# Benchmarking Massively Parallelized Multi-Task Reinforcement Learning for Robotics Tasks

**Anonymous authors**
Paper under double-blind review

**Keywords:** Multi-Task Learning, Reinforcement Learning, Robotics.

## Summary

Multi-task Reinforcement Learning (MTRL) has emerged as a critical training paradigm for applying reinforcement learning (RL) to a set of complex real-world robotic tasks, which demands a generalizable and robust policy. At the same time, *massively parallelized training* has gained popularity, not only for significantly accelerating data collection through GPU-accelerated simulation but also for enabling diverse data collection across multiple tasks by simulating heterogeneous scenes in parallel. However, existing MTRL research has largely been limited to off-policy methods like SAC in the low-parallelization regime. MTRL could capitalize on the higher asymptotic performance of on-policy algorithms, whose batches require data from current policy, and as a result, take advantage of massive parallelization offered by GPU-accelerated simulation. To bridge this gap, we introduce a massively parallelized **M**ulti-**T**ask **Bench**mark for robotics (MTBench), an open-sourced benchmark featuring a broad distribution of 50 manipulation tasks and 20 locomotion tasks, implemented using the GPU-accelerated simulator IsaacGym. MTBench also includes four base RL algorithms combined with seven state-of-the-art MTRL algorithms and architectures, providing a unified framework for evaluating their performance. Our extensive experiments highlight the superior speed of evaluating MTRL approaches using MTBench, while also uncovering unique challenges that arise from combining massive parallelism with MTRL.

## Contribution(s)

1. This paper introduces MTBench, a unified GPU-accelerated benchmark for massively parallelized multi-task reinforcement learning (MTRL) in two robotics settings, manipulation and locomotion.
   **Context:** Existing robotics MTRL benchmarks, such as Meta-World (Yu et al., 2021), have impractically long experimental runtimes, hindering the development and reproducibility of MTRL research. Other GPU-accelerated benchmarks for robotics do not support MTRL out of the box. We address both of these concerns with our end-to-end MTRL benchmark.
2. This paper conducts comprehensive experiments to evaluate all aspects of MTRL, including base RL algorithms, gradient manipulation methods, and neural network architectures.
   **Context:** We confirm whether the reliance on off-policy methods in the MTRL literature holds in the massively parallel regime, and then evaluate a suite of MTRL schemes using on-policy methods across our evaluation settings.
3. This paper presents four key observations on applying existing MTRL schemes to massively parallelized training in robotics. These insights guide the selection of MTRL schemes and inform future research directions.
   **Context:** Massively parallelized training is emerging as a popular paradigm, introducing unique challenges for existing RL methods (Singla et al., 2024; D'Oro et al., 2022; Li et al., 2023; Gallici et al., 2024). However, MTRL development has yet to leverage this paradigm.

# Benchmarking Massively Parallelized Multi-Task Reinforcement Learning for Robotics Tasks

**Anonymous authors**
Paper under double-blind review

## Abstract

Multi-task Reinforcement Learning (MTRL) has emerged as a critical training paradigm for applying reinforcement learning (RL) to a set of complex real-world robotic tasks, which demands a generalizable and robust policy. At the same time, *massively parallelized training* has gained popularity, not only for significantly accelerating data collection through GPU-accelerated simulation but also for enabling diverse data collection across multiple tasks by simulating heterogeneous scenes in parallel. However, existing MTRL research has largely been limited to off-policy methods like SAC in the low-parallelization regime. MTRL could capitalize on the higher asymptotic performance of on-policy algorithms, whose batches require data from current policy, and as a result, take advantage of massive parallelization offered by GPU-accelerated simulation. To bridge this gap, we introduce a massively parallelized **M**ulti-**T**ask **Bench**mark for robotics (MTBench), an open-sourced benchmark featuring a broad distribution of 50 manipulation tasks and 20 locomotion tasks, implemented using the GPU-accelerated simulator IsaacGym. MTBench also includes four base RL algorithms combined with seven state-of-the-art MTRL algorithms and architectures, providing a unified framework for evaluating their performance. Our extensive experiments highlight the superior speed of evaluating MTRL approaches using MTBench, while also uncovering unique challenges that arise from combining massive parallelism with MTRL.

## 1 Introduction

Deep reinforcement learning has been successfully applied to a wide range of decision-making tasks, including Atari games (Mnih et al., 2013), the game of Go (Silver et al., 2016), and continuous control tasks (Hwangbo et al., 2019; Wurman et al., 2022). While these applications have achieved remarkable task-specific performance, recent research trends have shifted towards developing general-purpose agents capable of solving multiple tasks or adapting to diverse environments (Cobbe et al., 2020; Kirk et al., 2023; Park et al., 2024). This transition is partly motivated by the demands of real-world robotics applications, where versatility and robustness are essential. For example, tabletop manipulation often requires acquiring multiple skills to accomplish complex tasks (Pinto & Gupta, 2016; Yu et al., 2021) and legged locomotion demands adaptability to traverse challenging terrains (Lee et al., 2020; Liang et al., 2024).

To facilitate the learning of a general-purpose robotic agent, massively parallelized training (»1000 simulations) has gained popularity with the advancement of GPU-accelerated simulators (Liang et al., 2018b; Freeman et al., 2021; Makoviychuk et al., 2021; Mittal et al., 2023; Tao et al., 2024; Zakka et al., 2025). These simulators have significantly mitigated hardware and runtime constraints for learning *single tasks*, reducing experiment durations from days to minutes (Liang et al., 2018b; Rudin et al., 2022). However, in the multi-task setting, no out-of-the-box solution exists to allocate a fixed number of environments per task on a single GPU, allowing for simultaneous diverse data collection and end-to-end MTRL training. Additionally, massively parallelized online batched RL

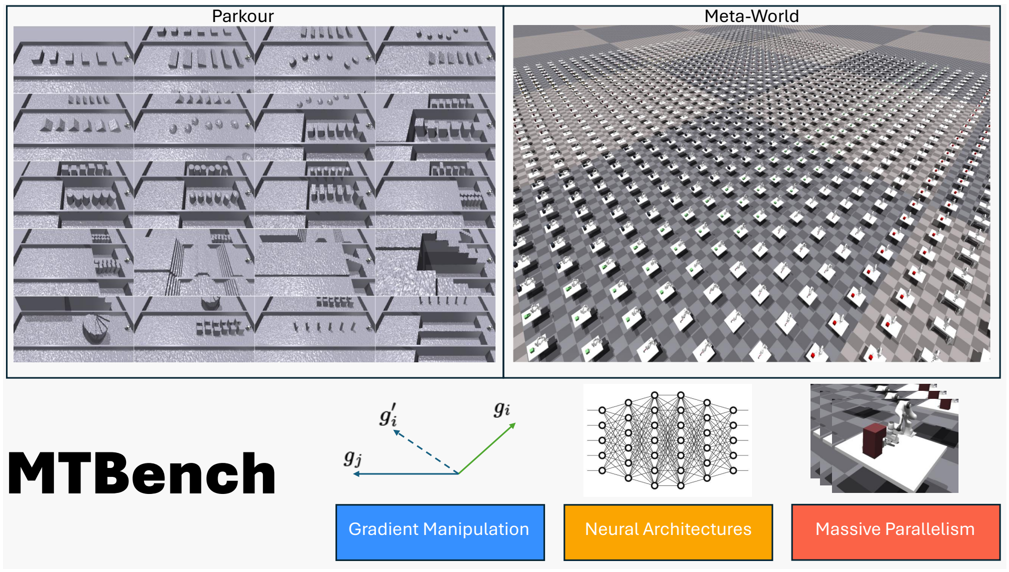

Figure 1: MTBench has two robotics settings: Parkour and Meta-World that leverage massive parallelism for MTRL and offer a suite of MTRL schemes introduced over the years.

introduces new, non-trivial algorithmic challenges. For example, on-policy methods like PPO reach a saturation point beyond which additional parallelization no longer improves performance (Singla et al., 2024). Meanwhile, off-policy methods such as SAC and Q-Learning become unstable, losing their sample efficiency compared to on-policy methods as interaction with parallel environments unbalances the replay ratio (D'Oro et al., 2022; Li et al., 2023; Gallici et al., 2024).

