MULTI-CRITIC ACTOR LEARNING:
TEACHING RL POLICIES TO ACT WITH STYLE

Siddharth Mysore  
Boston University & Electronic Arts

George Cheng & Yunqi Zhao  
Electronic Arts

Kate Saenko  
Boston University & MIT-IBM Watson AI Lab

Meng Wu  
Electronic Arts

ABSTRACT

Using a single value function (critic) shared over multiple tasks in Actor-Critic multi-task reinforcement learning (MTRL) can result in negative interference between tasks, which can compromise learning performance. Multi-Critic Actor Learning (MultiCriticAL) proposes instead maintaining separate critics for each task being trained while training a single multi-task actor. Explicitly distinguishing between tasks also eliminates the need for critics to learn to do so and mitigates interference between task-value estimates. MultiCriticAL is tested in the context of multi-style learning, a special case of MTRL where agents are trained to behave with different distinct behavior styles, and yields up to 56% performance gains over the single-critic baselines and even successfully learns behavior styles in cases where single-critic approaches may simply fail to learn. In a simulated real-world use case, MultiCriticAL enables learning policies that smoothly transition between multiple fighting styles on an experimental build of EA’s UFC game.

1 INTRODUCTION

Reinforcement Learning (RL) offers an interesting means by which to develop interesting behaviors in a variety of controls settings. The work presented in this paper is primarily contextualized by our efforts to develop RL techniques for teaching RL-policies to behave with multiple distinct and specific styles for game-play. RL can be useful in developing control policies for game-play and has been demonstrated as being capable of learning human-like or even superhuman play, most notably beating top-ranked human players in games such as Dota 2 (Berner et al., 2019), StarCraft (Vinyals et al., 2019), Chess (Silver et al., 2017), and Go (Silver et al., 2016). RL-based control in games offers a wide array of potential applications, including testing (Zhao et al., 2020; Ariyurek et al., 2021; Gordillo et al., 2021), game design (Gisslen et al., 2021), providing competition for human players (Berner et al., 2019), or simply as a means to develop more interesting game-play AI behaviors (Zhao et al., 2020; Alonso et al., 2020). RL algorithms can be notorious however for training aesthetically unappealing policies, but prior work has demonstrated that, with careful shaping of the reward functions, it is possible to derive highly specific desirable behavior (Peng et al., 2018; Mysore et al., 2018; Mysore et al., 2021). Machine learning (ML) solutions to game-play AI can however represent a significant increase in compute requirements over their heuristic-based counterparts, especially when using deep learning techniques. If a learned control policy could represent multiple desirable styles and offered a controllable way to transition between the learned styles, this could reduce the total compute burden of the ML-based controller, making it more practically viable. Ideally, developing an ML model would constitute learning a single, low-complexity policy that would model and transition smoothly between multiple behavior styles. Reduced model complexity and deploying just a single policy network enables greater compute efficiency through hardware parallelism and reduced memory costs, which is beneficial in resource-constrained applications, such as videogames.

We explore the problem of multi-style RL, a special case of the more commonly explored problem of multi-task RL (MTRL). Whereas MTRL techniques typically seek to solve a wide array of RL problems under a single learning campaign, multi-style RL adds the extra constraint that the ‘tasks’ in question all have identical system dynamics but would be characterized by different re-
The core reinforcement learning (RL) problem is framed as a finite-horizon Markov decision process, \( \langle S, A, P, r, \gamma \rangle \), with state transition dynamics \( P : S \times A \rightarrow S \), for states \( s \in S \) and actions \( a \in A \), and reward function \( r : S \times A \rightarrow \mathbb{R} \). The RL objective is to train a policy, \( \pi : S \rightarrow A \) such that the expected sum of discounted rewards, \( \mathbb{E}_{\tau_{\pi}} \left[ \sum_{t} \gamma^{t} r(s_t, a_t) \right] \), with discount factor \( \gamma \), is maximized for all trajectories, \( \tau_{\pi} \), sampled on policy \( \pi \). RL typically assumes that both the transition dynamics and reward dynamics are encompassed by a black-box environment. RL agents interact with the environment to observe the state response and generated rewards, but where specific reward functions and transition dynamics are not assumed known. Solving the RL optimization problem requires estimating expected rewards and using these approximations to optimize the action policies.

Among the different techniques for tackling RL optimization, we mainly consider Actor-Critic methods, whose key characteristic is their separation of the ‘actors’, representing the RL policy, and the ‘critic’, representing the value-function estimator that learns to estimate future rewards. This sep-
aration allows Actor-Critic RL to take advantage of the improved sample-efficiency of value-based techniques, such as Q-learning (Watkins & Dayan, 1992) while taking advantage of policy-based techniques, such as policy gradients (Sutton et al., 2000), for improved robustness to stochasticity and to enable learning in continuous action domains. Examples of prominent contemporary algorithms in this class of RL methods include TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017), DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018).

In algorithms based on Q-learning, such as DDPG, TD3, and SAC, the policy optimization criteria $J_{\pi_\theta}$ for a policy, $\pi$, parameterized by $\theta$, is proportional to the Q-value:

$$J_{\pi_\theta} \propto Q_{\pi_\theta}(s, a) = E_s[R(s, a)|s_0 = s, a_0 = a] = E_s[\gamma V_{\pi_\theta}(s', \pi_{\theta}(s'))]$$

where $s'$ is the next state reached from state $s$ in response to action $a$, and $R(\tau)$ is the discounted sum of rewards over trajectory $\tau$. For algorithms based on a more traditional policy gradient optimization, such as TRPO and PPO, policies are optimized proportional to the advantage function, $A_{\pi_\theta}$ and a measure of policy divergence:

$$J_{\pi_\theta} \propto \log(\pi_{\theta}(a|s)) A_{\pi_\theta}(s, a) = \log(\pi_{\theta}(a|s)) (Q_{\pi_\theta}(s, a) - V_{\pi_\theta}(s))$$

where the value-function $V_{\pi}(s) = E_{s'}[R(s, \pi(s))|s_0 = s] = E_s[r(s, \pi(s)) + \gamma V_{\pi}(s')]$. For critic networks parameterized by $\phi$, either a state-value estimator, $V_\phi$, or Q-value (state-action-value) estimator, $Q_\phi$, are optimized with optimization criteria $J_{V_\phi}$ or $J_{Q_\phi}$ respectively:

$$J_{V_\phi} \propto ||V_\phi(s) - (r(s, \pi(s)) + V_\pi(s') \sim P(s, \pi(s)))||$$

$$J_{Q_\phi} \propto ||Q_\phi(s, a) - (r(s, a) + Q_\pi(s') \sim P(s, a), \pi(s')))||$$

While specific details around how actors and critics are optimized vary between algorithms, the above represents the fundamentals of optimization criteria upon which we shall build our discussion.

