Hard Tasks First: Multi-Task Reinforcement Learning Through Task Scheduling

Myungsik Cho [1] Jongeui Park [1] Suyoung Lee [1] Youngchul Sung [1]

Abstract

Multi-task reinforcement learning (RL) faces the significant challenge of varying task difficulties, often leading to negative transfer when simpler tasks overshadow the learning of more complex ones. To overcome this challenge, we propose a novel algorithm, Scheduled Multi-Task Training (SMT), that strategically prioritizes more challenging tasks, thereby enhancing overall learning efficiency. SMT introduces a dynamic task prioritization strategy, underpinned by an effective metric for assessing task difficulty. This metric ensures an efficient and targeted allocation of training resources, significantly improving learning outcomes. Additionally, SMT incorporates a reset mechanism that periodically reinitializes key network parameters to mitigate the simplicity bias, further enhancing the adaptability and robustness of the learning process across diverse tasks. The efficacy of SMT's scheduling method is validated by significantly improving performance on challenging Meta-World benchmarks.

1. Introduction

Recent advancements in deep learning, particularly in computer vision (Krizhevsky et al., 2012), have spurred the integration of deep learning with reinforcement learning (RL), forming what is known as deep RL. This approach has shown promise in complex control tasks, ranging from mastering Atari games using raw pixel data (Mnih et al., 2015) to conquering the game of Go (Silver et al., 2016), and advancing locomotion control (Schulman et al., 2015; Lillicrap et al., 2016a; Schulman et al., 2017; Haarnoja et al., 2018b; Fujimoto et al., 2018). A key aspect of this success is the use of deep neural networks (DNNs) as function approximators, enabling the representation of intricate policies and state-action value functions over vast state-action spaces.

However, a limitation of deep RL lies in its typical focus on individual tasks, leading to sub-optimal sample usage and inefficiency, particularly for complex tasks. Multi-task RL (Wilson et al., 2007; Pinto & Gupta, 2017; Zeng et al., 2018; Hausman et al., 2018; Yang et al., 2020), inspired by the principles of multi-task learning (Caruana, 1997), addresses this by training a single policy network capable of handling multiple tasks, improving sample efficiency through shared parameters. Despite its advantages, multi-task RL encounters the challenge of negative transfer, where learning certain tasks can impede the learning of others, destabilizing training (Sun et al., 2020).

To overcome these challenges, this paper introduces a novel approach: a scheduled multi-task RL method that is resilient to varying task difficulties and enhances performance in more challenging tasks. Our method involves developing a general agent incorporating a task representation vector into its action value function and policy network. We assess each task's difficulty relative to the current policy and adaptively schedule the learning sequence based on this assessment. This approach prioritizes difficult tasks early in training, reducing the undue influence of simpler tasks and enhancing overall performance. We validate our method using the Meta-World benchmark (Yu et al., 2019), which comprises 50 robotic arm manipulation tasks. Our results demonstrate significant improvements over the baseline algorithm.

The primary contributions of this work are:

- The introduction of a novel scheduled multi-task RL approach, the first of its kind to robustly handle varying task difficulties and mitigate negative transfer in practical applications.
- A unique method for evaluating task difficulty and adaptively scheduling training combined with a reset mechanism, significantly enhancing performance on more challenging tasks.
- Empirical evidence showing superior performance of our method compared to base algorithms on the Meta-

[1]School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea. Correspondence to: Youngchul Sung <ycsung@kaist.ac.kr>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

World benchmark, accompanied by an in-depth analysis of the underlying mechanisms.

2. Preliminaries

2.1. Multi-Task Reinforcement Learning

An RL task is modeled as a discrete-time finite-horizon Markov decision process (MDP), represented by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho, \gamma, H)$, where $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ denotes the state space, $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ the transition probability, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, $\rho : \mathcal{S} \to \mathbb{R}^+$ the initial state distribution, $\gamma \in [0, 1)$ the discount factor, and H the horizon. At each timestep t, the agent observes the current state $s_t \in \mathcal{S}$ of the environment and takes action $a_t \in \mathcal{A}$ based on its policy $\pi(a_t|s_t)$. Subsequently, the environment rewards the agent with r_t according to the reward function $r_t = r(s_t, a_t)$ and transitions to the next state s_{t+1} based on the transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$. The trajectory of length H is denoted by $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_H, a_H, r_H)$. Let $q_\pi(\tau)$ be its distribution under the transition probability $\mathcal{P}$ and policy $\pi(a_t|s_t)$.