On the other hand, learning general-purpose robotic agents has also motivated multi-task RL (MTRL), which aims to learn a single policy that maximizes average performance across multiple tasks. By leveraging task similarities (Pinto & Gupta, 2016), MTRL often enhances sample efficiency, requiring fewer transitions to match the performance of single-task counterparts. Prior research has primarily focused on addressing optimization challenges introduced by multiple learning signals, either from a gradient-based perspective (Yu et al., 2020; Liu et al., 2024; 2023) or through neural architecture design (Yang et al., 2020; Sodhani et al., 2021; Sun et al., 2022; Hendawy et al., 2024). However, existing MTRL approaches have focused on using off-policy methods in low-parallelization settings ($\leq$ 500 workers) (Espeholt et al., 2018; Liang et al., 2018a; James et al., 2020). With massive parallelization applied to MTRL, we no longer need to deal with how to distribute experience collection and learning, instead utilizing on-policy algorithms, whose batches require data from current experience and as a result, take advantage of the parallelization offered by GPU-based simulators.

To support large-scale MTRL experiments and advance the development of general-purpose robotic agents, we introduce a massively parallelized **M**ulti-**T**ask **Bench**mark for robotics (MTBench). This open-source benchmark includes a diverse set of 50 manipulation tasks and 20 locomotion tasks (top row of Figure 1), implemented using the GPU-accelerated simulator IsaacGym. Each task allows for procedurally generating infinitely many variations by modifying factors such as initial states and terrain configurations. Additionally, MTBench integrates four base RL algorithms with seven state-of-the-art MTRL algorithms and architectures, providing a unified framework to evaluate their performances.

Based on our experiments, we highlight the following major observations:

**(O1) On-Policy > Off-Policy:** Choosing between on-policy RL methods or off-policy methods affects performance more than the MTRL scheme applied in massively parallel training. Off-policy RL's asymptotic performance struggles to match on-policy RL in this regime.

**(O2) Prioritize Wall-Clock Time over Sample Efficiency:** In the massively parallel regime, wall-clock efficiency is more critical than sample efficiency, as experience collection scales easily with more GPUs.

**(O3) Value Learning is the Key Bottleneck in MTRL:** Multi-task RL struggles primarily with value estimation rather than policy learning, as gradient conflicts mostly impact the critic function.

**(O4) Curriculum Learning is Crucial for Sparse-Reward Tasks:** MTRL alone does not help exploration in sparse-reward tasks; curriculum learning is essential for overcoming early stagnation.

## 2 Background

### 2.1 GPU Accelerated Simulation

Traditionally, simulators used for online RL rely on the coordination between CPU and GPU where the CPU handles physics simulation and observation/reward calculations while the GPU handles neural network training and inference, leading to frequent slow memory transfers between the two many times during the RL training process. Now, GPU-accelerated simulators provide access to the results of physics simulation on the GPU, and as a result, we have all relevant data - observations, actions, and rewards - remaining on the GPU throughout the learning process. This development allows for massive parallelization and as a result, dramatically reduces MTRL training time from days or weeks on thousands of CPU cores to just hours on a single GPU.

Specifically, NVIDIA IsaacGym offers a Tensor API that directly exposes the physics state of the world in Python, so we can directly populate and manage massively parallelized heterogeneous scenes for all tasks, avoiding the communication overhead of synchronizing experience collection and neural network training across distributed systems (Espeholt et al., 2018; Nair et al., 2015).

### 2.2 Multi-task reinforcement learning

A finite horizon, discrete-time MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \mu, \gamma)$, where $\mathcal{S} \in \mathbb{R}^n$ denotes the continuous state space, $\mathcal{A} \in \mathbb{R}^m$ denotes the continuous action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ denotes the stochastic transition dynamics, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function, $\mu : \mathcal{S} \to \Delta(\mathcal{S})$ denotes the initial state distribution, and $\gamma \in (0, 1]$ is the discount factor. A policy parameterized by $\theta$, $\pi_\theta(a_t|s_t) : \mathcal{S} \to \Delta(\mathcal{A})$, is a probability distribution over actions conditioned on the current state. RL learns a policy $\pi_\theta$ such that it maximizes the expected cumulative discounted return $J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$ where $a_t \sim \pi_\theta$.

**Problem statement** Each task $\tau$ is sampled the task distribution $p(\mathcal{T})$ is a different MDP $\mathcal{M}^\tau = (\mathcal{S}^\tau, \mathcal{A}^\tau, \mathcal{P}^\tau, r^\tau, \mu^\tau, \gamma^\tau)$. MTRL learns a single policy $\pi_\theta$ that maximizes the expected cumulative discounted return averaged across all tasks $J(\theta) = \sum_{\tau \in \mathcal{T}} J_\tau(\theta)$. The only restriction we place upon $\mathcal{M}^\tau$ is that their union shares a universal state space $\mathcal{S}$ and by appending a task embedding to the state, we give the policy the ability to distinguish what task each observation belongs to.

A change in any part of a $\mathcal{M}^\tau$ constitutes what it means to define a new task. In locomotion, each task from $p(\mathcal{T})$ would be associated with a different goal to reach *in the same control setting*, so only $r^\tau$ would differ across tasks. In tabletop manipulation like Meta-World, the tasks range from basic skills like pushing and grasping to more advanced skills combining these basic skills, so the goals ($r^\tau$) and state spaces ($\mathcal{S}^\tau$) vary across tasks.

## 3 Benchmark

The proposed massively parallelized multitask RL benchmark provides a unified framework for simulating two key robotics task categories: manipulation and locomotion, within the IsaacGym simulator. For manipulation, we incorporate 50 tasks from Meta-World (Yu et al., 2021), chosen for their simplicity, task diversity, and well-designed shaping rewards. The locomotion domain includes 20 diverse quadrupedal Parkour tasks from Eurikaverse (Liang et al., 2024), the most comprehensive Parkour benchmark, encompassing a wide range of established locomotion challenges. The following sections provide a detailed overview of these task domains and the evaluation protocols.

## 3.1 Meta-World

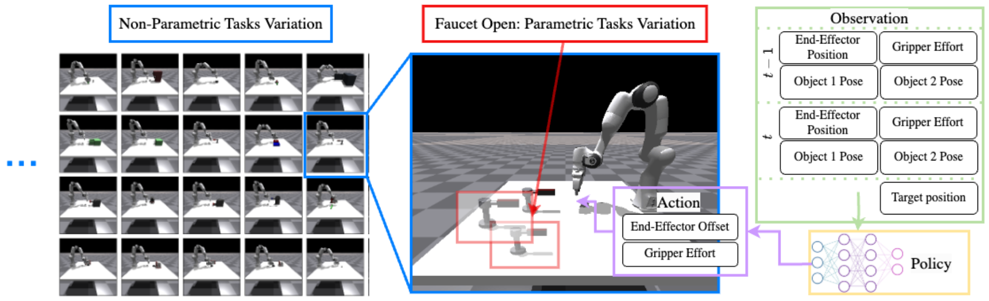

Figure 2: Illustrations of non-parametric tasks variation, parametric tasks variation of Faucet Open, and the observation and action space of the RL agents in the Meta-World benchmark.

**Task Descriptions:** Meta-World consists of 50 tabletop manipulation tasks that require a simulated one-armed robot (Franka Robotics, 2017) to interact with one or two objects in various ways, such as pushing, picking, and placing. Within each task, Meta-World also provides parametric goal variation over the initial object position and target position. Each task has a pre-defined success criterion (Appendix C.1). Our reimplementation of Meta-World makes necessary changes by updating Sawyer to Franka Emika Panda and tuning the reward function of each task to ensure that the tasks are individually solvable.

