Gu et al., 2021b), they show that the Equation (7) for all output sequence $\boldsymbol{y} = y_{1:L}$ from input sequence $\boldsymbol{u} = u_{1:L}$ can be computed efficiently by parallel convolution where $\overline{\boldsymbol{A}}$ is fixed HiPPO matrix and $\overline{\boldsymbol{K}}$ is called SSM convolution kernel or filter.

$$\begin{aligned} y_k &= \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots\overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k \\ \boldsymbol{y} &= \overline{\boldsymbol{K}} * \boldsymbol{u}, \quad \overline{\boldsymbol{K}} \in \mathbb{R}^L = (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, ..., \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}) \end{aligned} \tag{7}$$

However, the LSSL has some disadvantages. The computation of the power of the state matrix is numerically unstable and requires significant memory usage. They address these issues by parametrizing $\overline{\boldsymbol{A}}$ as a Diagonal Plus Low-Rank matrix, enabling the fast and stable calculation of the SSM convolution kernel. We will use the S4 as a dynamic feature extractor for policy and value functions.

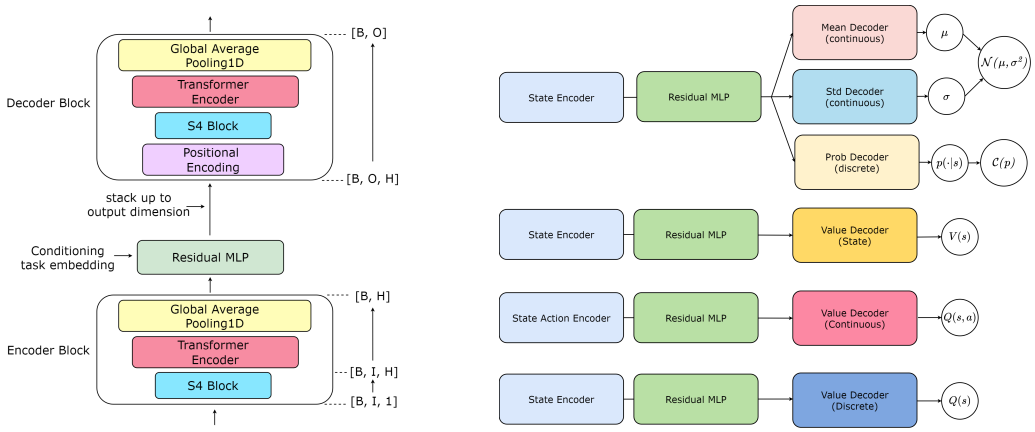
3 Method

3.1 Arbitrary 1D Input-Output Agent

Basic Building Block. Our architecture is inspired by (Kaiser et al., 2017; Metz et al., 2017; Jaegle et al., 2021; Trabucco et al., 2022). In Figure 1a, the basic building block of policy and value networks is composed of S4, Transformer, GlobalAveragePooling (GAP), Residual MLP, and Positional Encoding. To handle various state and action spaces in a multi-environment setting, we

need a structure that can process arbitrary input and output. For this purpose, We substitute the linear embedding layer in front of the TransformerEncoder to the S4 layer. The role of the S4 layer can be viewed as a dynamic feature embedding layer to take various length inputs. In the encoder block, the S4 layer takes the vector input and expands it to the 1-column matrix by considering the input dimension as sequence length and then transforms the sequence of scalars to a sequence of vectors. This matrix feeds to the TransformerEncoder and is averaged on the input dimension by the GAP. In the middle Residual MLP, we feed the task embedding vector and sum it with a hidden vector from the encoder. On the other hand, for the decoder block, we stack the same hidden vector from the previous block up to the output dimension, resulting in a matrix, and then apply positional encoding. After passing through the S4 and Transformer, the output is averaged on the hidden dimension, resulting in a vector whose length is the output dimension. The network can take and produce an arbitrary length vector by these simple procedures. Such a network parametrizes the policy and value functions, and during the actual learning process, they take context information as input containing the environment name and the action space.

Design of Policy and Value Network. As shown in Figure 1b, the policy takes an arbitrary length input vector and then encodes it as a fixed-size hidden vector. Since action space can be discrete or continuous, we build the decoder layers for each type. Depending on the type of the action space, the policy generates either a Gaussian distribution or a Categorical distribution. We also add a state-dependent standard deviation decoder like SAC (Haarnoja et al., 2018) to adjust the degree of exploration depending on the environment. Note that our basic building block can also construct arbitrary input-output Q networks. So, our architecture can be utilized in off-policy, offline methods. If this is employed in the algorithm that uses both the state-action value function and state value function, like IQL (Kostrikov et al., 2021), multiple offline datasets can be simultaneously trained without the parametrized embedding functions for each environment.



(a) Basic Building Block. (b) Policy Network and Value Networks.

Figure 1: The building blocks for policy and value networks. B corresponds to the batch size, I to the input dimension, H to the hidden dimension, and O to the output dimension. The shapes of the tensors are described next to the block in Figure 1a. They consider both the input and output as sequences. The vector coming out of the encoder block and the task embedding vector are summed and then fed into the Residual MLP. By receiving the output dimension, they stack up the hidden vector from Residual MLP to the output dimension. Note that GAP is applied to the input dimension in the encoder while it is applied to the hidden dimension in the decoder. By this procedure, it can handle states and actions of arbitrary lengths.

3.2 Stabilizing Multi-Objective Optimization

Normalization. Different environments have varying ranges for states and rewards. We observed that the value loss fluctuated significantly when optimizing a value network for multiple environments simultaneously, making it challenging to learn and converge the value function properly. We normalize the state and reward to stabilize the value loss by using the running mean and standard deviation (Hessel et al., 2019).