3 Related Work

We broadly categorize multi-task RL (MTRL) techniques as either single-policy, if the same policy function activations are used to determine actions over all tasks, or multi-policy, if more separate activations (in the case of multi-headed policies) or policy functions are used. Multi-policy techniques range from training multiple policies and selecting or mixing between them (Andreas et al., 2017; Peng et al., 2018; Dewangan et al., 2018; Yang et al., 2017; Huang et al., 2017), to simultaneously or iteratively training additional policies while sharing representation between policies (Rusu et al., 2015; Teh et al., 2017; Rusu et al., 2016). Techniques requiring multiple policy networks are limited by their storage and computational costs during inference, and multi-headed policies can limit our ability to smoothly transition between behaviors, which makes them less attractive in the context of RL for game-play. Therefore, we focus instead on single-policy learning.

Single-policy MTRL techniques train a single policy function to solve multiple tasks, without relying on information from previously trained policies. Model-free meta-learning techniques have been proposed to improve multi-task generalization, in the absence of task specificity (Finn et al., 2017; Nichol et al., 2018), but where explicit task distinction can be exploited, a common approach has been to exploit task encoding to distinguish between tasks. A simple, yet effective approach to this problem has been to extend state representations with one-hot task encoding (Peng et al., 2018; Yu et al., 2020; Yang et al., 2020), thus allowing policies to distinguish between individual tasks and decide actions in a task-appropriate way. Recent work has also sought to enhance the performance and generalizability of such single-policy techniques through automated task encoding (Sodhani et al., 2021) or through soft-modularization (Yang et al., 2020), which encourages the development of distinct information pathways per task within actor networks.

Our work considers the special MTRL case of multi-style learning. Multi-style learning requires agents to learn different behaviors for highly similar states while providing potentially highly dissimilar rewards for each style. This variability in the reward signals can have an adverse impact on the quality of learned value-functions when using a single value estimator if the estimator cannot sufficiently distinguish between styles. We hypothesize that negative interference more strongly impacts value-function learning in multi-style tasks as task similarity may make specific style-values harder to distinguish. Multi-style learning is however an aspect of MTRL that has not received much
prior consideration. Peng et al. (2018) explore the most similar setting to our work – where agents were required to learn different movement styles in animation tasks. Their work tested single-policy MTRL and found that policies were only able to learn a limited number of styles, requiring the authors to resort to a multi-policy approach.

Most works considering the Actor-Critic MTRL problem focus predominantly on how the actors, i.e. the policy-functions, are optimized, with less attention given to critics. The problem of negative interference between tasks in the broader context of MTRL is one that has been highlighted and studied before (Yang et al., 2020; Parisotto et al., 2015; Teh et al., 2017; Standley et al., 2020) however, but has similarly been approached as a policy optimization problem. A number of multi-policy techniques were noted to maintain separate valuations per policy (Teh et al., 2017; Andreas et al., 2017; Peng et al., 2018), though some work has also suggested that sharing critics can allow optimization to better capture useful shared knowledge to improve policy optimization (Dewangan et al., 2018; Yang et al., 2017). A similar idea dominates single-policy techniques, where all the studied methods maintain a single value-function estimator across all tasks (Nichol et al., 2018; Yu et al., 2020; Peng et al., 2018; Yang et al., 2020; Sodhani et al., 2021). While task-encoding may allow separate task-values to be learned implicitly by the critics, this was never found to be explicitly designed. Our work alternatively frames the multi-style learning framework as one that explicitly learns distinct per-style values to propose a value-learning optimization scheme that is better able to accommodate disparate reward functions.

4 Multi-Critic Actor Learning

The policy functions, i.e., the ‘actor’ part of Actor-Critic frameworks, represent the core decision-making processes of RL agents and the success or failure of any algorithm rests on the policies’ abilities to learn useful mappings. The learned critics have limited utility after training and learned value-function estimators are also known for being poor approximations of the true value-functions they attempt to capture (Ilyas et al., 2020), so it makes sense that prior work has focused more on how policies are constructed and trained. The limitations of value learning are however exactly why the specifics of value-network constructions and optimization are important to consider. A critical element in the performance of RL algorithms can be the representational capacity of the learned value-functions (Mysore et al., 2021a; Andrychowicz et al., 2021). It, therefore, stands to reason, that better structuring how value-networks represent the learned values can be an important component of improving value-function learning and therefore policy performance.

A common formulation of single-critic MTRL optimization involves including a task-encoding, $\Omega$, to distinguish between learned tasks and to specify task rewards and updating the state representation $\tilde{s} = (s, \Omega)$. The optimization criteria in Equations (1-4) are correspondingly updated to use $\tilde{s}$ in place of $s$. The resulting MTRL optimization criteria typically represent rewards and learned values by scalar values. Practically, the value-functions are learned by a single network with a single output node, regardless of the number of tasks. A single learned value-function representing multiple values may enforce continuity between the learned values, which may not necessarily reflect the true value functions and could compromise the quality of learned values and resulting policies.

Value Optimization for Multi-Critic Actor Learning  

By separating the learned values per style, assumptions of continuity between style-values can be dismissed and style (or task) values can be learned explicitly, without relying on the critics to inherently learn the distinctions. We propose Multi-Critic Actor Learning (MultiCriticAL) with a style-conditioned value optimization criteria:

$$J_{V_{\tilde{s}, \Omega}} \propto \left| \sum \omega_i \left[ V_{\phi_i}^\pi(s) - (r_i(s, \pi(\tilde{s}))) + V_{\phi_i}^\pi(s') \sim P(s, \pi(\tilde{s})) \right] \right|$$  \hspace{1cm} (5)

$$J_{Q_{\tilde{s}, \Omega}} \propto \left| \sum \omega_i \left[ Q_{\phi_i}^\pi(s, a) - (r_i(s, a)) + Q_{\phi_i}^\pi(s') \sim P(s, a, \pi(\tilde{s}))) \right] \right|$$  \hspace{1cm} (6)

where, for $M$ styles, the task/style encoding, $\Omega = [\omega_0, \ldots, \omega_M]$ and $\omega_i \in [0, 1]$. The policy optimization criteria requires the value functions to be continuous with respect to the states and task encoding, and this property can be maintained by appropriately selecting the value functions when computing the policy optimization criteria. The specifics of the exact formulation of the optimization criteria would vary in accordance with the base Actor-Critic algorithm employed, with Equa-
Practically, multi-valued critics may be represented by separate independent critic networks – multi-network (MN) MultiCriticAL – or a single base critic network with multiple heads, where each head represents a different learned value – multi-headed (MH) MultiCriticAL. While the latter allows for some learned representations to be shared between the value functions, the former may be more beneficial when sharing is likely to lead to undesirable interference. This is visually shown in Figure 1 and extended in Figure 14 in Appendix C.