**Observation and Action Spaces:** Despite sharing a common state space dimensionality, the semantic meaning of certain dimensions varies across tasks. The state representation comprises the end-effector's 3D position in $\mathbb{R}^3$, the normalized gripper effort in $\mathbb{R}^1$, the object 3D positions from two objects in $\mathbb{R}^6$, and the quaternion representation of the two objects' orientation in $\mathbb{R}^8$. For tasks involving a single object, the state dimensions corresponding to a second object are set to zero. To account for temporal dependencies, the observation space concatenates the state representations from two consecutive time steps and appends the 3D position of the target goal. This results in a final observation vector of 39 dimensions. The action space is also consistent across the tasks, comprising of the displacement of the end-effector in $\mathbb{R}^3$ and the normalized gripper effort in $\mathbb{R}^1$. An overview of the observation and action can be seen in Figure 2.

**Evaluation Settings:** Following Yu et al. (2021), we provide two evaluation settings: multi-task 10 (MT10) and multi-task 50 (MT50), where MT10 consists of 10 selected tasks and MT50 consists of all 50 tasks. During the evaluation, the overall *success rates* (SR) and the *cumulative reward* (R) are reported across tasks, with each task featuring 10 independent runs with randomly sampled goal parameters. We note that our code supports defining any custom subset of tasks and their associated number of environments in 2 lines of a config file.

## 3.2 Parkour Benchmark

**Task Descriptions:** The twenty Parkour tasks are imported from Eurekaverse Liang et al. (2024). In each task, the agent controls a quadrupedal Unitree Go1 robot (Unitree Robotics, 2021) to track predefined waypoints while traversing one of 20 different terrain categories (Liang et al., 2024).

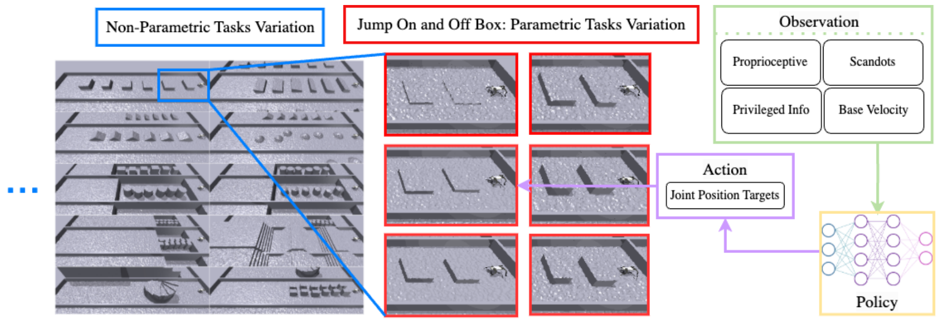

Figure 3: Illustrations of non-parametric tasks variation, parametric tasks variation of Jump On and Off Box, and the observation and action space of the RL agents in the Parkour benchmark.

These tasks challenge various motor skills, including climbing boxes, walking on slopes, jumping, navigating stepping stones, ascending stairs, maneuvering through narrow hallways, weaving through agility poles in a zig-zag pattern, and maintaining balance. Figure 8 on the right shows bird-eye views of these terrain categories.

Each task also provides parametric terrain variations defined by a set of terrain parameters, whose definitions and valid ranges are detailed in the supplementary materials. Additionally, each task introduces a one-dimensional continuous variable, termed *difficulty*, and a predefined mapping from the *difficulty* to a set of terrain parameters. This *difficulty* measure aligns with human intuition; for instance, high boxes present a greater challenge than lower boxes for a quadrupedal robot to jump on and off.

**Observation and Action Spaces:** The observation of the agent is slightly simplified for more efficient benchmarking compared to Eurekaverse. The observation is compromised by proprioceptive observation in $\mathbb{R}^{48}$, scandots of the terrain environments in $\mathbb{R}^{132}$, base linear velocity in $\mathbb{R}^{3}$, and privileged information in $\mathbb{R}^{29}$. The action assigns joint position targets at a frequency of 50 Hz for a Proportional-Derivative (PD) controller. An overview of the observation and actions is shown in Figure 8. The reward function resembles Fu et al. (2023), which encourages positive linear and angular velocities that point to the next waypoint, while minimizing energy consumption.

**Evaluation Settings:** We define two evaluation settings: Parkour-easy and Parkour-hard. Parkour-easy consists of 200 terrains, with each of the 20 tasks assigned 10 terrains generated at the lowest difficulty level. In contrast, Parkour-hard also includes 200 terrains but distributes difficulty levels uniformly across the 10 terrains per task, providing a more diverse and challenging evaluation setting. In addition, Parkour also supports custom evaluation settings like Meta-World. Before the training, all the evaluated methods are pre-trained on flat ground to acquire the basic walking gait. Such a pre-training phase is typical in the literature (Zhuang et al., 2023; Cheng et al., 2024).

During evaluation, we measure *progress* (P) as the ratio of the current waypoint index to the total number of waypoints at the time of episode termination. An agent that successfully traverses the entire terrain achieves a *progress* score of 100%. The overall *progress* is computed as the average over 200 terrains, with each terrain evaluated across 10 independent runs.

## 3.3 Algorithms

We re-implement a suite of algorithms and MTRL schemes using a popular learning library RL-Games (Makoviichuk & Makoviychuk, 2021), providing a unified benchmark for end-to-end vectorized MTRL training across many seeds and hyperparameters on a single GPU. Our benchmark is highly extensible towards new RL algorithms as well as schemes along the two axes of MTRL research, gradient manipulation, and neural architectures. There is a brief overview in Appendix A.

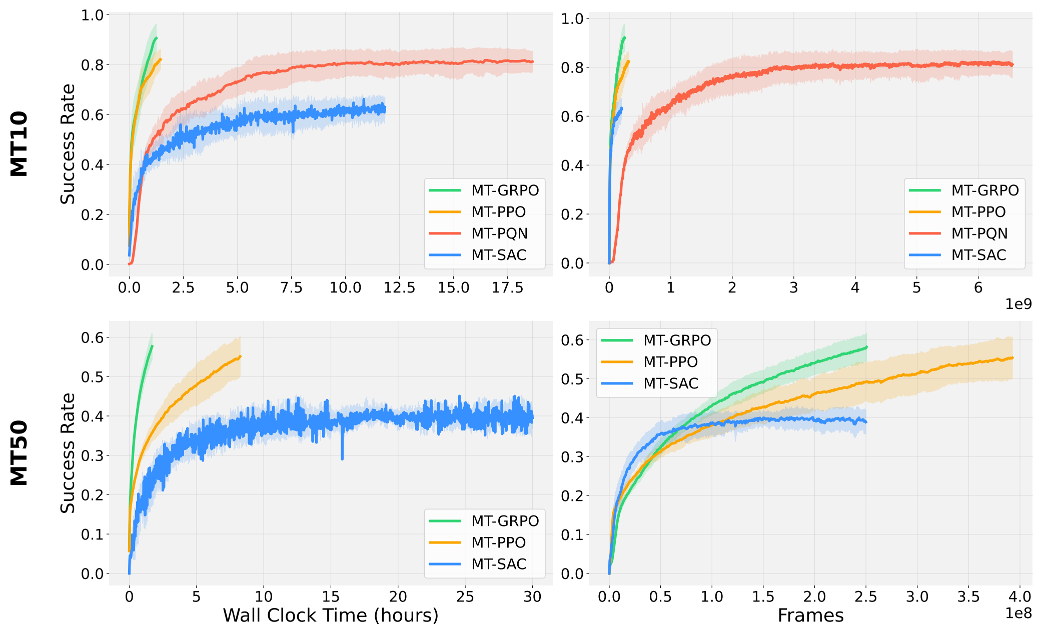

Figure 4: **Vanilla MTRL performance in Meta-World.** We present success rates averaged over 10 seeds for each RL algorithm on MT10 and MT50. We see that on-policy methods (MT-PPO, MT-GRPO) continue to improve with more experience, achieving a substantially higher success rate than off-policy methods (MT-SAC, MT-PQN) in substantially less time.