Gradient Clipping. In multi-task learning, if the magnitude and similarity of the gradients for each

task differ, a particular task may dominate, leading to a negative transfer or suboptimal performance for other tasks (Yu et al., 2020). Kurin et al., 2022 showed that a simple unitary scalarization and regularization could improve multi-task performance. Inspired by this, we clip the gradient with the norm to avoid task domination and stabilize the optimization procedure.

3.3 Decentralized Distributed Algorithm for Heterogeneous Environments

Standard RL benchmarks and frameworks provide the interface for parallel environment execution (Brockman et al., 2016; Raffin et al., 2021; Liang et al., 2018; Weng et al., 2022a). However, the provided **VectorEnv** classes do not support parallel processing for heterogeneous environments. Different types of environments essentially cannot be batch-processed without an interface that supports padding and masking due to their distinct state and action spaces. Unless there isn't a specially designed environment like MetaMorph (Gupta et al., 2021), we should loop over the number of environments. We need additional techniques for efficient multiple environment rollout and optimization. We adopt and modify the DD-PPO (Wijmans et al., 2019) method for heterogeneous settings. The core idea is to run the copies of policy and value networks on multiple GPUs using DistributedDataParallel, performing rollout in different types of environments simultaneously, and then computing and synchronizing the gradients for training. We describe the entire process in the Algorithm 1. Note that S is the number of sub-environments for each worker. If we want to train 32 environments with 8 GPUs, S is 4, and the worker iterates over it. This decentralized distributed training procedure also can be applied to IQL (Appendix, Algorithm 2). We create as many replay buffers as there are environments, load the offline datasets, and then assign copies of the model parameters and buffers to each GPU worker to perform offline multi-environment training.

Algorithm 1 DD-PPO for Heterogeneous Online Multi-Environment RL

Require: number of the workers W, copies of policy and value parameters $\{\theta_w\}_{w=1}^{W}$, environments $\{E_i\}_{i=1}^{N}$, storages $\{B_i\}_{i=1}^{N}$.

Initialize parameters and assign S environments and S on-policy storages on each worker, $S = N/W$.
for $s = 1$ **to** S **do**
    Collect experiences in E_s for T timesteps with policy π_{θ_w} on distributed parallel workers
    Compute advantage estimates $\hat{A}_1, ...\hat{A}_T$
end for
for $k = 1$ **to** K **do**
    for $m = 1$ **to** M **do**
        $J_{\text{total}}(\theta_w^m) = 0$
        for $s = 1$ **to** S **do**
            Get minibatch $\mathcal{B}_s^m$ from B_s and calculate sum of policy and value losses $J_s(\theta_w^m)$
            $J_{\text{total}}(\theta_w^m) = J_{\text{total}}(\theta_w^m) + J_s(\theta_w^m)$
        end for
        Compute gradients of the sum of loss $\nabla_\theta J_{\text{total}}(\theta_w^m)$.
        $\theta_{1:W}^{m+1} = \mathsf{ParamUpdate}\Big(\theta_{1:W}^m, \mathsf{AllReduce}\big(\nabla_\theta J_{\text{total}}(\theta_1^m), ..., J_{\text{total}}(\theta_W^m)\big)\Big)$.
    end for
end for

4 Experiments

We train a multi-modal agent that can perform in heterogeneous environments. We evaluate our architecture and conduct multi-environment training in gym environments: Classic Control, Mujoco, and Atari. We aim to demonstrate whether multi-environment training is possible in the experiment. In online RL, we utilize the Envpool library (Weng et al., 2022b) to accelerate simulation execution speed, and we employ the D4RL dataset for offline RL (Fu et al., 2020).

4.1 Online and Offline Multi-Environment Training

Online Multi-Environment Training. First, we try to train the agent in 16 heterogeneous environments (Classic control, Mujoco). Except for four environments, Ant-v4, BipedalWalker-v3, Humanoid-v4, and Walker2d-v4, they perform similarly to or better than agents trained in a single environment. We hypothesize that the reason the 4 environments are not performing well is because they are more dominantly optimized for the 12 remaining environments.