5.1 Path Following

Environment Design and Reward Shaping In the path following environment, given its current position in space, the current phase of motion $\in [0, 1]$, and an indicator on which shape to reproduce, the agent is expected to follow a trajectory that generates the desired shape. Three shapes are tested:
The path-following task requires agents to learn a Circular, Square, or Triangular path. MultiCriticAL agents can learn to follow all 3 shapes with better performance compared to the more typical single-critic multi-task value-learning frameworks of MTSAC and MTPPO.

Circle, Square and Triangle. This simple problem adds no additional complexity related to generalization or complex system dynamics and simply expects policies to learn a mapping function that encodes the desired shape. The multi-style objective is for the agents to accurately produce any of the three required shapes, given the task encoding. Agents are rewarded based on the absolute error between the agent’s current and desired position in space. The rewards are cast to be positive in the range of $[0, 1]$ per step. Appendix D offers further details, with code in Supplementary Material.

Results

The pathing behavior learned for each algorithm over the three shapes is shown in Figure 5.1. While single-style SAC and PPO are able to learn to follow each shape, performances of MTSAC and MTPPO in the multi-style setting are compromised. This is most prominent with MT-SAC. MTPPO fares better but performance in the multi-style setting is still notably worse. Adapted to both algorithms, MultiCriticAL more consistently learns to follow all three shapes. It should also be noted that larger networks were needed for the PPO algorithm to solve this task, using two hidden layers with 32 neurons each, as opposed to SAC’s 8 each. This likely contributes to the improved performance of PPO in the multi-task setting as the networks being used have higher relative representational capacity, but are still outperformed by MultiCriticAL.

5.2 Pong with Rotatable Paddles

Environment Design and Reward Shaping

We use a modified game of Pong to explore a multi-style learning problem where one style is represented by a sparse reward, and the other a dense reward, anticipating that this might be a case where single-critic value-function estimation may fail. Pong typically allows only a single degree of freedom for paddle movement and does not normally command nuanced strategies. Adding an extra degree of freedom by allowing the player’s paddles to rotate introduces increased complexity to the game and facilitates the training of different behavior styles. For this game, we consider two primary styles of play: (i) aggressive – with the objective of winning the game as quickly as possible, and (ii) defensive – where the objective is to avoid losing while prolonging the game as long as possible. In a real game setting, these could be analogous to hard- and easy-mode AI respectively, where the former tries to give the opponent a challenge and the latter may help the opponent learn to play. Our implementation is based on the Pong environment provided by the Pygame Learning Environment (PLE) [Tasfi, 2016]. For simplicity, the basic opponent AI cannot rotate its paddle and uses a heuristic-based Pong controller provided by PLE. The Aggressive style rewards winning as quickly as possible while the Defensive style encourages avoiding losing and ensuring that the opponent can keep receiving the ball to prolong the game (additional details in Appendix D).

Results

We first established baseline performance by training SAC and PPO on the Aggressive and Defensive play-styles. Figure 3 shows samples of play strategies adopted for each style, where the Aggressive agent exploits paddle rotation to bounce the ball off the side-wall to outmaneuver the opponent. Conversely, Defensive agents attempt to return the ball directly to the opponent, exploiting the paddle rotation to angle the ball back towards the opponent. As desired, aggressive play yields high win-rates, with plays lasting significantly shorter than defensive play, which would win approximately half of the games but play longer games (videos in Supplementary Material).
Multi-style performance is tested by introducing a one-hot style encoder and randomly uniformly sampling either defensive or aggressive play for each new training episode. We observed that both MTSAC and MTPPO failed to learn to properly distinguish between play-styles. MTSAC agents often defaulted to highly defensive play (likely due to the over-representation of the dense defensive rewards in the replay buffers) while MTPPO agents tended to learn intermediate behavior that was neither aggressive nor defensive. MultiCriticAL agents were however able to successfully learn to distinguish between the styles with both base algorithms, with performance comparable to single-style agents. Table 1 and Figure A.2.1 show performance statistics averaged over 20 test episodes per seed for each algorithm.

We also test how MultiCriticAL agents would respond to changes in the style encoding at test time. As shown in Table 2 when provided an intermediate (0.5,0.5) encoding, the agents presented with behavior statistics that were almost halfway between that of their aggressive and defensive statistics. This was even without explicit conditioning, when agents were trained with purely binary style indicators. When explicitly interpolating between styles in training (by providing a multi-dimensional style-reward, with aggressive and defensive rewards scaled by their encoding weight), both MultiCriticAL SAC and PPO agents yielded performance statistics that more closely matched the mid-points between their aggressive and defensive behaviors.

### 5.3 Sonic the Hedgehog

**Training Setup** To test MultiCriticAL in a more established multi-task setting, we tested two Sonic the Hedgehog games, using the Gym-retro wrappers designed for the games during the Gym-retro contest [Nichol et al., 2018], using the same setup configuration developed for the competition. MultiCriticAL is set up to maintain a separate critic for each trained level. Sufficient experimentation for statistical rigor on this benchmark required hardware parallelism, which our code only supports for PPO. SAC is therefore omitted for this benchmark.

**Results** Performance on 17 of the main levels for each game, averaged over 10 tests per agent, per level, are presented in Figure A.2.2 with per-level tabulated numerical rewards provided in Appendix A. Sonic the Hedgehog appears to be a game where learning shared representations over multiple levels helps overall performance, for both MultiCriticAL and MTPPO, though this observation does not hold for Sonic the Hedgehog 2, where MTPPO agents perform worse on average than agents trained on single-levels. MultiCriticAL agents tend to perform comparable or better than independently trained and MTPPO agents in almost all cases, however, with 35% and 20% improvements on Sonic the Hedgehog and 56% and 40% improvement on Sonic the Hedgehog 2 of MN-MultiCriticAL and MH-MultiCriticAL over MTPPO respectively.