**Base MTRL Algorithms** We implement four RL algorithms: MT-PPO, a multi-task version of Proximal Policy Optimization (Schulman et al., 2017); MT-GRPO (Guo et al., 2025), a variant of PPO introduced for the language modeling but adapted here for control; MT-SAC, a multi-task version of Soft Actor-Critic (Haarnoja et al., 2018); and MT-PQN, a novel multi-task extension to Parallel Q-learning (Gallici et al., 2024) to handle continuous control problems. We enable multi-task learning by simply augmenting the observation space with one-hot task embeddings.

**MTRL Schemes** We implement two categories of MTRL schemes that can be easily combined with any of our base algorithms. The first category consists of gradient manipulation methods: PCGrad (Yu et al., 2020), CAGrad (Liu et al., 2024), and FAMO (Liu et al., 2023). The second category consists of multi-task architectures: CARE (Sodhani et al., 2021), MOORE (Hendawy et al., 2024), PaCO (Sun et al., 2022), and Soft-Modularization (Yang et al., 2020).

**Curriculum Learning** Unlike Meta-World, where reward functions are carefully designed with dense rewards, locomotion tasks often rely on sparse reward signals (e.g., moving forward to the next waypoints). As a result, strategies like curriculum learning have been widely adopted to facilitate learning in challenging tasks, such as running (Margolis et al., 2024) and jumping onto high platforms (Liang et al., 2024). Inspired by this, we incorporate a simple curriculum strategy to train Parkour-hard tasks. In Parkour-hard, each task consists of ten terrains with varying levels of *difficulty*. Agents always begin on the easiest terrain and progress to more challenging ones if they achieve a *progress* of at least 80% in their current terrain. We refer to the Parkour-hard training with curriculum learning by Parkour-hard-cl.

## 4 Results

In this section, we present the results of our benchmark across our evaluation settings and empirically justify the aforementioned four major observations.

| Methods \ Tasks | MT10 | | MT50 | | Parkour-easy | Parkour-hard | Parkour-hard-CL |
|---|---|---|---|---|---|---|---|
| | SR ↑ | R ↑ | SR ↑ | R ↑ | P(%) ↑ | P(%) ↑ | P(%) ↑ |
| Vanilla | $82.05 \pm 4.41$ | $914.75 \pm 42.72$ | $48.12 \pm 4.65$ | $602.07 \pm 57.90$ | $80.39 \pm 0.43$ | $54.51 \pm 0.82$ | $68.12 \pm 0.43$ |
| Multihead | $82.51 \pm 5.27$ | $953.45 \pm 50.51$ | $63.28 \pm 3.18$ | $780.51 \pm 30.03$ | $73.17 \pm 2.53$ | $49.65 \pm 1.27$ | $62.17 \pm 1.05$ |
| PCGrad | $\mathbf{93.19} \pm \mathbf{1.79}$ | $\mathbf{1050.75} \pm \mathbf{16.72}$ | $53.28 \pm 3.94$ | $669.94 \pm 50.29$ | $\mathbf{80.61} \pm \mathbf{0.78}$ | $\mathbf{55.98} \pm \mathbf{0.24}$ | $68.37 \pm 0.65$ |
| CAGrad | $90.72 \pm 3.75$ | $1023.24 \pm 29.76$ | $53.38 \pm 3.90$ | $657.85 \pm 37.52$ | $80.25 \pm 0.32$ | $55.88 \pm 0.91$ | $67.75 \pm 0.43$ |
| FAMO | $90.33 \pm 4.17$ | $1010.49 \pm 24.00$ | $\mathbf{65.07} \pm \mathbf{2.91}$ | $\mathbf{814.40} \pm \mathbf{24.73}$ | $79.79 \pm 0.29$ | $55.56 \pm 0.97$ | $\mathbf{68.57} \pm \mathbf{0.76}$ |
| PaCO | $\mathbf{84.42} \pm \mathbf{1.96}$ | $\mathbf{958.25} \pm \mathbf{29.89}$ | $54.07 \pm 5.17$ | $687.23 \pm 59.95$ | $78.63 \pm 0.70$ | $\mathbf{58.65} \pm \mathbf{0.77}$ | $64.15 \pm 0.98$ |
| MOORE-S | $80.19 \pm 4.93$ | $915.69 \pm 19.47$ | $43.73 \pm 2.15$ | $523.04 \pm 21.52$ | $64.61 \pm 1.69$ | $46.78 \pm 0.65$ | $49.53 \pm 0.52$ |
| MOORE-M | $87.09 \pm 3.83$ | $973.52 \pm 15.32$ | $62.61 \pm 3.45$ | $745.64 \pm 41.93$ | - | - | - |
| CARE-S | $78.05 \pm 5.04$ | $887.72 \pm 41.32$ | $45.63 \pm 5.01$ | $572.14 \pm 54.33$ | - | - | - |
| CARE-M | $79.47 \pm 6.03$ | $898.08 \pm 38.58$ | $55.68 \pm 2.97$ | $647.27 \pm 34.16$ | - | - | - |
| Soft-Modularization | $79.94 \pm 4.51$ | $908.66 \pm 46.53$ | $56.20 \pm 3.40$ | $693.57 \pm 41.76$ | $69.29 \pm 3.81$ | $47.28 \pm 0.23$ | $51.43 \pm 0.35$ |

Table 1: We present our evaluation metrics of all MTRL schemes using MT-PPO on all evaluation settings. All results are averaged over 10 seeds and use 250M frames. Best-performing approaches in each category are presented in **bold** text if they surpass both the Vanilla and Multihead (M) baselines. The training curves for these experiments are shown in Appendix D.

### 4.1 Choosing the MTRL Base Algorithm (O1, O2)

To illustrate how on-policy MTRL methods leverage massive parallelism, we first evaluate one on-policy method, MT-PPO, alongside two off-policy methods, MT-SAC and MTPQN, in Meta-World. Figure 4 presents the learning curves with respect to both wall-clock time and the number of environment interactions. Since this observation is concerned with answering what the best base MTRL algorithm is, we tune all aspects of each method to achieve its highest success rate, including using different network architectures. We present the full hyperparameter and model details in the supplementary materials.

**On-policy methods outperform traditional off-policy methods.** Using MT-SAC as representative of traditional off-policy algorithms used for MTRL, Figure 4 shows there is a substantial performance gap in success rate (roughly 87% and 65% for MT-PPO and MT-SAC respectively) and more relevant to researchers, a substantial wall-clock time difference (roughly 1.5 hours and 13 hours) after 250M frames of collected experience. Of course, it is possible to speed up MT-SAC to match MT-PPO's runtime by decreasing the gradient steps per epoch to that of MTPPO, but this results in a near-zero success rate. Furthermore, as the number of tasks increases to the MT-50 setting, these gaps increase.

Traditional off-policy methods in the end-to-end single GPU setting cannot effectively leverage increased environment interaction in the massively parallelized regime, where their stability, performance, and runtime greatly rely on the ratio of gradient updates to environment steps i.e update-to-data (UTD) ratio (D'Oro et al., 2022) being greater than or equal to 1. Prior research like (Li et al., 2023) on leveraging massive parallelism for off-policy methods exists but is not adapted for the multi-task setting yet and requires multiple GPUs.

**Off-Policy methods can be designed for the massively parallelized regime.** In Figure 4, we also included our adaptation of PQN (Gallici et al., 2024) to the multi-task continuous control setting. The details of our implementation are in Appendix B. Surprisingly, applying these simple changes to an originally discrete action algorithm and left to run long enough, MT-PQN can almost match the performance of MT-PPO (roughly an x% drop) in MT10. Considering PQN's performance and stability, similar simulation throughput to PPO, and lack of a replay buffer suggest that smartly adapting PQN to continuous control could be a promising research direction.

### 4.2 MTRL Schemes (O2, O3)

Table 1 reports the major evaluation results of all MTRL schemes using MTPPO. Unless otherwise specified, all the gradient manipulation methods use the same three-layer MLP neural networks.