The primary objective of conventional single-task RL is to maximize the expected discounted sum of rewards $J(\pi) = \mathbb{E}_{\tau\sim\rho_\pi}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t\right]$. On the other hand, in multi-task RL considered in this paper, the goal is not just to optimize the policy for a single task, but obtain a single policy that performs well over multiple tasks. Formally, we consider a set of tasks $\mathcal{C} = \{\mathcal{T}_i\}_{i=1}^N$ and a distribution $p(\mathcal{T})$ over $\mathcal{C}$, where each task $\mathcal{T}_i$ is characterized by a distinct MDP $\mathcal{M}_i = (\mathcal{S}, \mathcal{A}, \mathcal{P}_i, r_i, \rho_i, \gamma, H)$ described above and $p(\mathcal{T})$ is typically assumed to be uniform. The MDPs $\{\mathcal{M}_i\}_{i=1}^N$ share the same state and action spaces, but have different reward functions, transition probabilities, and initial state distributions. The goal of multi-task RL is to learn a single policy π that maximizes the average expected return across all tasks sampled from $p(\mathcal{T})$. Thus, the objective can be expressed as

$$\max_\pi \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}[J(\pi, \mathcal{T})], \tag{1}$$

where $J(\pi, \mathcal{T})$ is the expected return of a single task $\mathcal{T} \in \mathcal{C}$.

We will assume that the environment is fully observable, allowing the agent to access the current state s_t of the environment, and that the reward function is bounded above by $R_{\max}$, that is,

$$\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} r_i(s, a) = R_{\max} < \infty, \quad \forall i.$$

There exist several architectures for the single policy π handling multiple tasks. In this paper, we use a deep generalized policy $\pi_\theta(a|s, z)$ that takes a task representation z as input alongside the current state s, where θ is the policy parameter. The learned task representation replaces one-hot encoded task information. The details of our generalized policy architecture will be presented in Section 4. As the backbone algorithm to train the policy, a general RL algorithm can be adopted. In this paper, we use SAC (Haarnoja et al., 2018b) as our backbone algorithm.

2.2. Probabilistic Inference for Reinforcement Learning

Our algorithm requires a metric to measure the goodness of a policy π_θ for an RL task $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho, \gamma, H)$. Such a metric can be obtained from the work of Levine (2018). The basic idea is to check how far a trajectory τ generated by a policy π_θ is from an optimal trajectory for the task. For details, please refer to Levine (2018). Here, we briefly summarize the work relevant to our development. In (Levine, 2018), a fully observable MDP is represented as a control problem involving a probabilistic graphical model by introducing a binary random variable $\mathcal{O}$. Given a trajectory τ, $\{\mathcal{O} = 1\}$ denotes the event that τ is optimal, whereas $\{\mathcal{O} = 0\}$ denotes the event that τ is not optimal. Then, the following conditional probability is defined (Levine, 2018):

$$p(\mathcal{O} = 1 \mid \tau) = \exp\left(\sum_{t=1}^{H} \gamma^{t-1}(r_t - R_{\max})\right), \tag{2}$$

where $R_{\max}$ is the least upper bound of the reward function r. In addition, the prior distribution for τ is assigned as

$$p(\tau) = \rho(s_0) \prod_{t=1}^{H} \left[\mathcal{U}(a_t) p(s_{t+1}|s_t, a_t)\right], \tag{3}$$

where $\rho(s_0)$ and $p(s_{t+1}|s_t, a_t)$ are the true initial state distribution and transition probability, and $\mathcal{U}$ is the uniform distribution on $\mathcal{A}$.

The probability distribution $q_\theta(\tau)$ of τ generated by the policy π_θ is simply given by (Sutton & Barto, 1998)

$$q_\theta(\tau) = \rho(s_0) \prod_{t=1}^{H} \left[\pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)\right]. \tag{4}$$

We adopt $q_\theta(\tau)$ as our variational posterior distribution and attempt to minimize the distance between $q(\tau)$ and $p(\tau|\mathcal{O} = 1)$, where the latter is the posterior distribution of an optimal trajectory under our prior (3). In particular, the KL divergence between $q(\tau)$ and $p(\tau|\mathcal{O} = 1)$ can be computed as

$$\begin{aligned} &- \mathrm{KL}\left(q_\theta(\tau) \parallel p(\tau \mid \mathcal{O} = 1)\right) \\ &\quad = \mathbb{E}_{\tau\sim q_\theta}\left[\log p(\tau \mid \mathcal{O} = 1) - \log q_\theta(\tau)\right] \\ &\quad = \mathbb{E}_{\tau\sim q_\theta}\left[\sum_{t=1}^{H} \left(\gamma^{t-1} r_t - \log \pi_\theta(a_t|s_t)\right)\right] - C, \end{aligned} \tag{5}$$

where C is constant irrespective of the policy. Note that the first term corresponds to the expected return, and the second term corresponds to the expected entropy of policy $\pi_\theta(\cdot|s)$. Interestingly, the theory suggests that an optimal trajectory is a combination of return (exploitation) and exploration. A more detailed derivation of equation (5) is provided in Appendix E

2.3. Resetting Deep Networks

In deep single-task RL, learning tends to overfit early experiences, called the primacy bias, which is initially explored by Nikishin et al. (2022). To counter this bias, Nikishin et al. (2022) proposed a method that periodically implements resets of the RL agent's parameters while preserving the replay buffer. The key aspect of this innovative approach is that although the performance of the deep agent suddenly drops after reset, the performance quickly improves with good trajectories obtained in the later stage of learning thus eliminating the primacy bias. This approach has demonstrated marked improvements in diverse environments. Extending this concept, D'Oro et al. (2023) suggested modulating the frequency of resets based on the volume of updates rather than environmental steps, thereby enhancing sample efficiency through more regular resets in scenarios of elevated replay ratios. Advancing this line of research, Kim et al. (2023) introduced a novel reset mechanism that incorporates deep ensemble learning, mitigating the performance collapse after reset. Moreover, Schwarzer et al. (2023) refined the reset mechanisms and demonstrates human-level efficiency in Atari game environments.