---

Table 1: Learning different play-styles on Pong with rotatable player paddles

<table>
<thead>
<tr>
<th>Agent</th>
<th>Single-style</th>
<th>MT w/ one-hot</th>
<th>MN-MultiCriticAL</th>
<th>MH-MultiCriticAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aggressive-Setting → Style = (1, 0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reward ↑</td>
<td>0.89 ± 0.05</td>
<td>-0.20 ± 0.35</td>
<td>0.85 ± 0.01</td>
</tr>
<tr>
<td></td>
<td>Win-rate ↑</td>
<td>94 ± 2%</td>
<td>40 ± 13%</td>
<td>86 ± 3%</td>
</tr>
<tr>
<td></td>
<td>Play-time ↓</td>
<td>280 ± 102</td>
<td>170 ± 51</td>
<td>255 ± 35</td>
</tr>
<tr>
<td></td>
<td>Defensive-Setting → Style = (0, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reward ↑</td>
<td>162.75 ± 12.15</td>
<td>166.05 ± 14.04</td>
<td>169.60 ± 8.50</td>
</tr>
<tr>
<td></td>
<td>Win-rate ↑</td>
<td>51 ± 3%</td>
<td>50 ± 5%</td>
<td>30 ± 5%</td>
</tr>
<tr>
<td></td>
<td>Play-time ↓</td>
<td>1536 ± 113</td>
<td>1571 ± 122</td>
<td>1661 ± 53</td>
</tr>
</tbody>
</table>

Table 2: Interpolating between aggressive and defensive Pong play-styles → Style = (0.5, 0.5)

<table>
<thead>
<tr>
<th>Agent</th>
<th>SAC</th>
<th>PPO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MN-MultiCriticAL</td>
<td>MH-MultiCriticAL</td>
</tr>
<tr>
<td></td>
<td>Training with Binary Style Selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Win-rate ↑</td>
<td>71 ± 3%</td>
</tr>
<tr>
<td></td>
<td>Play-time ↑</td>
<td>770 ± 148</td>
</tr>
<tr>
<td></td>
<td>Training with Explicit Style Interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Win-rate ↑</td>
<td>73 ± 6%</td>
</tr>
<tr>
<td></td>
<td>Play-time ↑</td>
<td>834 ± 108</td>
</tr>
</tbody>
</table>
Figure 4: Evaluated over two Sonic games with 17 training levels per game, where levels function similarly but have significantly different layouts and visual design, MultiCriticAL offers consistently superior performance to the single-critic baseline and achieves comparable, if not better performance to agents trained specifically for each level – additional data and visualization in Appendix A.2.2.

Figure 5: We show 8 seconds of play with one representative frame per second for the different styles of play for the MultiCriticAL agent trained to play UFC. The RL agent has black hair while the stock AI has white hair. All styles are captured for the same RL agent, trained to play with multiple behavior styles. As desired, the agent more frequently goes on the offensive when set to Aggressively, prioritizes constant blocking and avoiding damage on Defensive, standing its ground and blocking incoming attacks on Blocking and moving around the ring for Moving.

5.4 UFC Environment

For a use case more reflective of real-world application, we trained AI fighters in an experimental build of UFC Knockout Mode. A gym-like (Brockman et al., 2016) wrapper was built for the game environment to enable an RL agent to play against the game’s stock AI. Note that this is not a feature of the commercially released game but is a modification made for research purposes.

Environment design and Reward Shaping The RL agent observes information about the player’s and opponent’s position, velocity, health, and actions being taken, stacked over 8 steps. The RL-agent is able to control 11 basic discrete high-level actions including forward and backward movement, high or low blocking, left or right attacking (with straights, uppercuts, or hooks), and is also allowed to take no action. The RL-agent is conditioned over 2 sets of 2 playing styles: aggressive or defensive and blocking or moving. During training, aggression vs. defensiveness, and blocking vs. moving are set on $[0, 1]$ scales with $(0.5, 0.5)$ representing a neutral style which is meant to balance behavior from both scales. For all styles, the agent is rewarded for winning and is penalized for every strike request. For the aggressive style, the agent is additionally rewarded for reducing opponent health. For the defensive style, the agent gets a small positive reward for each step that it remains alive, is penalized for losing health, and is rewarded for reducing opponent health (though not as much as with the aggressive style). For the blocking style, the agent is rewarded for every successful block, and for the moving style, the agent is rewarded for maintaining a positive velocity. Details are provided on the state and action spaces, reward tuning, and training in Appendix D.4.
Results MultiCriticAL allows PPO agents to successfully learn and smoothly transition between the different fighting styles. The behavior styles are distinguished visually and Figure 5 attempts to capture this through frames from captured gameplay with videos included in the Supplementary Material. Reward statistics for each style are presented in Table 3 and Figure A.2.3. Aggressive and defensive plays are the most distinct but there are also observable differences between control that prioritizes blocking and moving, with the blocking-style conditioning agents to hold their ground while moving prompts agents to move around the ring more. When comparing MultiCriticAL to MTPPO, we note that, while rewards at the extremes of style distinction are similar, MultiCriticAL appears to be better at more consistently maintaining intermediate styles (when a setting is at 0.5). Visually, it appears that MTPPO also tended to favor blocking and defense over learning more nuanced behavior (Table 6).

5.5 Discussion

While explicit separation of the learned value functions per style under the MultiCriticAL framework offers a demonstrable and consistent improvement over the single-critic baselines, it must be noted that MT-AL and MTPPO baselines do not necessarily reflect the state-of-the-art in MTRL, but they do reflect how value optimization is typically handled in contemporary literature. It is likely that utilizing some of the improvements in policy optimization techniques discussed in Section 3 may contribute to improved performance of policies trained using single-critic value-optimization, however the same may also further improve the performance of agents trained with MultiCriticAL and remains to be studied. Curiously, MN-MultiCriticAL consistently outperformed MH-MultiCriticAL on all the tested tasks, which calls into question the idea that shared representation is important and/or useful [Dewangan et al., 2018] [Yang et al., 2017] and is an aspect that may warrant further study. An interesting extension could be to extend modularization [Yang et al., 2020] [Andreas et al., 2017] to value-function optimization. Learning with curricula [Bengio et al., 2009] [Graves et al., 2017] [Mysore et al., 2019] [Andreas et al., 2017] has also been demonstrated to (sometimes dramatically) improve RL performance and multi-style/task curricula also seem like a potential natural fit for multi-critic optimization. Recent work [Mysore et al., 2021] [Andrychowicz et al., 2021] [Ilyas et al., 2020] has also demonstrated that value function capacity can be a key bottleneck in RL performance. Increasing the network capacity of a single-critic value-function may improve the efficacy of the critic, but this is not guaranteed and there are few principled ways to determine what a sufficient network capacity may be. Significantly increased computational resources may be required for the learnability of the multi-style problems under a single-critic framework. MultiCriticAL also requires increased compute during training, particularly MN-MultiCriticAL, but MH-MultiCriticAL allows for a bounded increase to computational load (< 3% as tested - see Appendix A] with an acceptable compromise in relative performance while still outperforming single-critic value learning.