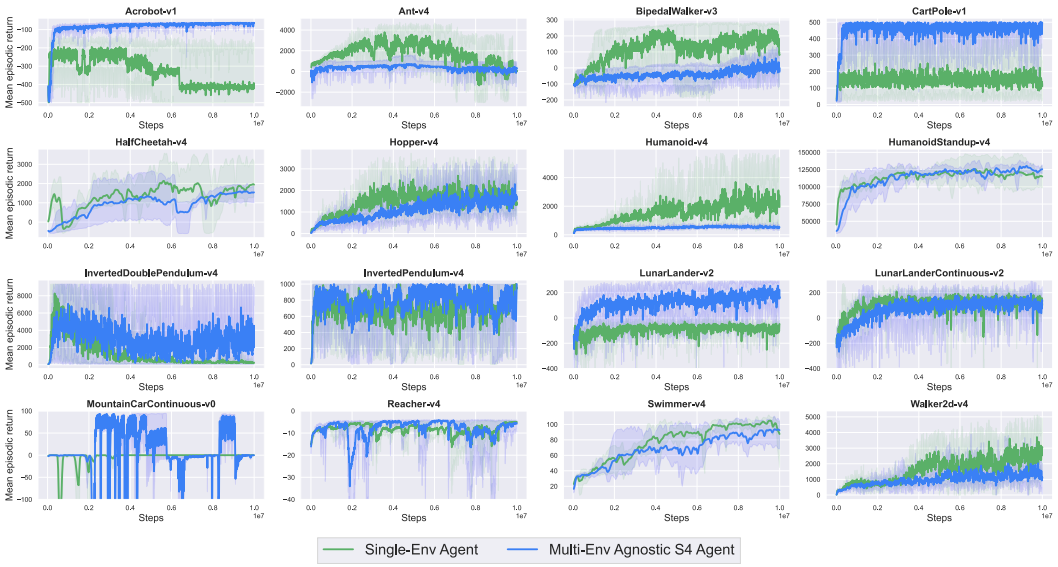


Figure 2: Results of multi-environment training in 16 heterogeneous environments on 4 seeds. The y-axis is the mean episodic return of 32 parallel same environments. It is possible to simultaneously train on both Classic control and Mujoco domains with a 1D-agnostic architecture. **6 Classic control environments**: Acrobot-v1, BipedalWalker-v3, CartPole-v1, LunarLander-v2, LunarLanderContinuous-v2, MountainCarContinuous-v0. **10 Mujoco environments**: Ant-v4, HalfCheetah-v4, Hopper-v4, Humanoid-v4, HumanoidStandup-v4, InvertedDoublePendulumn-v4, InvertedPendulumn-v4, Reacher-v4, Swimmer-v4, Walker2d-v4.

Offline Multi-Environment Training. We verify the feasibility of Offline Multi-Env RL. We implement the Decentralized Distributed IQL (DD-IQL) and conduct offline training on eight datasets simultaneously. Table 1 shows the normalized scores of IQL and DD-IQL. The training results for IQL are taken from Tarasov et al., 2022. In the DD-IQL, the policy and value functions are trained simultaneously using eight offline datasets. Except for HalfCheetah, Hopper, and AntMaze, the performances of DD-IQL are almost similar to IQL.

4.2 Online Multi-Environment Pretraining and Transfer Learning

This experiment aims to show that Multi-Env Agnostic Agents achieve performance close to the Single-Env Agent (an agent learned in a single environment) and to the Multi-Env Specific Agent (an agent having environment-specific encoders and decoders for different state-action space). The Multi-Env Specific Agent is a baseline with specific layers for each environment's input and output and a shared network in the middle. This architecture of the baseline is similar to parametrized embeddings in Gato (Reed et al., 2022). This means that the Multi-Env Specific Agent has an Encoder and Decoder blocks for each environment. In contrast, the encoder and decoder in Multi-Env Agnostic Agent are shared for multiple environments. Multi-Env Agnostic S4 Agent means that the embedding layer in front of the TransformerEncoder is the S4 layer. Similarly, Multi-Env Agnostic RNN Agent uses Bidirectional-GRU as a dynamic embedding layer. We try pretraining and transfer learning the three combinations of environments: 1) Classic-to-Mujoco, 2) Mujoco-to-Classic, and 3) Mix-to-Mix. All experimental results are normalized as $\text{normalized score} = 100 * \frac{\text{score} - \text{random score}}{\text{max score of single agent} - \text{random score}}$ based on the mean episodic returns of the Single-Env Agent and the Random Agent.

Pretraining. In Figure 3, the results represent the performances from learning eight environments

Table 1: Normalized performances in eight D4RL environments. While it didn't match the performances of the single-env policy across all environments (achieving about 89% of the performance), a possible advantage is its ability to perform transfer learning on new heterogeneous offline datasets.

Dataset	IQL (single-env policy)	DD-IQL (multi-env policy)
halfcheetah-medium-expert-v2	94.74 ± 0.52	82.07 ± 8.93
hopper-medium-expert-v2	107.42 ± 7.80	64.35 ± 2.34
walker2d-medium-expert-v2	111.72 ± 0.86	105.08 ± 4.59
antmaze-umaze-v2	77.00 ± 5.52	63.97 ± 9.85
pen-expert-v1	128.05 ± 9.21	120.84 ± 14.22
door-expert-v1	106.65 ± 0.25	105.19 ± 0.12
hammer-expert-v1	128.68 ± 0.33	125.4 ± 1.61
relocate-expert-v1	106.11 ± 4.02	103.02 ± 0.53
Total avg	107.54	96.22

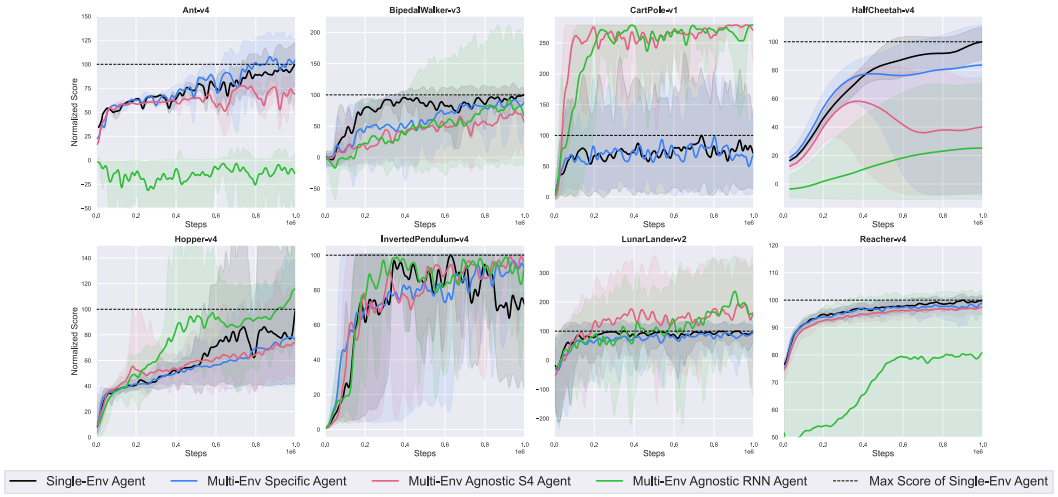


Figure 3: Mix-to-Mix multi-environment pretraining results on 4 seeds. The Single-Env Agent is parametrized only by Residual MLP blocks and represents the results from training on the single environment. The Multi-Env Specific Agent has multiple encoders and decoders that are parametrized by Residual MLP for processing the input and output for each environment, with a shared middle layer. The Multi-Env Agnostic Agent shares the entire network for multiple environments. In the case of S4, compared to Bi-GRU, it has the favorable characteristic of having a bounded gradient, which may result in less variance in performance during the training process.

simultaneously. The black line indicates the performance of the Single-Env Agent, and the black dotted line represents its peak performance. Although there is a performance decline in the HalfCheetah environment for the Multi-Env Agnostic Agents, they approach the performance of both the Multi-Env Specific Agent and Single-Env Agent in other environments. To compare the architectures, we pretrain for 1 million steps and then perform transfer learning on new environments for another 1 million steps.