3. Motivation: The Simplicity Bias in Deep Multi-Task RL

In this section, we present a motivating example that illustrates the challenges to conventional deep multi-task RL, which treats all target tasks with equal priority. The challenge is pronounced when the set of target tasks comprises tasks with different difficulties and stochastic gradient updates are used for training as most deep RL algorithms.

3.1. Heterogeneity in Task Complexity

In multi-task RL, tasks often vary significantly in complexity. For example, consider the Meta-World environment (Yu et al., 2019), which serves as a prime benchmark for multi-task RL. The Meta-World benchmark features a spectrum of tasks that demand varying amounts of samples and learning time. To quantify these differences, we conducted separate training sessions for four representative tasks—'reach,' 'drawer-close,' 'peg-insert-side,' and 'push'—using the same Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018b). The rendered images of the four tasks are shown in Figure 1. The first two tasks, 'reach' and 'drawer-close,' are much easier compared to the other two tasks, 'peg-insert-side' and 'push'. As a result, they demonstrate fast learning curves, as shown in Figure 2. On the other hand, the learning curve for the difficult task of 'push' is very slow and has high variance.

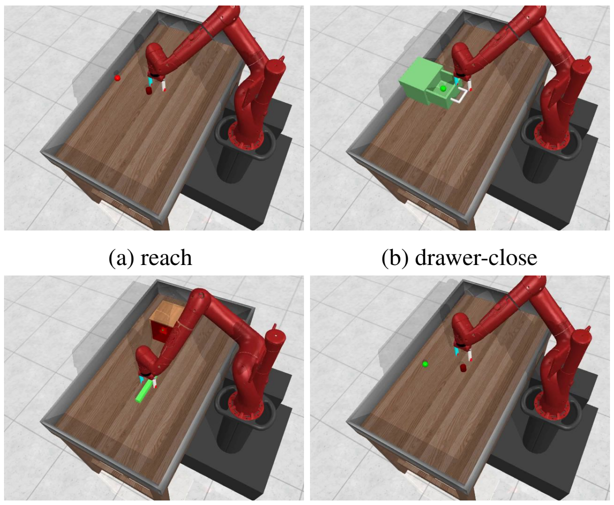

(a) reach (b) drawer-close (c) peg-insert-side (d) push

Figure 1. Rendered image of the four representative tasks—'reach,' 'drawer-close,' 'peg-insert-side,' and 'push'—of the Meta-World benchmark's MT10 benchmark.

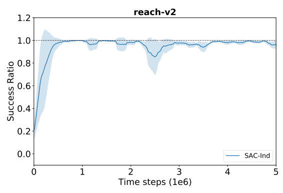
3.2. The Simplicity Bias

In the previous subsection, we observed that a set of tasks for multi-task RL is typically composed of tasks of various difficulties. Now, we want to see how the heterogeneity of task complexity affects the learning process of a conventional multi-task RL agent. For this, we selected two sets Γ_1 and Γ_2 of tasks, where each set is composed of two tasks and both sets include the difficult 'push' task. We compared the performance on the 'push' task under two different settings. The first set Γ_1 is composed of 'reach' and 'push' tasks, and the second set Γ_2 is composed of 'peg-insert-side' and 'push' tasks. Note that the difficult 'push' task is paired with the easier task of 'reach' in the first set, whereas the difficult 'push' task is paired with the relatively difficult task of 'peg-insert-side' in the second set. For each of these two settings, we trained the multi-task agent with a conventional multi-task objective derived from (1):

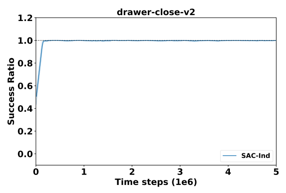
$$J_{MT}^{conv} = \frac{1}{|\Gamma_j|} \sum_{\mathcal{T}_i \in \Gamma_j} J(\pi_\theta, \mathcal{T}_i), \quad j = 1, 2, \tag{6}$$

under the assumption of uniformly distributed $p(\mathcal{T})$. The key aspect of the conventional multi-task objective (6) is assigning equal priority to all the tasks in the multi-task set.