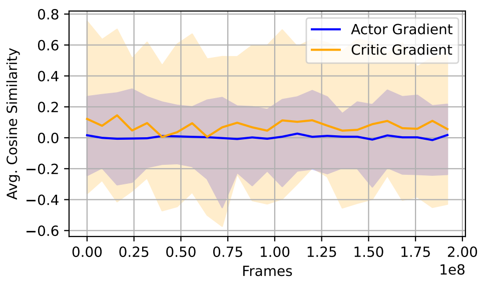

Figure 5: The average gradient cosine similarity across all task pairs in MT10 for both actor and critic networks. The shadow areas represent the ranges between minimum and maximum cosine similarities.

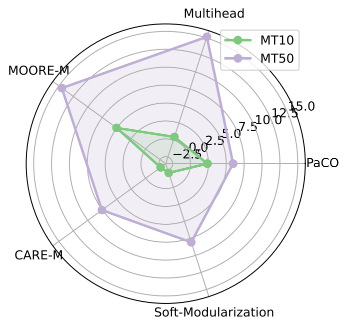

Figure 6: Success rate (SR) differences of five neural network architectures relative to the Vanilla baseline in MT10 and MT50.

**Multi-task architectures show greater performance gains with larger task sets.** As shown in Figure 6, the benefits of multi-task architectures become more pronounced as the number of tasks increases. In MT10, vanilla PPO performs similarly to advanced multi-task architectures. However, in MT50, the best-performing multi-task architecture, MOORE-M, surpasses MLP by 17% in success rate. This improvement is likely due to enhanced knowledge sharing that only manifests in training diverse enough tasks, such as MT50. However, similar performance gains are not observed in the Parkour benchmark, likely due to the insufficient task diversity in Parkour tasks.

**Resolving gradient conflict consistently improves the performance.** The three gradient manipulation methods (rows 3-5 in Table 1) consistently outperform vanilla PPO across all evaluation settings. This suggests that gradient conflicts are still a common optimization challenge in multi-task RL problems. Among these methods, FAMO shows superior scalability in larger task sets, likely due to its simple strategy of adaptive task weighting, which eliminates the need for backpropagating through each task's loss and scales better with the increasing number of tasks.

**Value learning is the key bottleneck in MTRL.** Prior research in MTRL has shown that addressing gradient conflicts improves performance in value-based RL algorithms like SAC. Our benchmarking results extend this observation to the actor-critic framework, demonstrating that gradient conflicts also arise when learning the critic network in PPO. However, we do not observe similar conflicts in policy optimization. This observation aligns with prior work using on-policy actor-critic algorithm for large-scale multi-task learning (Hessel et al., 2019). Figure 5 shows the average cosine similarity across all task gradient pairs for both actor and critic networks, where critic gradients manifest lower minimum similarities.

### 4.3 Reward Sparsity (O4)

Although tasks in Meta-World and the Parkour Benchmark are defined independently of their reward functions, training performance is significantly influenced by reward design. We adopt commonly used reward formulations in both domains. In Meta-World, tasks utilize dense rewards, which provide continuous feedback to guide specific interactions between the robotic arm and objects. In contrast, the Parkour Benchmark employs a sparse reward scheme, where the agent is rewarded solely for maintaining forward velocity toward waypoints, without receiving additional signals for intermediate behaviors.

**Dense rewards increase the complexity of multi-task critic learning.** In multi-task RL, dense reward functions introduce challenges for critic learning, as different tasks exhibit varying reward distributions and gradient magnitudes. Addressing these conflicts leads to around 30% performance

| Method \ Tasks | MT10 | | MT50 | |
|---|---|---|---|---|
| | SR ↑ | R ↑ | SR ↑ | R ↑ |
| Vanilla MT-PPO | $82.05 \pm 4.41$ | $914.75 \pm 42.72$ | $48.12 \pm 4.65$ | $602.07 \pm 57.90$ |
| Vanilla GRPO | $92.09 \pm 5.43$ | $904.72 \pm 33.59$ | $57.81 \pm 3.54$ | $637.07 \pm 34.34$ |

Table 2: **Effect of removing the critic.** We compare how removing the main source of gradient confliction affects success rate in Meta-World, which uses dense reward.

increase in dense-reward multi-task settings such as MT10 and MT50. However, the performance gains are relatively marginal in a sparse-reward multi-task setting like the Parkour benchmark.

**Curriculum learning is crucial for sparse-reward tasks.** In environments with sparse rewards, standard MTRL methods do not inherently enhance exploration, as agents receive limited feedback in each task. This challenge is particularly evident in the Parkour Benchmark, where agents tend to adopt overly conservative behaviors in more difficult tasks. Curriculum learning addresses this issue by structuring task progression, enabling agents to first master simpler behaviors before tackling more complex ones. By gradually increasing task difficulty, curriculum learning improves exploration efficiency and yields a 10% performance gain in *progress*, as observed when comparing Parkour-hard and Parkour-hard-cl (columns 7 and 8 in Table 1).

### 4.4 Learning without a Critic (O3)

To further investigate the impact of gradient confliction in the critic on MTRL, we can eliminate the critic by increasing the horizon length in MT-PPO to be equal to the length of the episode.

**MTRL can benefit from eliminating gradient confliction in the critic.** In fact, this setting is equivalent to implementing GRPO (Guo et al., 2025) without the KL term. We use the Monte Carlo estimate of the episodic return as the reward in the advantage calculation. In the dense reward setting, Table 2 indicates that MT-GRPO is a simple baseline that outperforms most MTRL schemes and deserves attention in future work.

**Massive Parallelism is well suited for reducing bias from an imperfect critic.** By directly using Monte Carlo returns instead of bootstrapping, we effectively eliminate the bias introduced by imperfect critic estimation. This approach represents a clear bias-variance tradeoff: while removing the critic increases the variance of our gradient estimates, this increased variance can be effectively mitigated through large batches (of size episode length times the number of parallel environments) made possible by massive parallelization (Sutton et al., 1999).

## 5 Related

### 5.1 Parallelizing RL

As Deep RL relies on training neural networks (learners) and collecting experience (actors), many methods have explored how to parallelize both of these aspects to speed up training over the years. Early works leveraged low levels of parallelization without hardware accelerators mainly for Atari either in a distributed compute cluster of hundreds (in some cases thousands) of CPU cores (Nair et al., 2015) or a single machine using a multi-threaded approach (Mnih et al., 2016). Hybrid CPU-GPU distributed frameworks introduce accelerating learners with GPUs (Babaeizadeh et al., 2016; Espeholt et al., 2018; Horgan et al., 2018) along with actors collecting experience across CPUs.

Unlike Atari, robotic control tasks rely on physics simulators (Todorov et al., 2012), where distributed RL methods using CPU-based simulators would demand even more intense hardware requirements (Liang et al., 2018a; Andrychowicz et al., 2020) due to running multiple simulator instances in parallel. A wave of recent GPU-accelerated simulators (Liang et al., 2018b; Freeman et al., 2021; Makoviychuk et al., 2021; Mittal et al., 2023; Tao et al., 2024; Zakka et al., 2025) has essentially alleviated the experience collection constraint and shown success in rapidly learning single-task robotic control tasks (Allshire et al., 2021; Rudin et al., 2022) with the modest hardware requirement of 1 GPU.

### 5.2 GPU-Accelerated Benchmarks

Several RL benchmarks have arisen as a result of GPU-based simulation mainly in JAX-based game environments (Cobbe et al., 2020; Lange, 2022; Morad et al., 2023; Bonnet et al., 2023; Rutherford et al., 2024; Matthews et al., 2024). In addition, classic continuous control tasks and simple robotic tasks are often bundled with GPU-accelerated simulators. However, our GPU-accelerated benchmark is explicitly designed for multi-task reinforcement learning in *robotics* on 1 GPU. The most similar effort to our work would be RLBench (James et al., 2020), Meta-World (Yu et al., 2021), or MTRL (Sodhani & Zhang, 2021), but all suffer from the hardware and runtime constraints induced by experience collection on CPU-based simulators (Todorov et al., 2012; James et al., 2019).