Transfer Learning. Figure 4 shows transfer learning results on eight new environments. The results containing the word 'Scratch' mean that the agent is trained from scratch (not pretraining and transfer learning). For the typical transfer learning setting, the Multi-Env Specific Agent adds new encoder and decoder blocks for new environments. A noteworthy thing is the consistently lower performance of the Multi-Env Specific Agent when conducting transfer learning across all environments. This indicates that for new environments, it might be better to train from scratch rather than perform transfer learning, and the Multi-Env Agnostic Agents result in good performance in most environments. An intuitive reason for this result could be the difference in the weight distribution of the Residual MLP layer in the Multi-Env Specific Agent compared to a new layer, potentially hindering optimization. In

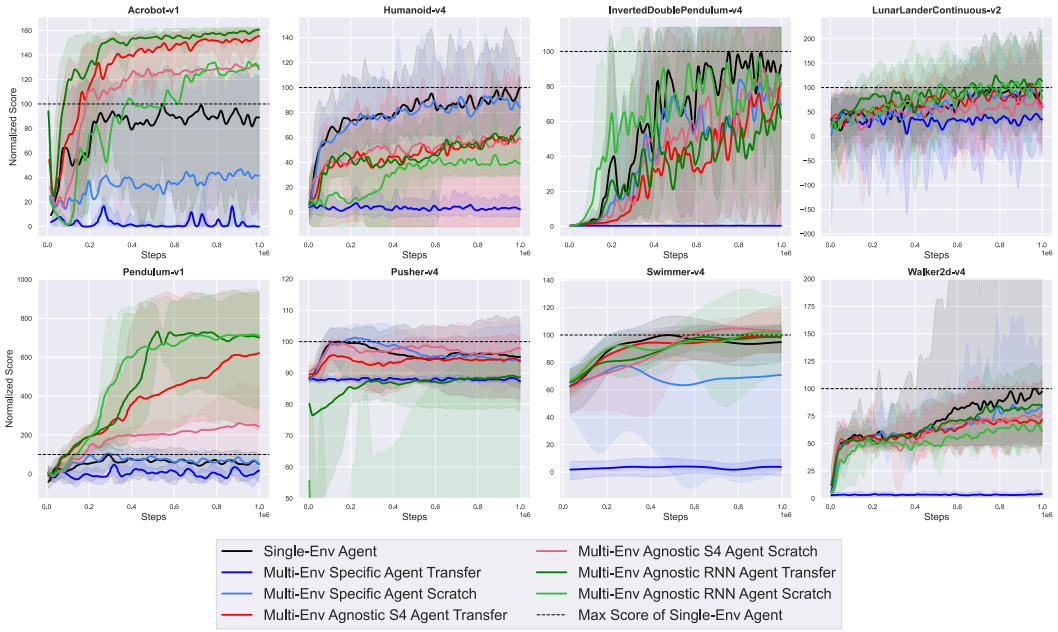


Figure 4: Mix-to-Mix multi-environment transfer learning results on 4 seeds. Note that the Multi-Env Specific Agent performs poorly in transfer learning when faced with new environments (**Blue**). The number of parameters for the Multi-Env Specific Agent increases due to the introduction of new layers, going from 5.8M in pretraining to 10.9M in transfer learning. In contrast, the Multi-Env Agnostic S4 Agent has 4M parameters. The lower performance of the Multi-Env Specific Agent Transfer could be due to the difficulty of optimization from different weight distributions between layers.

environments like Acrobot-v1 and Pendulumn-v1, experiences from pretraining seem to assist during transfer learning.

4.3 Ablation Study

We conduct ablation studies for measuring the performance when learning multiple environments with and without the middle Residual MLP layer and TransformerEncoder. We compare mean episodic returns in 3 cases: Pretraining, Transfer Learning, and Scratch Learning. In Figure 5, we can observe that, excluding Ant-v4, HalfCheetah-v4, and Reacher-v4, using the Transformer results in faster learning in the other five environments and the results in Appendix B also show a similar trend. Whether the Transformer is helpful will need to be analyzed by conducting extensive experiments in more environments.

5 Related Works

Previous research related to multi-environment reinforcement learning can be categorized into two types: Online Multi-Env RL and Offline Multi-Env RL. Most of these studies utilize padding, masking, and tokenization to preprocess data or build the environment's interface and ensure that agents handle the same state and action. The position of our work is in the middle of those two areas by considering heterogeneous gym environments: Classic Control, Mujoco, and Atari.