6 Conclusion

We study Actor-Critic optimization in multi-style RL problems, where agents are required to behave with different styles in the same environments, our work represents one of the first explorations of the application of RL to a wider array of multi-style problems. We hypothesized that limitations of single-actor, single-critic MTRL frameworks may result in interference between the reward signals for different styles due to high similarity in visited states and to mitigate this, we proposed Multi-Critic Actor Learning (MultiCriticAL), which separates critics into per-task value-function estimators, thus avoiding interference between style rewards and reduces learning complexity by not requiring critics to learn to distinguish between tasks. MultiCriticAL consistently outperforms the single-critic MTRL baselines, achieving up to 56% improvement in multi-task performance and successfully learning multiple behavior styles where single-critic methods would fail.
As noted in Section 5, training code is based upon the open-source OpenAI Spinning Up codebase (Achiam, 2018), with modifications made to support MultiCriticAL for PPO and SAC. The modified code-base is included in the Supplementary Material for this paper, along with the code for the custom environments for Path Following (presented in Section 5.1) and Pong with rotatable paddles (presented in Section 5.2). The README files included with the code detail installation and use of the code. For the Sonic the Hedgehog games, reproducing results would require acquiring the games’ ROM files and linking them to an install of gym-retro, as detailed by Nichol et al. (2018). Also included in Appendices D and E are additional details for the setup of the training environments discussed in the main body of the paper as well as training configurations used. We recognize that tests on the UFC environment may not be independently reproducible due to requiring access to proprietary development software and we therefore mainly discuss it as a demonstration of the practical utility for our proposed MultiCriticAL method, rather than as a benchmark task.

References


A ADDITIONAL DATA AND RESULTS

A.1 LEARNING CURVES

A.1.1 PATH FOLLOWING

Figure 6: Learning Curves for all tested SAC variants on the Path Following benchmark.

Figure 7: Learning Curves for all tested PPO variants on the Path Following benchmark.

A.1.2 PONG

Figure 8: Learning Curves for all tested SAC variants on the Pong benchmark.
Figure 9: Learning Curves for all tested PPO variants on the Pong benchmark.

A.1.3 Sonic the Hedgehog

Figure 10: Average Learning Curves the Sonic the Hedgehog benchmark games. Represented here is the averaged reward over the 17 levels of the game as training progresses. We chose this representation in the interest of providing a more compact representation instead of including 34 separate figures, with one for each level, as that would both require a lot of space and be more difficult to parse.
### A.2 Alternative Results Representation

#### A.2.1 Pong

![Box and Whisker plots for Pong Results presented in Section 5](image)

#### A.2.2 Sonic the Hedgehog

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Level</th>
<th>Single-level Policy</th>
<th>MT</th>
<th>PPO</th>
<th>MN-MultiCritical</th>
<th>MH-MultiCritical</th>
</tr>
</thead>
<tbody>
<tr>
<td>GHZ1</td>
<td>GreenHillZone.Act1</td>
<td>37.41 ± 15.84</td>
<td>38.70 ± 23.41</td>
<td>42.30 ± 5.32</td>
<td>40.72 ± 6.04</td>
<td></td>
</tr>
<tr>
<td>GHZ2</td>
<td>GreenHillZone.Act2</td>
<td>17.60 ± 9.66</td>
<td>26.40 ± 10.62</td>
<td>42.26 ± 2.07</td>
<td>32.63 ± 6.01</td>
<td></td>
</tr>
<tr>
<td>GHZ3</td>
<td>GreenHillZone.Act3</td>
<td>7.22 ± 4.07</td>
<td>9.37 ± 6.58</td>
<td>38.85 ± 25.18</td>
<td>12.02 ± 5.54</td>
<td></td>
</tr>
<tr>
<td>MZ1</td>
<td>MarbleZone.Act1</td>
<td>28.03 ± 5.45</td>
<td>20.20 ± 11.30</td>
<td>39.04 ± 13.30</td>
<td>40.40 ± 8.45</td>
<td></td>
</tr>
<tr>
<td>MZ2</td>
<td>MarbleZone.Act2</td>
<td>18.86 ± 5.23</td>
<td>16.79 ± 6.60</td>
<td>22.84 ± 1.70</td>
<td>22.92 ± 0.11</td>
<td></td>
</tr>
<tr>
<td>MZ3</td>
<td>MarbleZone.Act3</td>
<td>27.31 ± 2.36</td>
<td>22.64 ± 7.60</td>
<td>29.82 ± 0.68</td>
<td>28.50 ± 0.17</td>
<td></td>
</tr>
<tr>
<td>SYZ1</td>
<td>SpringYardZone.Act1</td>
<td>6.29 ± 2.71</td>
<td>7.82 ± 6.50</td>
<td>10.02 ± 5.34</td>
<td>9.16 ± 5.14</td>
<td></td>
</tr>
<tr>
<td>SYZ2</td>
<td>SpringYardZone.Act2</td>
<td>5.55 ± 3.71</td>
<td>11.59 ± 2.85</td>
<td>18.95 ± 3.41</td>
<td>13.47 ± 6.06</td>
<td></td>
</tr>
<tr>
<td>LZ1</td>
<td>LabyrinthZone.Act1</td>
<td>22.50 ± 3.18</td>
<td>22.04 ± 5.00</td>
<td>22.00 ± 6.40</td>
<td>25.16 ± 12.77</td>
<td></td>
</tr>
<tr>
<td>LZ2</td>
<td>LabyrinthZone.Act2</td>
<td>29.52 ± 0.40</td>
<td>29.65 ± 0.19</td>
<td>29.59 ± 0.31</td>
<td>29.46 ± 0.06</td>
<td></td>
</tr>
<tr>
<td>LZ3</td>
<td>LabyrinthZone.Act3</td>
<td>25.70 ± 4.02</td>
<td>22.04 ± 0.99</td>
<td>26.33 ± 0.13</td>
<td>25.82 ± 0.19</td>
<td></td>
</tr>
<tr>
<td>SLZ1</td>
<td>StarLightZone.Act1</td>
<td>35.03 ± 0.51</td>
<td>35.15 ± 1.06</td>
<td>47.80 ± 12.38</td>
<td>39.79 ± 4.38</td>
<td></td>
</tr>
<tr>
<td>SLZ2</td>
<td>StarLightZone.Act2</td>
<td>6.76 ± 0.14</td>
<td>13.05 ± 4.62</td>
<td>18.25 ± 0.30</td>
<td>17.04 ± 3.21</td>
<td></td>
</tr>
<tr>
<td>SLZ3</td>
<td>StarLightZone.Act3</td>
<td>14.47 ± 6.95</td>
<td>26.37 ± 5.01</td>
<td>30.40 ± 6.39</td>
<td>30.20 ± 1.00</td>
<td></td>
</tr>
<tr>
<td>SRBZ1</td>
<td>ScrapBrainZone.Act1</td>
<td>8.03 ± 1.65</td>
<td>10.74 ± 4.52</td>
<td>9.92 ± 4.13</td>
<td>11.68 ± 6.11</td>
<td></td>
</tr>
<tr>
<td>SRBZ2</td>
<td>ScrapBrainZone.Act2</td>
<td>12.52 ± 0.00</td>
<td>8.54 ± 4.27</td>
<td>11.61 ± 1.99</td>
<td>11.25 ± 1.51</td>
<td></td>
</tr>
</tbody>
</table>