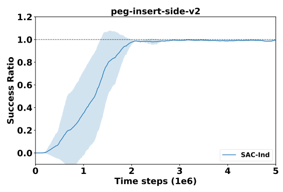
Figures 3(a) and 4(a) show the learning results for the set-



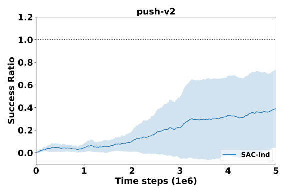
(a) reach



(b) drawer-close



(c) peg-insert-side



(d) push

Figure 2. Learning curves of four tasks: 'reach,' 'drawer-close,' 'peg-insert-side,' and 'push'. It takes much longer for the agent to succeed in the complex tasks, 'peg-insert-side' and 'push,' compared to the simpler tasks, 'reach' and 'drawer-close.'

tings Γ_1 and Γ_2, respectively. As seen in Figure 3(a), when the difficult 'push' task is paired with an easy 'reach' task, the agent completely fails to learn the difficult 'push' task. When the difficult 'push' task is paired with the relatively difficult 'peg-insert-side' task, on the other hand, the agent partially learns the difficult 'push' task as seen in Figure 4(a). Contrary to our initial intuition, the difficult 'push' is learned when it is paired with another difficult task, not when it is paired with an easy task. For more detailed examination of simplicity bias, a simple gridworld example is provided in Appendix F.

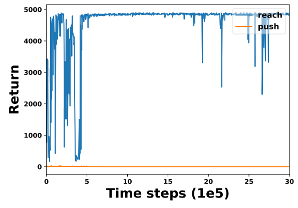
3.3. Gradient Magnitude Analysis of the Simplicity Bias

In order to investigate the cause of the phenomenon in the previous subsection, we delve further into the problem by examining the policy gradient. The policy gradient is the main quantity that directly affects policy updates for policy gradient or actor-critic methods. The policy gradient of the multi-task objective (6) is given by

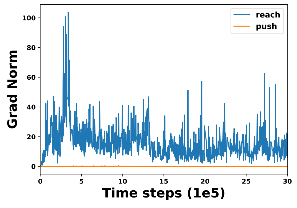
$$\nabla_\theta J_{MT}^{conv} = \frac{1}{|\Gamma_j|} \sum_{\mathcal{T}_i \in \Gamma_j} \nabla_\theta J(\pi_\theta, \mathcal{T}_i), \quad j = 1, 2. \tag{7}$$

We assume that the computation of the policy gradient is done stochastically based on mini-batches from replay buffers (Lillicrap et al., 2016b).

Figures 3(b) and 4(b) show the policy gradients $\nabla_\theta J_{\mathrm{SAC}}(\pi_\theta, \mathcal{T}_1)$ and $\nabla_\theta J_{\mathrm{SAC}}(\pi_\theta, \mathcal{T}_2)$ over time for the two sets Γ_1 and Γ_2, respectively. In the first setting, where the difficult 'push' task is paired with the easy 'reach' task, it is seen in Figure 3(b) that the gradient norm of the easy 'reach' task increases quickly because the episodic return for an easy task increases quickly, and the policy gradient is

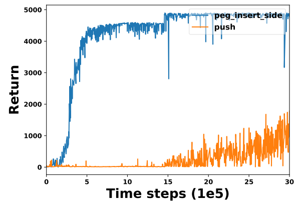


(a) Return

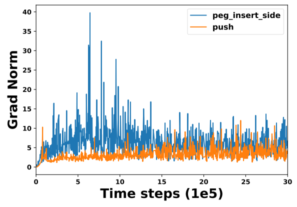


(b) Gradient norm

Figure 3. Learning curves for the 'reach'–'push' setting. When the discrepancy between the task complexity is large, the agent fails to succeed on the harder task, which is 'push' in this case.



(a) Return



(b) Gradient norm

Figure 4. Learning curves for the 'peg-insert-side'–'push' setting. When the discrepancy between the task complexity is moderate, the agent succeeds in both tasks.

the product of the expected return and the score in principle (Sutton & Barto, 1998). On the other hand, the gradient norm of the difficult 'push' task does not increase over time. This is because the large gradient of the easy task in the early stage of learning pushes the multi-task policy towards the optimal direction of the easy task and prevents the agent from learning the difficult task. In this way, the overall multi-task learning settles down to solving the easy task without much opportunity for learning the difficult task. We will refer to this phenomenon as the *simplicity bias in multi-task learning*. One might think that when the easy task is eventually fully learned, its gradient norm becomes small due to the score term, and the learning is more focused on the difficult task, allowing the gradient norm of the difficult task to eventually increase. However, as time goes on, the learning rate for stochastic gradient shrinks in general. Hence, policy updates are performed less frequently over time and the policy settles without permitting prolonged time for learning the difficult task.

On the other hand, Figure 4(b) shows that the gradient norm of the difficult 'push' task increases together with the gradient norm of another difficult task, 'peg-insert-side,' when the 'push' task is paired with the 'peg-insert-side' task. This is because both tasks are difficult, the return levels of both tasks are comparable in the early stage of learning and hence the simplicity bias does not occur.