## 6 Conclusions

We present MTBench, an MTRL benchmark including a GPU-accelerated implementation of Meta-World and locomotion tasks, extensive gradient manipulation/neural architecture baselines, and an initial study on the current state as well as future directions of MTRL in the massively parallel regime. However, a key limitation is our limitation to state-based MTRL to retain high simulation throughput, which we hope to resolve with pixel-based MTRL using NVIDIA IsaacLab in a future release of MTBench.

Future work can use MTBench beyond learning tabula rasa MTRL methods. One can explore offline RL, imitation learning, or distillation methods by writing additional code to rapidly collect transitions from expert single-task agents. Another application of our benchmark could be as part of the ‘finetune’ step in the ‘pretrain, then finetune’ paradigm where one pre-trains on a diverse set of tasks using offline RL and rapidly finetune agents online using our environments.

## A MTRL Schemes

Here, we present an overview of each state-of-the-art MTRL baseline in MTBench.

### A.1 Gradient manipulation methods

Gradient manipulation methods compute a new gradient of the multi-task objective, incurring the overhead of solving an optimization problem per iteration as well as storing and computing $K$ task gradients.

**PCGrad:** Projecting Conflicting Gradients (Yu et al., 2020) observe when the gradients of any two task objectives $l_i$ conflict ( defined as having negative cosine similarity) and when their magnitudes are sufficiently different, optimization using the average gradient will cause negative transfer. It attempts to resolve gradient confliction by a simple procedure manipulating each task gradient $\nabla l_i$ to be the result of iteratively removing the conflict with each task gradient $\nabla l_j, \forall j \in [K], j \neq i$.

$$\nabla l_i' \leftarrow \nabla l_i - \frac{\nabla l_i^T \nabla l_j}{\|\nabla l_j\|^2} \nabla l_j \quad \text{if} \quad \nabla l_i^T \nabla l_j < 0 \tag{1}$$

**CAGrad:** Conflict-Averse Gradient descent (Liu et al., 2024) resolves the gradient conflict by finding an update vector $d \in \mathbb{R}^m$ that minimizes the worst-case gradient conflict across all the tasks.

More specifically, let $g_i$ be the gradient of task $i \in [K]$, and $g_0$ be the gradient computed from the average loss, CAGrad seeks to solve such an optimization problem:

$$\max_{d \in \mathbb{R}^m} \min_{i \in [K]} \langle g_i, d \rangle \quad \text{s.t.} \quad \|d - g_0\| \leq c\|g_0\| \tag{2}$$

Here, $c \in [0, 1)$ is a pre-specified hyper-parameter that controls the convergence rate. The optimization problem looks for the best update vector within a local ball centered at the averaged gradient $g_0$, which also minimizes the conflict in losses $\langle g_i, d \rangle$.

**FAMO:** Fast Adaptive Multitask Optimization (Liu et al., 2023) addresses the under-optimization of certain tasks when using standard gradient descent on averaged losses without incurring the $O(K)$ cost to compute and store all task gradients, which can be significant, especially as the number of tasks increases. FAMO leverages loss history to adaptively adjust task weights, ensuring balanced optimization across tasks while maintaining $O(1)$ space and time complexity per iteration.

### A.2 Neural Architectures

Neural Architecture methods seek to avoid task interference by learning shared representations, which are fed to the prediction head. Such representations accelerate MTRL.

**CARE:** Contextual Attention-based Representation learning (Sodhani & Zhang, 2021) utilizes metadata associated with the set of tasks to weight the representations learned by a mixture of encoders through the attention mechanism.

**MOORE:** Mixture Of Orthogonal Experts (Hendawy et al., 2024) uses a mixture of experts to encode the state and orthogonalizes those representations to encourage diversity, weighting these representations from a task encoder.

**PaCO:** Parameter Compositional (Sun et al., 2022) learns a base parameter set $\phi = [\phi_1 \cdots \phi_k]$ and task-specific compositional vector $w_k$ such that multiplying $\phi$ and $w_k$ represents the task parameters $\theta_k$.

**Soft-Modularization:** Yang et al. (2020) also uses a mixture of experts to encode the state but also uses a routing network to softly combine the outputs at each layer based on the task.

## B PQN

Parallel Q-learning (Gallici et al., 2024) is a recent off-policy TD method designed for discrete action spaces and massively parallelized GPU-based simulators that casts aside the tricks introduced over the years to stabilize deep Q learning such as replay buffers (Mnih et al., 2013), target networks (Mnih et al., 2015) and double Q-networks (Wang et al., 2016) by simply introducing regularization in the function approximator like LayerNorm (Ba et al., 2016) or BatchNorm (Ioffe & Szegedy, 2015). Coupled with this architectural change, PQN exploits vectorized environments by collecting experience in parallel for $T$ steps.

As our action space is continuous, we follow Seyde et al. (2023) to present a modified version of PQN through bang-off-bang control and treating continuous control as a $M$ agent multi-agent problem where each actuator is an agent in a cooperative game. Then, the state-action function $Q_\theta(\mathbf{s_t}, \mathbf{a_t})$ is factorized as the average of M different state-action functions $Q^i_\theta(\mathbf{s_t}, a^i_t)$, where the $i$th state-action function predicts the value of the bang-off-bang actions in $i$th action dimension following Sunehag et al. (2017).

$$Q_\theta(\mathbf{s_t}, \mathbf{a_t})) = \frac{1}{M} \sum_{i=1}^{M} Q^i_\theta(\mathbf{s_t}, a^i_t) \tag{3}$$

In code, the output of the state-action function is of size $(B, M, n_b)$ where $B$ is the batch size, $m$ is the action dimension/number of actuators (4) and $n_b$ is the number of bins per dimension (3). The action value is recovered by first taking the max over the bin dimension and then the mean over the action dimension. By taking the max over the bin dimension, Seyde et al. (2023) sidestepped taking a max over the continuous action space. Now, we can compute the Bellman target and in the case of PQN, n-step returns.

$$y_t = r(\mathbf{s_t}, \mathbf{a_t}) + \gamma \frac{1}{M} \sum_{i=1}^{M} \max_{a^i_{t+1}} Q^i_\theta(\mathbf{s_{t+1}}, a^i_{t+1}) \tag{4}$$

## C Meta-World

### C.1 Success

We report two evaluation metrics, the overall success rate averaged across tasks and the cumulative reward achieved by the multi-task policy. Following the original Meta-World, success is a boolean indicating whether the robot brings the object within an $\epsilon$ distance of the goal position at *any* point during the episode, which is less restrictive than works qualifying a success only if it occurs at the *end* of an episode. Mathematically, success occurs if $\|o - g\|_2 < \epsilon$ is satisfied at least once, where $o$ is the object position and $g$ is the goal position.

Rather than defining the success rate as the maximum success rate over some evaluation rollouts as some previous work did, the success rate is defined as the proportion of success in the large number of environments that terminate every step. The reported success rate is this success rate averaged over the last 5 epochs of training. Due to massive parallelization, there is no need to separately roll out the learned policy in a separate process.

## D Additional Results

### D.1 Gradient Manipulation

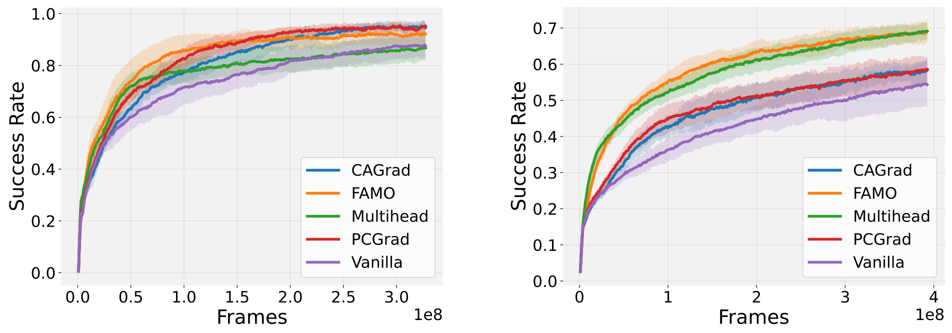

Figure 7: The training curves for the three gradient manipulation methods and two baselines in both MT10 (left) and MT50 (right).