| **Average**  | 19.42 ± 2.04       | 20.48 ± 2.24        | 27.39 ± 2.65 | 24.73 ± 1.31 |
### A.2.3 UFC Reward and Action Statistics

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Level</th>
<th>Single-level Policy</th>
<th>MTPPO</th>
<th>MN-MultiCriticAL</th>
<th>MH-MultiCriticAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EHZ1</td>
<td>EmeraldHillZone.Act1</td>
<td>48.58 ± 1.10</td>
<td>61.72 ± 1.08</td>
<td>65.82 ± 1.48</td>
<td>44.10 ± 0.07</td>
</tr>
<tr>
<td>EHZ2</td>
<td>EmeraldHillZone.Act2</td>
<td>21.82 ± 2.37</td>
<td>18.43 ± 10.09</td>
<td>28.31 ± 6.27</td>
<td>23.65 ± 13.28</td>
</tr>
<tr>
<td>CPZ1</td>
<td>ChemicalPlantZone.Act1</td>
<td>26.58 ± 3.29</td>
<td>28.21 ± 2.01</td>
<td>37.48 ± 10.18</td>
<td>33.65 ± 11.64</td>
</tr>
<tr>
<td>CPZ2</td>
<td>ChemicalPlantZone.Act2</td>
<td>28.82 ± 4.32</td>
<td>18.84 ± 8.90</td>
<td>24.60 ± 9.85</td>
<td>27.01 ± 5.00</td>
</tr>
<tr>
<td>ARZ1</td>
<td>AquaticRunZone.Act1</td>
<td>15.30 ± 7.31</td>
<td>5.85 ± 0.00</td>
<td>24.28 ± 5.56</td>
<td>19.18 ± 9.24</td>
</tr>
<tr>
<td>ARZ2</td>
<td>AquaticRunZone.Act2</td>
<td>37.40 ± 6.89</td>
<td>24.12 ± 2.85</td>
<td>39.41 ± 7.30</td>
<td>33.15 ± 10.23</td>
</tr>
<tr>
<td>CNZ</td>
<td>CasinoNightZone.Act1</td>
<td>17.12 ± 0.97</td>
<td>12.02 ± 3.51</td>
<td>20.34 ± 5.01</td>
<td>15.65 ± 1.99</td>
</tr>
<tr>
<td>HTZ1</td>
<td>HillTopZone.Act1</td>
<td>7.82 ± 0.20</td>
<td>6.36 ± 2.72</td>
<td>7.76 ± 0.85</td>
<td>5.65 ± 0.57</td>
</tr>
<tr>
<td>HTZ2</td>
<td>HillTopZone.Act2</td>
<td>6.44 ± 4.23</td>
<td>4.22 ± 2.81</td>
<td>19.89 ± 2.96</td>
<td>21.30 ± 3.08</td>
</tr>
<tr>
<td>MCZ1</td>
<td>MysticCaveZone.Act1</td>
<td>10.25 ± 0.34</td>
<td>7.42 ± 1.14</td>
<td>9.74 ± 1.44</td>
<td>10.23 ± 0.17</td>
</tr>
<tr>
<td>MCZ2</td>
<td>MysticCaveZone.Act2</td>
<td>8.57 ± 2.31</td>
<td>5.47 ± 0.00</td>
<td>8.75 ± 3.32</td>
<td>11.96 ± 1.59</td>
</tr>
<tr>
<td>OOO1</td>
<td>OilOceanZone.Act1</td>
<td>14.95 ± 3.87</td>
<td>8.65 ± 4.18</td>
<td>19.14 ± 2.58</td>
<td>13.04 ± 4.28</td>
</tr>
<tr>
<td>OOO2</td>
<td>OilOceanZone.Act2</td>
<td>10.20 ± 0.90</td>
<td>9.92 ± 2.11</td>
<td>13.72 ± 4.16</td>
<td>14.06 ± 3.22</td>
</tr>
<tr>
<td>MZ1</td>
<td>MetropolisZone.Act1</td>
<td>14.64 ± 3.25</td>
<td>6.07 ± 1.94</td>
<td>14.39 ± 2.58</td>
<td>14.03 ± 3.52</td>
</tr>
<tr>
<td>MZ2</td>
<td>MetropolisZone.Act2</td>
<td>12.79 ± 2.68</td>
<td>11.50 ± 2.53</td>
<td>13.12 ± 0.37</td>
<td>16.90 ± 7.06</td>
</tr>
<tr>
<td>MZ3</td>
<td>MetropolisZone.Act3</td>
<td>6.49 ± 2.27</td>
<td>10.31 ± 1.59</td>
<td>11.75 ± 2.29</td>
<td>17.34 ± 5.16</td>
</tr>
<tr>
<td>WFZ</td>
<td>WingFortressZone</td>
<td>27.43 ± 0.88</td>
<td>8.74 ± 0.41</td>
<td>24.81 ± 9.08</td>
<td>21.20 ± 5.53</td>
</tr>
</tbody>
</table>

| Average | 15.90 ± 1.74 | 13.74 ± 1.06 | 21.11 ± 2.44 | 19.22 ± 1.96 |

![Figure 12: Sonic Results from Tables 4 and 5 visualized](image)

We observe from Figure A.2.3 and Table 6 that MultiCriticAL more consistently learns delineated styles that are consistent with the behaviors that the rewards try to encourage. Note that, with our method, it is, in fact, the aggressive style that wins most fights consistently and the defensive style learns to block and move to not lose as quickly, whereas MTPPO tends seems to equate defensive play with just blocking, and this also contributes to a higher rate of losing. We see as well that the intermediate settings correspond to a more balanced blend of action probabilities between extremes when compared to MTPPO.
As an exercise in curiosity, we also consider what would happen if we attempted to derive intermediate styles by interpolating between actions in the Pong game. This was meant to test an approach similar to the multi-policy action composition technique used by Peng et al. (2018). Unlike in their case where actions were weighted according to each actor’s estimated critic value, we instead directly interpolate between actions proposed by pre-trained aggressive and defensive agents. We take this approach as the training for either style would not offer the value functions any meaningful information about the other unseen style(s), prohibiting the form of value comparison employed by Peng et al. We hypothesized that it is possible and even likely that policies would destructively interfere with each other if the longer-term goals of constituent policies conflict with each other and this is exactly what was observed in testing. Except for intermediate styles very close to the learned extremes, interpolating between actions proposed by the aggressive and defensive agents caused a near-complete failure in the composite policy’s control.