As seen, the simplicity bias is a major obstacle in multi-task RL for a general set of tasks with diverse task complexity, leading to overfitting to easy tasks, failing to learn more complex tasks, and deteriorating the overall average performance. One right-away idea to cope with this situation is to apply different weights for different task gradients in (7), as done in other gradient manipulation works (Yu et al., 2020; Liu et al., 2021). However, determination of the gradient weights poses another design issue, which is not trivial. In the remainder of this paper, we will tackle the problem of 'how to effectively mitigate or overcome the simplicity bias in multi-task RL and achieve improved overall performance' from a totally different approach.

4. Scheduled Multi-Task Training for Multi-Task Reinforcement Learning

In this section, we describe the environment setting, baselines, and implementation details. To address the question raised in the previous section regarding the optimal structure and algorithm for effectively dealing with the varying task difficulties of multi-task RL, we propose the **S**cheduled **M**ulti-Task **T**raining (SMT) algorithm. This new approach is specifically designed to counteract the negative transfer in multi-task RL by prioritizing more challenging tasks, thus enhancing the overall learning efficiency. The complexity-based scheduling, which is our main component, is supported by two components: reset mechanism and budget-based training.

- **Complexity-Based Scheduling**: At its core, SMT employs a strategic scheduling approach that emphasizes the early resolution of more challenging tasks to prevent the overshadowing effect of simpler tasks. This strategy, underpinned by a novel metric for task difficulty assessment, ensures real-time task prioritization and optimal allocation of learning resources.
- **Reset Mechanism**: SMT incorporates a reset mechanism to address the simplicity bias. By periodically reinitializing network parameters while keeping the replay buffer, this mechanism maintains the model's adaptability and broadens its generalization capabilities across various tasks.
- **Budget-Based Training**: SMT divides the training budget into two stages for effective resource utilization. Initially focusing on tasks judged solvable, it later reallocates budgets toward previously unsolvable tasks, preventing waste of the budget.

The overall flow of our SMT framework is illustrated in Figure 5. SMT utilizes three deep neural networks: a policy network π_θ, a Q-value function network Q_ψ, and a task embedding network E_φ. For each task $\mathcal{T}_i$ in the task set $\mathcal{C}$, we maintain a distinct experience replay buffer $\mathcal{D}_i$. The objective is to maximize $\sum_{\mathcal{T}_i \in \mathcal{C}} J(\pi_\theta, \mathcal{T}_i)$ within a total budget of agent-environment interactions B_{total}. We discuss the aforementioned key components of SMT in the following subsections.

4.1. Complexity-Based Scheduling

Based on the observations we made in the previous section, two things are clear: (i) We should not train tasks with drastically different difficulties at the same time; and (ii) since difficult tasks require more training time, they should be prioritized. This naturally leads us to adopt a complexity-based scheduling scheme that simultaneously trains K most complex tasks at the moment. We will call this set of K tasks as the *training pool* $\mathcal{P}_t$, and call the set of the remaining tasks as the *main pool* $\mathcal{P}_m$. As the agent learns more and more about a task, it will progressively get easier, eventually returning back to the main pool. The vacancy in $\mathcal{P}_t$ would then be filled by a task in $\mathcal{P}_m$ that is most difficult at the moment. The complexity of a task is assessed by generating n_{eval} of trajectories using the policy every T_{eval} time-steps on each task and computing the average value of the metric (5).

The loss functions for the trainer are given by:

$$\ell^{\pi}(\theta) = \frac{1}{|\mathcal{C}|} \sum_{\mathcal{T}_i \in \mathcal{C}} \mathbb{E}_{s \sim \mathcal{D}_i} \left[\ell^{\pi}(\theta; s) \right],$$

$$\ell^{Q}(\psi) = \frac{1}{|\mathcal{C}|} \sum_{\mathcal{T}_i \in \mathcal{C}} \mathbb{E}_{(s,a) \sim \mathcal{D}_i} \left[\ell^{Q}(\psi; s, a) \right],$$

where $\ell^{\pi}(\theta; s)$ and $\ell^{Q}(\psi; s, a)$ are per sample actor and critic loss functions, respectively. Note that updates for tasks $\mathcal{T}_i \notin \mathcal{P}_t$ may still occur if their experience replay buffers $\mathcal{D}_i$'s are not empty.

4.2. Reset Mechanism

The problem with naive complexity-based scheduling is that selecting simple tasks at the early stages of the training process is inevitable because we do not have access to the task complexity *a priori*. Due to the simplicity bias, the policy and value networks are overfitted to simpler tasks at the early stages, severely hindering the training of more complex tasks that will appear later. To solve this problem, we adopt a reset mechanism proposed by Nikishin et al. (2022): reset the policy parameter θ and the Q-value function parameter ψ while leaving the experience replay buffers untouched.