## References

Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. *arXiv preprint arXiv:2108.09779*, 2021.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I Midgley, Elshadai Tegegn, Tristan Kalloniatis, et al. Jumanji: a diverse suite of scalable reinforcement learning environments in jax. *arXiv preprint arXiv:2306.09884*, 2023.

Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11443–11450. IEEE, 2024.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Franka Robotics. Franka emika panda robot, 2017. URL https://www.franka.de. Accessed: 2025-02-17.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning*, pp. 138–149. PMLR, 2023.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning, 2024. URL https://arxiv.org/abs/2407.04811.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290.

Ahmed Hendawy, Jan Peters, and Carlo D'Eramo. Multi-task reinforcement learning with mixture of orthogonal experts, 2024. URL https://arxiv.org/abs/2311.11385.

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, pp. 3796–3803, 2019.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay, 2018. URL https://arxiv.org/abs/1803.00933.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

Stephen James, Marc Freese, and Andrew J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

Zechu Li, Tao Chen, Zhang-Wei Hong, Anurag Ajay, and Pulkit Agrawal. Parallel $q$-learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *International Conference on Machine Learning*. PMLR, 2023.

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*, pp. 3053–3062. PMLR, 2018a.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pp. 270–282. PMLR, 2018b.

William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models, 2024. URL https://arxiv.org/abs/2411.01775.

Bo Liu, Yihao Feng, Peter Stone, and Qiang Liu. Famo: Fast adaptive multitask optimization, 2023. URL https://arxiv.org/abs/2306.03792.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning, 2024. URL https://arxiv.org/abs/2110.14048.

Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning. https://github.com/Denys88/rl_games, May 2021.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021. URL https://arxiv.org/abs/2108.10470.

Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *The International Journal of Robotics Research*, 43(4):572–587, 2024.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning, 2024. URL https://arxiv.org/abs/2402.16801.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. DOI: 10.1109/LRA.2023.3270034.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. URL https://api.semanticscholar.org/CorpusID:205242740.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PmLR, 2016.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. Popgym: Benchmarking partially observable reinforcement learning. *arXiv preprint arXiv:2303.01859*, 2023.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning, 2016. URL https://arxiv.org/abs/1609.09025.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pp. 91–100. PMLR, 2022.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktaschel, Chris Lu, and Jakob Nicolaus Foerster. Jaxmarl: Multi-agent rl environments and algorithms in jax, 2024. URL https://arxiv.org/abs/2311.10090.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Tim Seyde, Peter Werner, Wilko Schwarting, Igor Gilitschenski, Martin Riedmiller, Daniela Rus, and Markus Wulfmeier. Solving continuous control via q-learning, 2023. URL https://arxiv.org/abs/2210.12566.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Jayesh Singla, Ananye Agarwal, and Deepak Pathak. Sapg: Split and aggregate policy gradients. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, Proceedings of Machine Learning Research, Vienna, Austria, July 2024. PMLR.

Shagun Sodhani and Amy Zhang. Mtrl - multi task rl algorithms. Github, 2021. URL https://github.com/facebookresearch/mtrl.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, 2021. URL https://api.semanticscholar.org/CorpusID:231879645.

Lingfeng Sun, Haichao Zhang, Wei Xu, and Masayoshi Tomizuka. Paco: Parameter-compositional multi-task reinforcement learning. *ArXiv*, abs/2210.11653, 2022. URL https://api.semanticscholar.org/CorpusID:253080666.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017. URL https://arxiv.org/abs/1706.05296.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. DOI: 10.1109/IROS.2012.6386109.

Unitree Robotics. *Go1 User Manual*. Unitree Robotics, 2021. Available at https://www.unitree.com/go1.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2016. URL https://arxiv.org/abs/1511.06581.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization, 2020. URL https://arxiv.org/abs/2003.13661.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning, 2020. URL https://arxiv.org/abs/2001.06782.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. URL https://arxiv.org/abs/1910.10897.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carlo Sferrazza, Yuval Tassa, and Pieter Abbeel. Mujoco playground: An open-source framework for gpu-accelerated robot learning and sim-to-real transfer., 2025. URL https://github.com/google-deepmind/mujoco_playground.

Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher Atkeson, Soeren Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023.

# Supplementary Materials

*The following content was not necessarily subject to peer review.*

## E Hyperparameter Details

In this section, we provide hyperparameter values for each MTRL scheme.

### E.1 Meta-World MT-PPO

### E.2 Meta-World MT-GRPO

### E.3 Meta-World MT-PQN

### E.4 Meta-World MT-SAC

### E.5 Parkour MT-PPO

## F Comparison to the original Meta-World

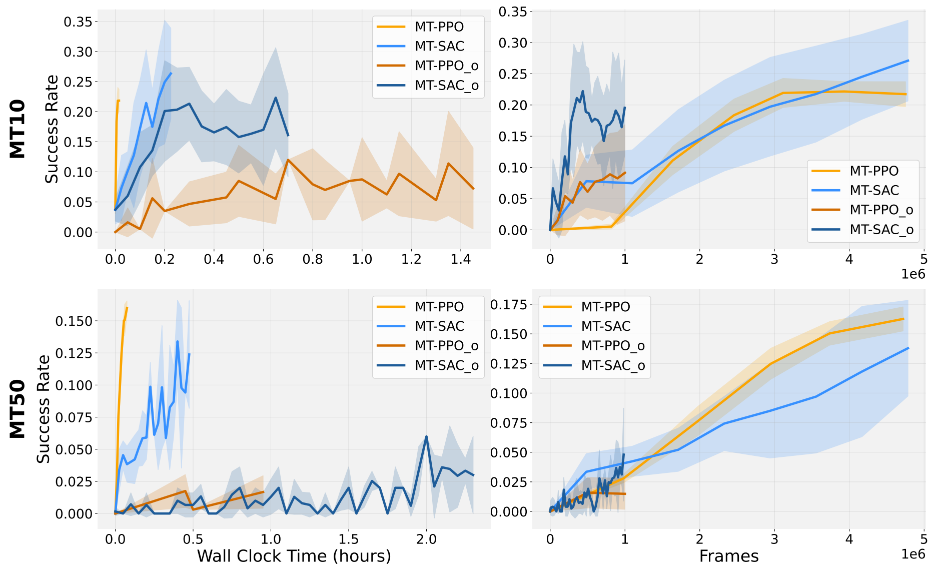

Figure 8: Comparison of our Meta-World to the original (_o) Meta-World using the hyperparameters listed from (Yu et al., 2021)