Composite policies were built by pairing a randomly selected pre-trained aggressive policy with a randomly selected defensive policy. On a (0.5,0.5) ‘middle’ setting, SAC composite policies achieved an average win-rate of 30% ± 5% (compared to the approximately 95% and 50% of the constituent policies) with an average Play-time of 213 ± 89 steps (often losing the game quickly). Similarly, composite PPO also achieves a low average win-rate of 15% ± 3% and similarly lose the game quickly. A high win-rate would not be expected, as the defensive policies are explicitly trained to avoid winning, but both policy classes are trained to avoid losing. Given the high loss-rate, composing policies in this way, for this multi-style task, can only be regarded as a failure. By contrast however, as we showed in Table 2 MultiCriticAL is capable of learning to interpolate between styles, both when and not explicitly trained to do so.

This experiment, by no means, constitutes exhaustive testing, and cannot be used to claim that all, or even most, forms of multi-policy policy composition would be ineffective and is therefore not included in the main body of the paper. We do believe however that it helps highlight a circumstance where policy composition can fail in cases where long-term strategies for different target behavior styles may interfere. Considered more generally, there are no guarantees that actions proposed by distinct policies would be similar enough, that a weighted sum over them would even make sense, as cooperation between policies is not something that is explicitly trained for. One might benefit from explicit conditioning, though this was outside the scope of our work.
B RELATING MULTICRITICAL TO DQNS AND MULTIT-Q LEARNING

The idea of using multiple Q functions to guide the training of a single policy is not strictly new to the broader context of RL and has been used in notably single-task RL for improving the stability of learned value functions. These include techniques such as Double Q-learning [Hasselt, 2010], Van Hasselt et al. [2016], Dueling Q-networks [Wang et al., 2016], and Bootstrapped DQNs [Osband et al., 2016; Steckelmacher et al., 2019]. There are also applications in multi-task learning, though we only encountered them in works considering DQN frameworks that, in effect, learn task-specific Q values for a single policy in DQN architectures [Rusu et al., 2016; 2015; D’Eramo et al., 2020]. As DQNs jointly encode both the action policies and the value representation in the same network, however, they do not share the practical benefits offered by Actor-Critic methods (and consequently MultiCriticAL) in separating the actors and critics. We believe that, practically, this is a meaningful distinction as DQN methods are (i) limited to discrete action spaces and (ii) need the full representational and computational complexity of the value estimation to be preserved at inference time after training, which is computationally inefficient. As shown by Andrychowicz et al. (2021) and Mysore et al. (2021a), oftentimes, it appears that the value function often bears the higher burden of required complexity to enable successful policy learning. Noting this, there is a significant practical advantage to being able to just train smaller, less computationally expensive actor-networks of actor-critic methods and disregard the value estimation networks at inference time. Given that our work is largely motivated by application to video games, where runtime resource use is an important consideration, this was an important distinction.

C MULTI-NETWORK VS. MULTI-HEADED TRAINING FRAMEWORK OVERVIEW

Figure 14: MN-MultiCriticAL and MH-MultiCriticAL frameworks compared. The main differences between the multi-network (MN) and multi-headed (MH) is in how the critic functions are ultimately represented by the neural network(s). In the MN case, separate and independent networks are maintained for each style/task value function, whereas the MH configuration allows value functions to share a common backbone and instead utilizes different network heads (i.e. output nodes) for each learned value. While the former appears to offer superior performance in most of the tested cases, the latter can be less computationally expensive and may incur reduced training costs. It should be noted, however, that neither architecture needs to impact the actor, thus preserving run-time inference cost.
D  ADDITIONAL ENVIRONMENT SETUP DETAILS

D.1  PATH FOLLOWING

**Observed State**  Agents observe a 4-dimensional state with the current \((x, y)\) coordinates of the agent in space and the current phase \(\rho \in [0, 1]\) represented by \([\sin(2\rho\pi), \cos(2\rho\pi)]\)

**Goal generation**  Goals, \(g\) are generated based on the current phase of motion. They are as follows:

- **Circle:** \(g(\rho) = [\sin(2\rho\pi), \cos(2\rho\pi)]\) \(\forall \rho \in [0, 1]\)
- **Square:** \(g(\rho) = \begin{cases} [\rho - 0.5, 0.5] & \rho \leq 0.25 \\ [0.5, 1.5 - 4\rho] & 0.25 \leq \rho \leq 0.5 \\ [2.5 - 4\rho, -0.5] & 0.5 \leq \rho \leq 0.75 \\ [-0.5, 4\rho - 3.5] & 0.75 \leq \rho \leq 1 \end{cases}\)
- **Triangle:** \(g(\rho) = \begin{cases} [1.5\rho, 0.5 - 3\rho] & \rho \leq 1/3 \\ [1.5 - 3\rho, -0.5] & 1/3 \leq \rho \leq 2/3 \\ [-1.5 + 1.5\rho, 1.5\rho - 1.5] & 2/3 \leq \rho \leq 1 \end{cases}\)

**Reward design**  For current state \(s\) and desired state \(g\): \(r = 1 - \log(|s - g|) / \log(\max(\Delta s))\) where \(\max(\Delta s)\) is the difference between the extents of the space bounds.

D.2  PONG

**Observed State**  Agents observe an 8-dimensional state with the current vertical \(y\) positions of both the player and opponent, the player’s velocity, the paddle angle, and the ball’s \((x, y)\) position and velocity.

**Reward design**  Aggressive play is conditioned with a sparse reward: +1 on a win and -1 on loss. Defensive play is conditioned with dense rewards: -1 on loss, -1 for bouncing the ball against walls, +0.5 for receiving the ball, +0.1 if the opponent successfully receives the ball, and +0.1 for every time-step of play. Defensive rewards encourage avoiding losing and ensuring that the opponent can receive the ball in order to prolong the game.