4.3. Budget-Based Training

Another issue of naive complexity-based scheduling is that if an unsolvable task were sampled in the earlier stage of the process, the algorithm would fail to allocate enough resources to the training on other tasks. Tasks that might have been solvable if given sufficient samples, would be

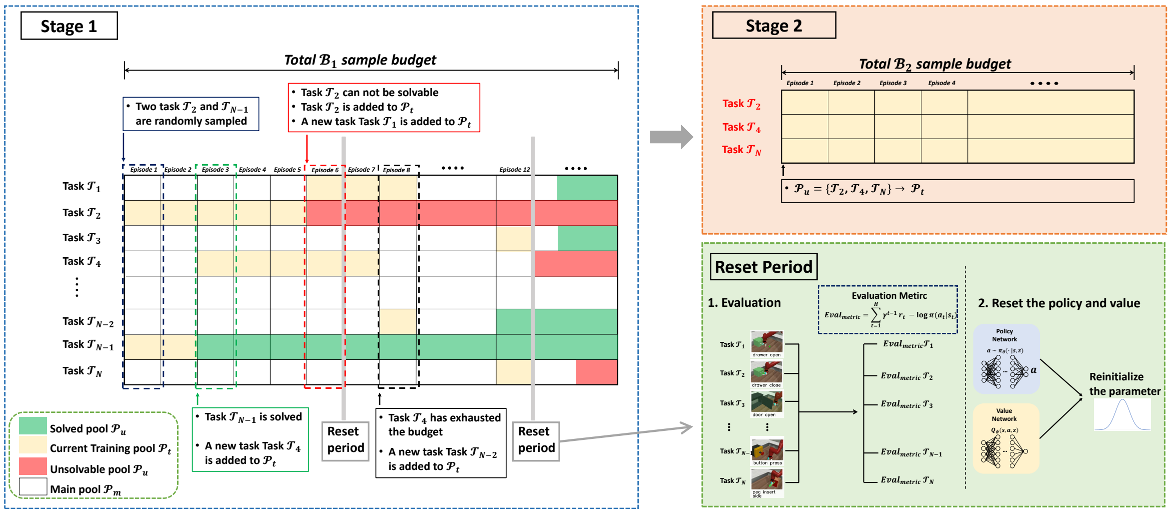

Stage 1
Total $\mathcal{B}_1$ sample budget
Two task $\mathcal{T}_2$ and $\mathcal{T}_{N-1}$ are randomly sampled
Task $\mathcal{T}_2$ can not be solvable
Task $\mathcal{T}_2$ is added to $\mathcal{P}_t$
A new task Task $\mathcal{T}_1$ is added to $\mathcal{P}_t$
Task $\mathcal{T}_{N-1}$ is solved
A new task Task $\mathcal{T}_4$ is added to $\mathcal{P}_t$
Reset period
Task $\mathcal{T}_4$ has exhausted the budget
A new task Task $\mathcal{T}_{N-2}$ is added to $\mathcal{P}_t$
Solved pool $\mathcal{P}_u$
Current Training pool $\mathcal{P}_t$
Unsolvable pool $\mathcal{P}_u$
Main pool $\mathcal{P}_m$
Stage 2
Total $\mathcal{B}_2$ sample budget
$\mathcal{P}_u = \{\mathcal{T}_2, \mathcal{T}_4, \mathcal{T}_N\} \rightarrow \mathcal{P}_t$
Reset Period
1. Evaluation
Evaluation Metirc
2. Reset the policy and value
Policy Network
Value Network
Reinitialize the parameter

Figure 5. Overview of the SMT framework. The complexity-based scheduling in Stage 1 is supported by the reset mechanism and budget-based training in Stage 2. For further explanation, please see Appendix H.

left unsolved, resulting in a poor final performance. To prevent this event from happening, we assign a budget b_i for each task $\mathcal{T}_i \in \mathcal{P}_t$. Whenever the agent interacts with the environment $\mathcal{M}_i$ for b_i time-steps, we check its training performance by computing the mean return for the n_{train} most recent training trajectories. If the training performance is below a predefined threshold m, the task is considered *unsolvable*. That task is removed from $\mathcal{P}_t$ and moved to the pool $\mathcal{P}_u$ of unsolvable tasks. Since the total budget is fixed, we set the per-task budgets b_i adaptively as κB where B is the remaining budget and $0 < \kappa < 1$ is a hyperparameter.

To further boost the sample efficiency, we also adopt an early-stopping mechanism for tasks that are already solved. If the mean return of the n most recent training trajectories for a task $\mathcal{T}_i$ exceeds a certain predefined threshold M, we consider it to be *solved* and move $\mathcal{T}_i$ to the pool $\mathcal{P}_s$ of solved tasks.

In the later stages of the algorithm, per-task budgets will not be large enough to correctly assess the task's solvability. Therefore, we must give the tasks in $\mathcal{P}_u$ a second chance. We do this by splitting B_{total} into two parts: B_1 for Stage 1 and B_2 for Stage 2, satisfying $B_1 + B_2 = B_{\text{total}}$. When Stage 1 is finished, that is, the agent has interacted with the environment for B_1 time-steps, we progress to Stage 2 by training all the tasks in $\mathcal{P}_u$ together, anticipating potential positive transfers.