Online Multi-Env RL. Huang et al., 2020 proposed a method to learn multiple morphologies with a modular policy. Subsequent papers focused on whether information about the morphology was necessary, based on transformer models (Kurin et al., 2020, Hong et al., 2021, Trabucco et al., 2022). Gupta et al., 2021 demonstrated that transformer-based policy can be generalized to unseen morphologies and new dynamics. The characteristics of these papers involve performing the attention mechanism on the feature dimension, which effectively encodes morphology information, distinguishing different environments and tasks. However, these works were limited to the continuous

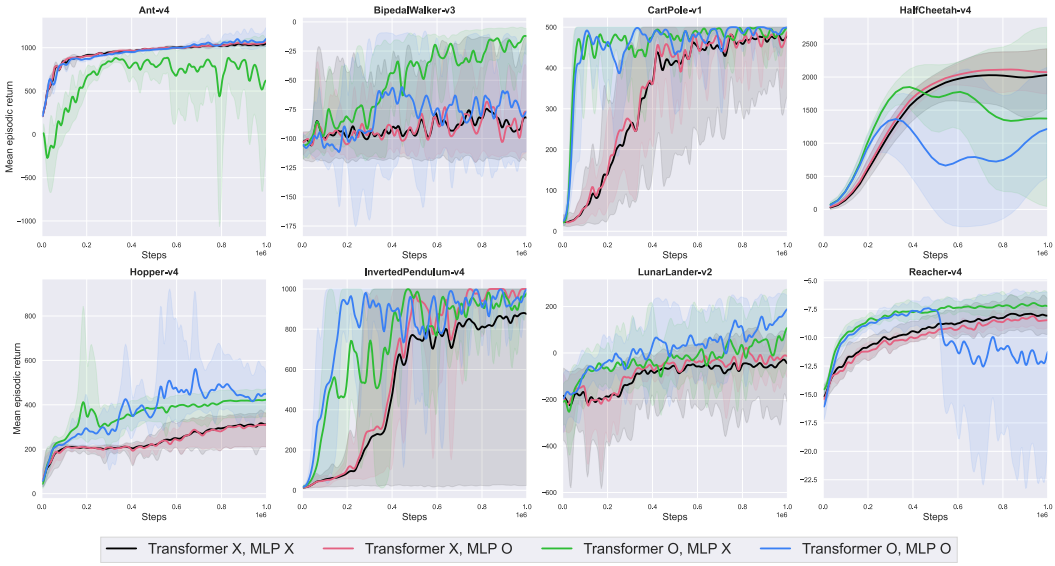


Figure 5: Results of ablation study about Mix-to-Mix multi-environment pretraining. In five environments, Equipping the TransformerEncoder block is beneficial.

control domain. On the one hand, Lu et al., 2023 first proposed the concept of Multi-Environment Meta-Learning and training the policy on multiple environments with random projection. In this concept, the RL agent learns multiple environments simultaneously and adapts to new environments with little experience. It might potentially be the necessary direction for the foundation model of RL agents.

Offline Multi-Env RL. Gato (Reed et al., 2022) and Multi-Game Decision Transformer (Lee et al., 2022) are the largest works on the Multi-Env RL. Their main characteristics are discrete tokenization and offline supervised sequence modeling with a Decision Transformer. By operating the attention mechanism on the temporal dimension, the transformer model predicts the next token conditioned by all previous tokens. Their works demonstrated that the DT-based policy achieved good performance across multiple environments. However, these models are parameter-heavy to be applied in the online RL setting. Furuta et al., 2022 developed multi-task benchmarking environments MxT-Bench and introduced morphology-task graph for unified representation. The model in this work showed good generalization performance for unseen morphologies and tasks by using data collected by a policy trained with PPO and conducting behavior distillation.

6 Conclusion

We proposed an agnostic architecture capable of multi-environment reinforcement learning regardless of the environment type by extending the multi-task RL. This architecture used fewer parameters than environment-specific architecture and was more beneficial in transfer learning. We combined the distributed training technique and conventional methods designed for stabilizing optimization to efficiently train the agent. This architecture was extended to offline RL by constructing arbitrary input-output action-value functions and showed near single-env learned agent's performance. In the future, it can be trained on the image-based RL domain with arbitrary image shape by introducing general-purpose CNN. However, since our architecture also used the Transformer, there remains a drawback. As the length of the input increases, the memory usage and computational load also increase. If we aim to introduce a multi-environment RL agent to real-world problems, it would be desirable to eliminate the Transformer for low-resource applications. Eventually, for the foundation model of reinforcement learning, we need a flexible and multi-modal architecture that can handle both continuous, discrete action spaces and the arbitrary shape of tensors as input and output. Moreover, we have to develop an RL algorithm that is not limited to only online or offline RL and implement efficient interfaces that can process heterogeneous environments.

Acknowledgments and Disclosure of Funding

This work was in part supported by the MSIT, Korea, under the ICT Creative Consilience program (IITP-2023-2020-0-01819) supervised by the IITP.

References

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Hiroki Furuta, Yusuke Iwasawa, Yutaka Matsuo, and Shixiang Shane Gu. A system for morphology-task generalization via unified representation and behavior distillation. In *The Eleventh International Conference on Learning Representations*, 2022.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021a.

Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021b.

Agrim Gupta, Linxi Fan, Surya Ganguli, and Li Fei-Fei. Metamorph: Learning universal controllers with transformers. In *International Conference on Learning Representations*, 2021.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

Sunghoon Hong, Deunsol Yoon, and Kee-Eung Kim. Structure-aware transformer policy for inhomogeneous multi-task reinforcement learning. In *International Conference on Learning Representations*, 2021.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pages 4455–4464. PMLR, 2020.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

David M Knigge, David W Romero, Albert Gu, Efstratios Gavves, Erik J Bekkers, Jakub M Tomczak, Mark Hoogendoorn, and Jan-Jakob Sonke. Modelling long range dependencies in nd: From task-specific to a general purpose cnn. *arXiv preprint arXiv:2301.10540*, 2023.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*, 2020.

Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and Pawan K Mudigonda. In defense of the unitary scalarization for deep multi-task learning. *Advances in Neural Information Processing Systems*, 35:12169–12183, 2022.