| Description | value | variable_name |
|---|---|---|
| Number of environments | 5120 / 6400 | num_envs |
| Minibatch size | 16384 / 25600 | minibatch_size |
| Horizon length | 32 | horizon |
| Mini-epochs | 5 | mini_epochs |
| Number of epochs | 1526 / 1221 | max_epochs |
| Episode length | 150 | episodeLength |
| Discount factor | 0.99 | gamma |
| Clip ratio | 0.2 | e_clip |
| Policy entropy coefficient | .005 | entropy_coef |
| Optimizer learning rate | 5e-4 | learning_rate |
| Optimizer learning schedule | fixed | lr_schedule |
| Advantage estimation tau | 0.95 | tau |
| Value Normalization by task | True | normalize_value |
| Input Normalization by task | True | normalize_input |
| Separate critic and policy networks | True | network.separate |
| CARE-Specific Hyperparameters | | |
| Network hidden sizes | [400,400,400] | care.units |
| Mixture of Encoders experts | 6 | encoder.num_experts |
| Mixture of Encoders layers | 2 | encoder.num_layers |
| Mixture of Encoders hidden dim | 50 | encoder.D |
| Attention temperature | 1.0 | encoder.temperature |
| Post-Attention MLP hidden sizes | [50,50] | attention.units |
| Context encoder hidden sizes | [50,50] | context_encoder.units |
| Context encoder bias | True | context_encoder.bias |
| MOORE-Specific Hyperparameters | | |
| MoE experts | 4 / 6 | moore.num_experts |
| MoE layers | 3 | moore.num_layers |
| MoE hidden dim | 400 | moore.D |
| Activation before/after task encoding weighting | [Linear, Tanh] | moore.agg_activation |
| Task encoder hidden sizes | [256] | task_encoder.units |
| Task encoder bias | False | task_encoder.bias |
| PaCO-Specific Hyperparameters | | |
| Number of Compositional Vectors | 5 / 20 | paco.K |
| Network hidden dim | 400 | paco.D |
| Network layers | 3 | paco.num_layers |
| Task encoder bias | False | task_encoder.bias |
| Task encoder init | orthogonal | task_encoder.compositional_initializer |
| Task encoder activation | softmax | task_encoder.activation |
| Soft-Modularization-Specific Hyperparameters | | |
| MoE experts | 2 | soft_network.num_experts |
| MoE layers | 4 | soft_network.num_layer |
| State encoder hidden sizes | [256,256] | state_encoder.units |
| Task encoder hidden sizes | [256] | task_encoder.units |
| PCGrad Hyperparameters | | |
| Project actor gradient | False | project_actor_gradient |
| Project critic gradient | True | project_critic_gradient |
| CAGrad Hyperparameters | | |
| Project actor gradient | False | project_actor_gradient |
| Project critic gradient | True | project_critic_gradient |
| Local ball radius for searching update vector | 0.4 | c |
| FAMO Hyperparameters | | |
| Regularization coefficient | 1e-3 | gamma |
| Learning rate of the task logits | 1e-3 | w_lr |
| Small value for the clipping of the task logits | 1e-2 | epsilon |
| Normalize the task logits gradients | True | norm_w_grad |

Table 3: Hyperparameters used for MTPPO. A '/' indicates the value used for MT10/MT50 respectively and otherwise is identical for each setting.

| Description | value | variable_name |
|---|---|---|
| Number of environments | 4096 / 6400 | num_envs |
| Minibatch size | 16384 / 25600 | minibatch_size |
| Episode length | 150 | episodeLength |
| Horizon length | 32 | horizon |
| Mini-epochs | 5 | mini_epochs |
| Number of epochs | 1908 / 1221 | max_epochs |
| Episode length | 150 | |
| Discount factor | 0.99 | gamma |
| Clip ratio | 0.2 | e_clip |
| Policy entropy coefficient | .005 | entropy_coef |
| Optimizer learning rate | 5e-4 | learning_rate |
| Optimizer learning schedule | fixed | lr_schedule |
| Advantage estimation tau | 0.95 | tau |
| Value Normalization by task | True | normalize_value |
| Input Normalization by task | True | normalize_input |
| Separate critic and policy networks | True | network.separate |

Table 4: Hyperparameters used for MT-GRPO in MT10 / MT50. A '/' indicates the value used for MT10/MT50 respectively and otherwise is identical for each setting.

| Description | value | variable_name |
|---|---|---|
| Number of environments | 8192 | num_envs |
| Gamma | .99 | gamma |
| Peng's Q(lambda) | .5 | q_lambda |
| Number of minibatches | 4 | num_minibatches |
| Episode length | 500 | episodeLength |
| Bang-off-Bang | 3 | binsPerDim |
| Action Scale | .005 | actionScale |
| Mini epochs | 8 | mini_epochs Max grad norm |
| 10.0 | max_grad_norm | |
| Horizon | 16 | horizon |
| Start epsilon | 1.0 | start |
| End epsilon | 0.005 | end |
| Decay epsilon | True | decay_epsilon |
| Fraction of exploration steps | .005 | exploration_fraction |
| Critic learning rate | 3e-4 | critic_lr |
| Anneal learning rate | True | anneal_lr |
| Value Normalization by task | False | normalize_value |
| Input Normalization by task | False | normalize_input |
| Use residual connections | True | q.residual_network |
| Number of LayerNormAndResidualMLPs | 2 | q.num_blocks |
| Network hidden dim | 256 | q.D |
| Batch norm input | False | q.norm_first_layer |

Table 5: Hyperparameters used for MT-PQN in MT10.

| Description | value | variable_name |
|---|---|---|
| Number of environments | 4096 | num_envs |
| Gamma | .99 | gamma |
| Separate critic and policy networks | True | network.separate |
| Number of Gradient steps per epoch | 32 | gradient_steps_per_itr |
| Learnable temperature | True | learnable_ temperature |
| Use distangeled alpha | True | use_disentangled_alpha |
| Initial alpha | 1 | init_alpha |
| Alpha learning rate | 5e-3 | alpha_lr |
| Critic learning rate | 5e-4 | critic_lr |
| Critic tau | .01 | critic_tau |
| Batch size | 8192 | batch_size |
| N-step reward | 16 | nstep |
| Grad norm | .5 | grad_norm |
| Horizon | 1 | horizon |
| Value Normalization by task | True | normalize_value |
| Input Normalization by task | True | normalize_input |
| Replay Buffer Size | 5000000 | replay_buffer_size |
| Target entropy coef | 1.0 | target_entropy_coef |

Table 6: Hyperparameters used for MT-SAC in MT10/MT50. A '/' indicates the value used for MT10/MT50 respectively and otherwise is identical for each setting. MT-SAC is very sensitive to the number of environments and replay ratio in the massively parallel regime.

| Description | value | variable_name |
|---|---|---|
| Minibatch size | 16384 | minibatch_size |
| Horizon length | 32 | horizon |
| Mini-epochs | 5 | mini_epochs |
| Number of epochs | 2000 / 4000 | max_epochs |
| Episode length | 800 | |
| Discount factor | 0.99 | gamma |
| Clip ratio | 0.2 | e_clip |
| Policy entropy coefficient | .005 | entropy_coef |
| Optimizer learning rate | 5e-4 | learning_rate |
| Optimizer learning schedule | adaptive | lr_schedule |
| Advantage estimation tau | 0.95 | tau |
| Value Normalization by task | False | normalize_value |
| Input Normalization by task | False | normalize_input |
| Separate critic and policy networks | True | network.separate |
| MOORE-Specific Hyperparameters | | |
| MoE experts | 2 | moore.num_experts |
| MoE layers | 2 | moore.num_layers |
| MoE hidden dim | 256 | moore.D |
| Activation before/after task encoding weighting | [Linear, Linear] | moore.agg_activation |
| Task encoder hidden sizes | [128] | |
| Task encoder bias | False | task_encoder.bias |
| Multihead | False | multi$_head$ |
| PaCO-Specific Hyperparameters | | |
| Number of Compositional Vectors | 5 | paco.K |
| Network hidden dim | 400 | paco.D |
| Network layers | 3 | paco.num_layers |
| Task encoder bias | False | task_encoder.bias |
| Task encoder init | orthogonal | task_encoder.compositional_initializer |
| Task encoder activation | softmax | task_encoder.activation |
| Soft-Modularization-Specific Hyperparameters | | |
| MoE experts | 2 | soft_network.num_experts |
| MoE layers | 2 | soft_network.num_layer |
| State encoder hidden sizes | [256,256] | state_encoder.units |
| Task encoder hidden sizes | [128] | task_encoder.units |
| PCGrad Hyperparameters | | |
| Project actor gradient | False | project_actor_gradient |
| Project critic gradient | True | project_critic_gradient |
| CAGrad Hyperparameters | | |
| Project actor gradient | False | project_actor_gradient |
| Project critic gradient | True | project_critic_gradient |
| Local ball radius for searching update vector | 0.4 | c |
| FAMO Hyperparameters | | |
| Regularization coefficient | 1e-4 | gamma |
| Learning rate of the task logits | 5e-3 | w_lr |
| Small value for the clipping of the task logits | 1e-3 | epsilon |
| Normalize the task logits gradients | True | norm_w_grad |

Table 7: Hyperparameters used for MTPPO in Parkour Benchmark. A '/' indicates the value used for Parkour-easy/Parkour-hard respectively and otherwise is identical for each setting.