To conclude, all tasks in $\mathcal{C}$ are categorized into four mutually exclusive pools: the main pool $\mathcal{P}_m$, the training pool $\mathcal{P}_t$, the unsolvable pool $\mathcal{P}_u$, and the solved pool $\mathcal{P}_s$. The main contribution of our work is the novel task scheduling, thus any type of policy network can be used for practical implementation. In our research, we utilize the VariBAD policy structure (Zintgraf et al., 2020) to extract pertinent information from each task, detailed in Appendix D. The overall process of the proposed method is illustrated in Figure 5 and is summarized in Algorithm 1

5. Experiments

Environment. In order to test how well the proposed method can perform on multiple complex tasks, we tested our approach with the Meta-World benchmark (Yu et al., 2019), which has 50 distinct robotic control tasks with a sawyer arm in the MuJoCo environment (Todorov et al., 2012). Our experiments use two modes, MT10 and MT50, with 10 and 50 manipulation tasks, respectively, from the benchmark, as shown in Figure 7. The task sets of MT10 and MT50 are given in Appendix A.

Baselines. We compared the proposed method with the baseline methods: 1) SAC with multi-task (SAC-MT): A shared policy with a one-hot task identification encoding and current state as input. 2) SAC with Multi-task Multi-head (SAC-MT-MH) (Yu et al., 2019): It is similar to SAC with multi-task but has an independent final layer in the policy network for each task (multi-head). 3) SAC with soft modularization (SAC-soft-modular) (Yang et al., 2020): Policy with multiple modules with soft modularization technique that gives a routing strategy for each task. 4) Gradient Surgery for Multi-Task Learning (PCGrad) (Yu et al., 2020): This approach addresses conflicting gradients in multi-task learning by projecting gradients to a shared subspace, improving overall learning efficiency and performance. 5) Parameter-compositional multi-task reinforcement learning (PaCo) (Sun et al., 2022): This method leverages parameter compositionality to enable efficient multi-task learning by

Table 1. Comparisons of per-task and average success ratios (%) of the Meta-World MT10 benchmark. For the task name corresponding to each task ID, see Appendix A.

	Task ID										
Algorithm	0	1	2	3	4	5	6	7	8	9	Average
SAC-MT	98±2.4	0±0.0	0±0.0	**100±0.0**	**100±0.0**	**100±0.0**	96±2.0	0±0.0	**100±0.0**	**100±0.0**	69.4 ± 0.8
SAC-MT-MH	**100±0.0**	28±21.8	0±0.0	**100±0.0**	98±2.5	**100±0.0**	**100±0.0**	46±21.8	**100±0.0**	**100±0.0**	77.2 ± 11.9
Soft Modular	**100±0.0**	32±14.9	0±0.0	**100±0.0**	**100±0.0**	**100±0.0**	**100±0.0**	12±14.9	**100±0.0**	**100±0.0**	74.4 ± 10.5
PCGrad	94±3.7	0±0.0	0±0.0	**100±0.0**	**100±0.0**	**100±0.0**	**100±0.0**	54± 39.9	**100±0.0**	**100±0.0**	74.8 ± 13.7
PaCo	**100±0.0**	44±25.2	0±0.0	**100±0.0**	**100±0.0**	**100±0.0**	**100±0.0**	80 ± 40	**100±0.0**	**100±0.0**	82.4 ± 14.2
SMT (Ours)	96±3.7	62±17.9	**34±13.8**	**100±0.0**	**100±0.0**	**100±0.0**	**100±0.0**	**76±31.9**	**100±0.0**	**100±0.0**	**86.8 ± 8.6**

composing task-specific parameters from a shared set of base parameters.

In all experiments, the policy, task embedding network, decoder network, and soft Q-function are implemented as neural networks with two hidden layers of size of 400 and ReLU activation. The output size of the task embedding network is 8 (dimension of the representation space Z).

5.1. Results on Meta-World

We report the results on MT10 and MT50 in Table 1 and 2, respectively. These results validate the efficacy of the SMT algorithm's innovative approach to multi-task learning in RL, particularly its dynamic task prioritization, reset mechanism, and strategic budget allocation, which collectively contribute to significant performance improvements across diverse tasks in the Meta-World benchmark.

Results on MT10. Referring to the MT10 results in Table 1, our SMT algorithm demonstrates superior performance across the board, achieving the highest average success rate of 86.8%. This represents a significant improvement over existing approaches. Notably, the SMT algorithm excels in tasks that other methods struggle with, such as task ID 1 ('push') and 7 ('peg-insert-side'). This underscores the effectiveness of the SMT's task prioritization and scheduling, which focuses on addressing more challenging tasks early in the training process.

Results on MT50. For the MT50 results in Table 2, the pattern of SMT's effectiveness continues. It outperforms other methods, especially noticeable in the average bottom-k success ratios. For instance, in the more challenging subset of tasks (bottom-20), while all baselines struggle, SMT shows a clear advantage.