Kuang-Huei Lee, Ofir Nachum, Mengjiao Sherry Yang, Lisa Lee, Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *Advances in Neural Information Processing Systems*, 35:27921–27936, 2022.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. *arXiv preprint arXiv:2303.03982*, 2023.

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4nd: Modeling images and videos as multidimensional signals with state spaces. *Advances in neural information processing systems*, 35:2846–2861, 2022.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1):12348–12355, 2021.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR, 2023.

Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. Corl: Research-oriented deep offline reinforcement learning library. *arXiv preprint arXiv:2210.07105*, 2022.

Brandon Trabucco, Mariano Phielipp, and Glen Berseth. Anymorph: Learning transferable polices by inferring agent morphology. In *International Conference on Machine Learning*, pages 21677–21691. PMLR, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022a. URL http://jmlr.org/papers/v23/21-1127.html.

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, et al. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 35:22409–22421, 2022b.

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019.

Zheng Xiong, Jacob Beck, and Shimon Whiteson. Universal morphology control via contextual modulation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 38286–38300. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/xiong23a.html.

Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

A DD-IQL Algorithms

Algorithm 2 DD-IQL for Heterogeneous Offline Multi-Environment RL

Require: number of the workers W, copies of policy and value parameters $\{\theta_w\}_{w=1}^{W}$, environments $\{E_i\}_{i=1}^{N}$, replay buffers $\{B_i\}_{i=1}^{N}$.

Initialize parameters and assign S environments and S replay buffers on each worker, $S = N/W$.
for $t = 1$ **to** T **do**
  $L_{\text{total}} = 0$
  for $s = 1$ **to** S **do**
    Get minibatch $\mathcal{B}_s$ from B_s and calculate loss L_V, L_Q, L_π with Equations (3), (4), (5)
    $L_{\text{env}} = L_V + L_Q + L_\pi$
    $L_{\text{total}} = L_{\text{total}} + L_{\text{env}}$
  end for
  Compute gradients of the sum of loss $\nabla_{\theta_w} L_{\text{total}}$.
  Update networks. $\theta_{1:W}^{t+1} = \mathsf{ParamUpdate}\Big(\theta_{1:W}^{t}, \mathsf{AllReduce}\big(\nabla_{\theta_1} L_{\text{total}}, ..., \nabla_{\theta_W} L_{\text{total}}\big)\Big)$.
end for

B Hyperparameters

Hyperparameter	Value
actor_lr	0.0003
critic_lr	0.0005
update_epochs	10
num_minibatches	8
max_grad_norm	0.5
clip_coef	0.2
entropy_coef	0.0
gamma	0.99
gae_lambda	0.95

(a) PPO Hyperparameters.

Hyperparameter	Value
d_model	256
activation	$GELU$
depth of ResMLP	2
hidden size of ResMLP	256
dropout	0.1
dim_feedforward	256
num_head	1
num_encoder_layers	2
d_state of S4	256
num_layers S4	2

(b) Hyperparameters of NN.

Table 2: Hyperparameters used in experiments

C Additional Experiment of Multi-Environment Training

C.1 Atari and Mujoco

We tried to construct a network to handle arbitrary image resolutions using S4ND (Nguyen et al., 2022) or CCNN (Knigge et al., 2023) for a general-purpose multi-modal agent. However, their learning speed was so slow that we could not conduct the experiment. Therefore, we explore using ResNet (He et al., 2016) to see if a multi-modal policy could be trained. We build a multi-modal policy by adding ResNet for the encoder Block in Figure 1a and simultaneously train on 6 Atari games and 6 Mujoco environments. Image inputs are embedded by the ResNet and fed to Residual MLP. The Atari games share the ResNet encoder, Residual MLP, and discrete S4-Transformer decoder. In Figure 6, the performances of multi-modal policy are poor in Atari games. It requires 20 million frames to achieve these performances, showing sample inefficiency. Due to the challenges of multi-objective optimization, hyperparameters might not be effective for multi-environment settings. We need to conduct an extensive hyperparameter search for this setting. In future research, architectures that work for all Atari games have to be investigated.