D.3  SONIC THE HEDGEHOG

**Reward scaling**  As recommended by Nichol et al. (2018), we scale rewards by a factor of 0.01 in order to encourage more stable learning with PPO.

D.4  EA UFC DEVELOPMENT ENVIRONMENT

**Environment**  The UFC environment is an experimental build of UFC. During training simulation, superfluous elements of the game, such as audio, spectators, etc. are disabled. Socket communication is used to interface between the game-code and the python-based RL algorithms. The environment wrapper controls the game simulation and extracts and processes state information before communicating it through the socket to the python RL code and interprets requests from the python code to trigger inputs in the game. During training, agents are trained for 240 epochs with 4000 sample interactions collected per epoch. Training is run in increments of 30 epochs to improve simulation and memory management stability. Episodes are terminated when the agent runs out the max episode length or if the health differential between the agent and opponent is over 40 units.
State and Action Space for the UFC environment

The environment returns the following data as 30-dimensional observations for the RL agent:

- Position: (X,Z) coordinates in the ring
- Velocity: scalar of trajectory velocity
- Health: [0,100] float value
- Action active: high-level action indicator
- Strike type: indicator of type of strike performed
- Strike segment: indicates the current stage of striking
- Ticks to event: time until next event in game ticks
- Target hit location: the region of the opponent’s body the agent is targeting (upper or lower)
- Block successful: indicates when a strike is successfully blocked

The agent is allowed to take the following actions:

- No action
- (Strike) Straight: left or right, single request
- (Strike) Uppercut: left or right, single request
- (Strike) Hook: left or right, single request
- Block: high or low, needs to be held to sustain blocking
- Move: forward or backward relative to current opponent position

Reward Design

The reward breakdown for the environment is as follows:

- Standard to all styles – Victory reward: +1; Strike request penalty: -0.001
- Aggressive style – successful attack reward: +|ΔHealth\_opponent|/100
- Defensive style – survival reward: +0.02 per step; attack reward: +|ΔHealth\_opponent|/200, health loss penalty: −ΔHealth\_agent/50
- Blocking style – block reward: +0.01 per successful block
- Movement style – velocity reward: +0.05v

E Training Configuration

Training code is based on OpenAI’s Spinning Up (Achiam, 2018) and is provided in the zip folder included as additional supplementary material with our main paper submission.

With the exception of hidden layer configuration, we use the same default hyperparameters for each of the algorithms used. Different hidden layer configurations were selected for each environment such that they would not be needlessly large but would consistently solve the problem in the single-style cases.

Network Configuration

Hidden layer configurations for both the actors and critics per environment:

- Path following: PPO [64,64], SAC [8,8]
- Pong: PPO [64,64], SAC [32,32]
- Sonic the Hedgehog 1 and 2: PPO [256,256]. Note additionally that the CNN feature extractor is shared between the actors and all critics in any particular training run, though they are not shared between separate training runs. The CNN feature extractor was configured based on the the extractor used by Mnih et al. (2015) as it is the same configuration used in the Retro-contest (Nichol et al., 2018).

The above network configurations correspond to the following numbers of trainable weights for each of the critic network(s).
Table 7: Trainable Critic parameters per task

<table>
<thead>
<tr>
<th>Task</th>
<th>Single-style</th>
<th>MT w/ one-hot</th>
<th>MN-MultiCriticAL</th>
<th>MH-MultiCriticAL</th>
<th>MH-MultiCriticAL vs. MT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Following</td>
<td>137</td>
<td>161</td>
<td>483</td>
<td>163</td>
<td>↑ 1.24%</td>
</tr>
<tr>
<td>Pong</td>
<td>5061</td>
<td>7109</td>
<td>14218</td>
<td>7164</td>
<td>↑ 0.77%</td>
</tr>
<tr>
<td>PPO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path Following</td>
<td>4545</td>
<td>4737</td>
<td>14211</td>
<td>4867</td>
<td>↑ 2.74%</td>
</tr>
<tr>
<td>Pong</td>
<td>68333</td>
<td>76545</td>
<td>153090</td>
<td>76802</td>
<td>↑ 0.33%</td>
</tr>
<tr>
<td>Sonic</td>
<td>197377</td>
<td>201729</td>
<td>342939</td>
<td>205841</td>
<td>↑ 2.04%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Misc. details  Additionally, a frame wrapper is used with Pong and the Sonic games to augment state information with information from the last 4 observed states. This wrapper is based on the FrameStack wrapper provided in OpenAI Baselines (Dhariwal et al., 2017).

Training time was determined by running training until performance plateaued, with an additional buffer given to multi-task training to account for the multi-task problem setting, with initial tests conducted to ensure that performance indeed plateaued.

Run configurations  Configurations for training per environment are as follows:

Path following
- Single-style PPO – Trained for 100 epochs with 4000 steps per epoch and 90 steps per episode
- Single-style SAC – Trained for 50 epochs with 4000 steps per epoch and 90 steps per episode
- Multi-style MTPPO – Trained for 300 epochs with 4000 steps per epoch and 90 steps per episode
- Multi-style MTSAC – Trained for 150 epochs with 4000 steps per epoch and 90 steps per episode
- Multi-style MultiCriticAL-PPO – Trained for 300 epochs with 4000 steps per epoch and 90 steps per episode
- Multi-style MultiCriticAL-SAC – Trained for 150 epochs with 4000 steps per epoch and 90 steps per episode

Pong with rotatable paddles
- Single-style PPO – Trained for 150 epochs with 4000 steps per epoch and variable length episodes
- Single-style SAC – Trained for 50 epochs with 4000 steps per epoch and variable length episodes
- Multi-style MTPPO – Trained for 250 epochs with 4000 steps per epoch and variable length episodes
- Multi-style MTSAC – Trained for 150 epochs with 4000 steps per epoch and variable length episodes
- Multi-style MultiCriticAL-PPO – Trained for 250 epochs with 4000 steps per epoch and variable length episodes
- Multi-style MultiCriticAL-SAC – Trained for 150 epochs with 4000 steps per epoch and variable length episodes

Sonic the Hedgehog (both 1 and 2)
- Single-style PPO – Trained for 30 epochs with 10000 steps per epoch and up to 2500 steps per episode, or until Sonic’s death, whichever occurs first
- Multi-style MTPPO – Trained for 300 epochs with 10000 steps per epoch and up to 2500 steps per episode, or until Sonic’s death, whichever occurs first
- Multi-style MultiCriticAL-PPO – Trained for 300 epochs with 10000 steps per epoch and up to 2500 steps per episode, or until Sonic’s death, whichever occurs first