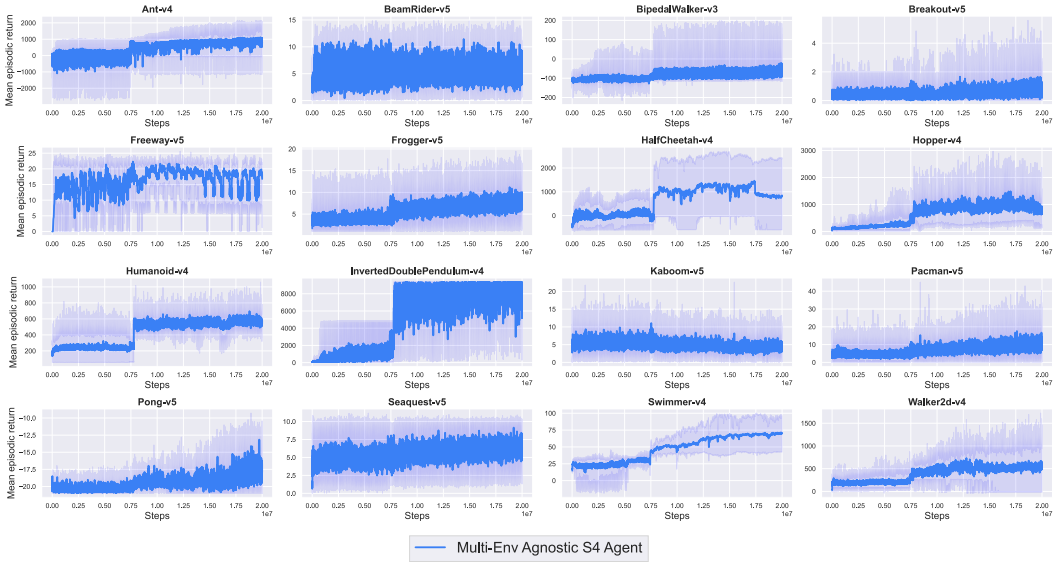


Figure 6: Results of multi-environment training in 16 heterogeneous environments on 4 seeds. The y-axis is the mean episodic return of 32 parallel same environments. While the sample efficiency and performance are low in the Atari environments, it suggests that it is possible to simultaneously train on both Atari and continuous control domains with a multi-modal policy. **8 Atari games**: BeamRider-v5, Breakout-v5, Freeway-v5, Frogger-v5, Kaboom-v5, Pacman-v5, Pong-v5, Seaquest-v5. **8 Continuous control environments**: Ant-v4, BipedalWalker-v3, HalfCheetah-v4, Hopper-v4, Humanoid-v4, InvertedDoublePendulumn-v4, Swimmer-v4, Walker2d-v4.

D Additional Experiments Results of Pretraining and Transfer Learning

D.1 Classic-to-Mujoco

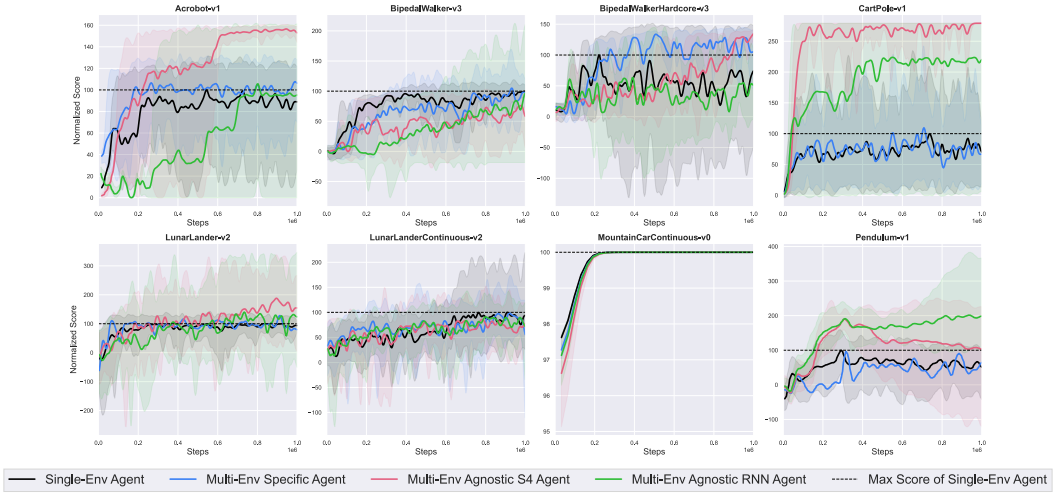


Figure 7: Classic-to-Mujoco multi-environment pretraining results.

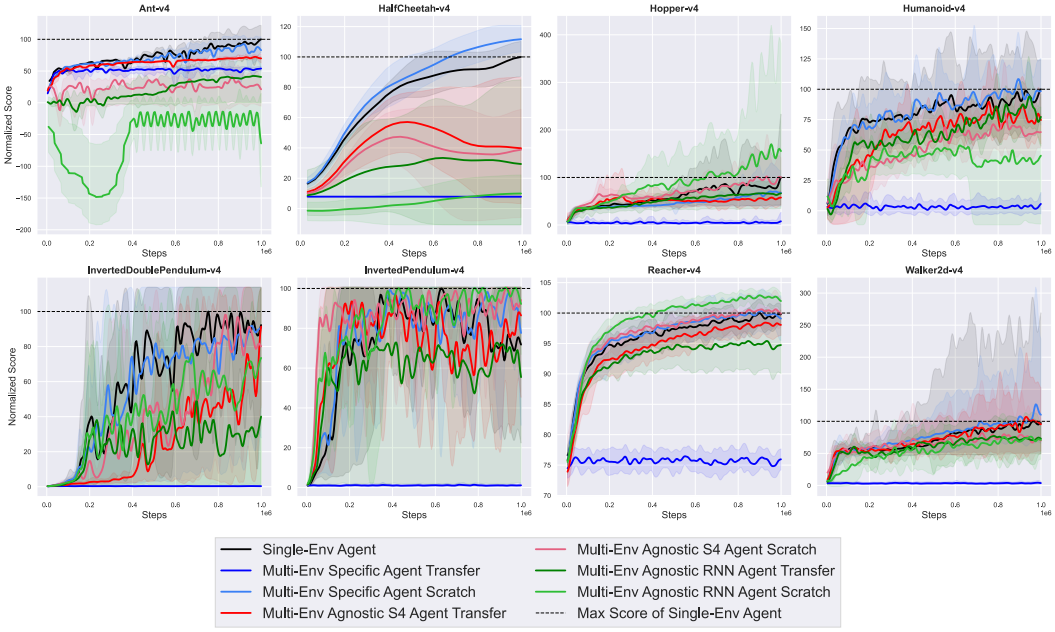


Figure 8: Classic-to-Mujoco multi-environment transfer learning results.

D.2 Mujoco-to-Classic

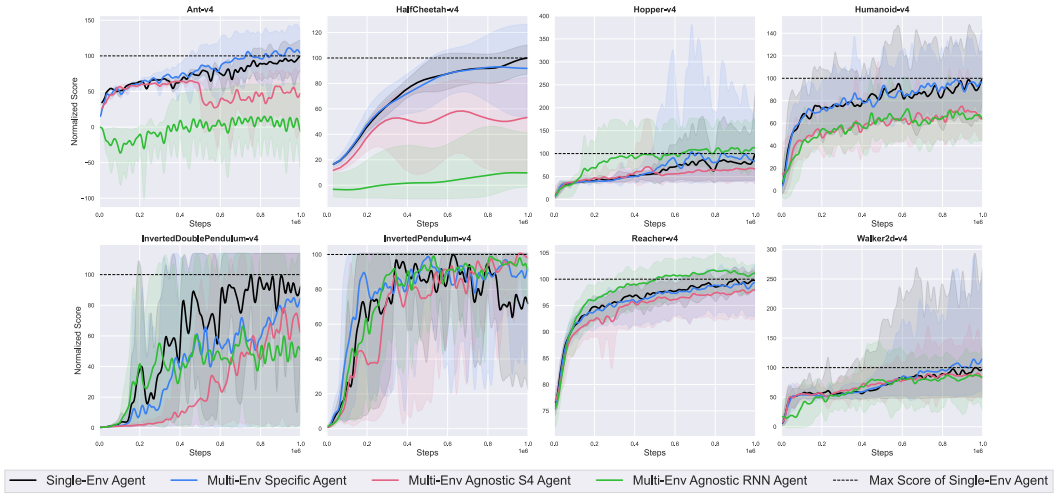


Figure 9: Mujoco-to-Classic multi-environment pretraining results.

Figure 10: Mujoco-to-Classic multi-environment transfer learning results.

E Ablation study

E.1 Transfer learning

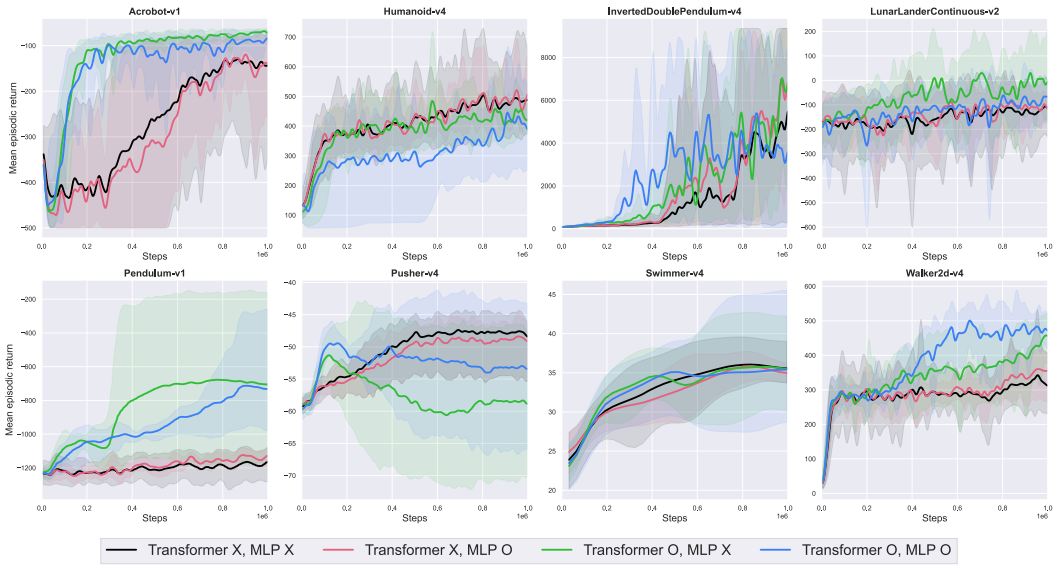


Figure 11: Results of ablation study about Mix-to-Mix multi-environment transfer learning.

E.2 Scratch learning

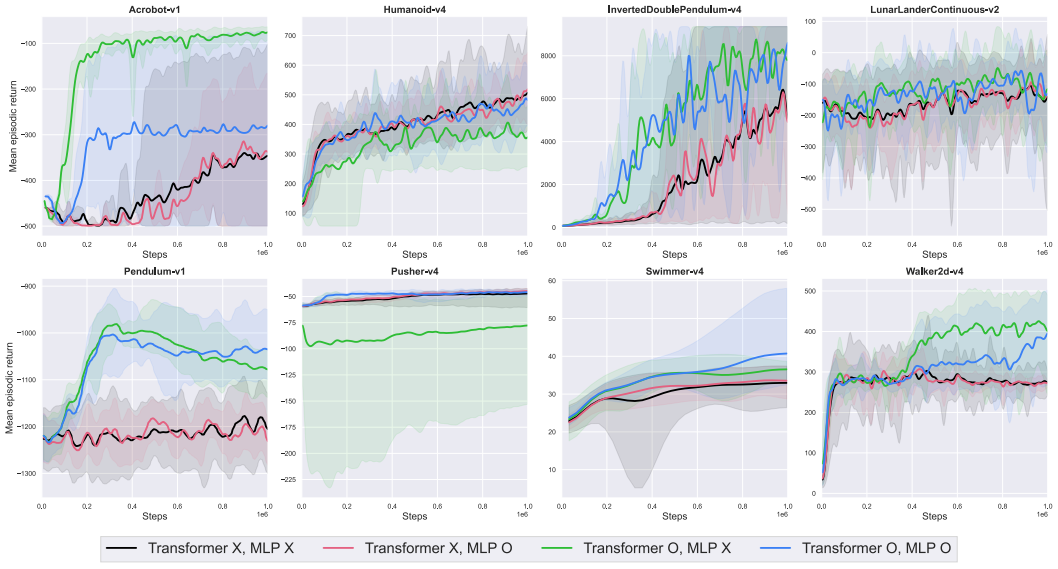


Figure 12: Results of ablation study about Mix-to-Mix multi-environment scratch learning