5.2. Analysis of the Proposed Scheduling Approach

In this section, to evaluate the effectiveness of the proposed scheduling method, we explore the changes in task-learning sequences facilitated by this scheduling strategy. We have a total interaction budget of $\mathcal{B}$, and for every timestep, each task $\mathcal{T}_i$ undergoes $\mathcal{S}_i$ interactions with the agent. The rate of current budget utilization for the task $\mathcal{T}_i$ is thus expressed as $\mathcal{S}_i/\mathcal{B}$. Accordingly, under this scheduling strategy, tasks that are selected with greater frequency have a higher budget usage rate, while those chosen less frequently have a lower rate of budget use.

Table 2. Comparison of average bottom-k success ratios of the Meta-World benchmark MT50 benchmark.

	Average Bottom-k Success Ratio (%)				
Algorithm	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$
SAC-MT	**0.0 ± 0.0**	3.7 ± 5.6	21.0 ± 13.8	40.7 ± 10.4	52.6 ± 8.3
SAC-MT-MH	**0.0 ± 0.0**	4.5 ± 6.1	26.1 ± 14.6	44.0 ± 11.0	55.2 ± 8.8
Soft Modular	**0.0 ± 0.0**	1.8 ± 3.7	23.7 ± 12.3	42.6 ± 9.5	54.1 ± 7.6
PCGrad	**0.0 ± 0.0**	**0.0 ± 0.0**	21.0 ± 12.9	39.9 ± 10.7	51.9 ± 8.5
PaCo	**0.0 ± 0.0**	4.6 ± 8.2	26.1 ± 15.0	44.6 ± 11.2	55.6 ± 9.1
SMT (Ours)	**0.0 ± 0.0**	**8.0 ± 8.9**	**26.8 ± 13.1**	**45.0 ± 9.9**	**56.0 ± 8.0**

Figure 6 shows the learning curve for the MT10 benchmark. As seen in Figure 6, the agent utilizes a higher number of training samples for challenging tasks such as 'push,' 'pick-place,' and 'peg-insert-side'. Conversely, simpler tasks such as 'drawer-close' and 'window-open', require fewer sample budgets compared to the more demanding tasks.

5.3. Ablation Studies

First, we conducted ablation studies on the key hyperparameters of SMT: κ, M, m, B_1 with the MT10 benchmark. The hyperparameter κ represents the size of the training pool $\mathcal{P}_t$, while m and M denote the thresholds for classifying tasks as unsolvable and solvable, respectively. Additionally, B_1 signifies the allocated budget for Stage 1.

The ablation result is shown in Table 3, showing the average success ratio according to the task set configuration in the MT10 benchmark. There, $\mathcal{C}_{\text{easy}}$ denotes the set of relatively

Table 3. Ablation studies on the hyperparameters

	Hyperparameters								
	κ		M		m		B_1		Default
Task Set	0.7	0.9	2000	3000	500	2000	1.5×10^7	2.0×10^7	
$\mathcal{C}_{\text{easy}}$	**99.6 ± 0.1**	95.6± 3.7	62.5± 28.3	83.2± 11.9	92.7± 5.8	99.5± 0.2	94.3±4.2	99.5±0.2	99.3±0.3
$\mathcal{C}_{\text{difficult}}$	50.3±34.3	**69.8±19.6**	54.1±27.9	61.1±18.4	68.2±22.3	63.3±25.8	67.6±23.4	66.2±10.2	68.0±16.1
$\mathcal{C}$	79.8±15.5	85.3±9.8	59.1±23.6	74.4±12.7	82.9±9.4	85.1±9.9	83.6±8.9	82.9±10.0	**86.8 ±8.6**

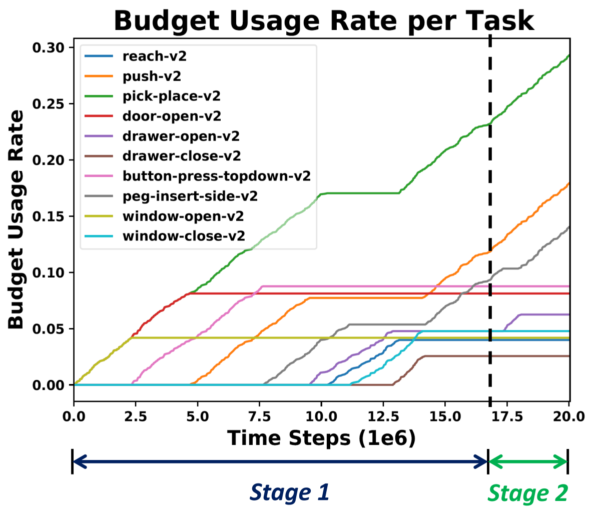


Figure 6. Learning curve for the budget usage rate ($\mathcal{S}_i/\mathcal{B}$) for each task $\mathcal{T}_i$ within the MT10 benchmark. For more complex tasks, the agent tends to utilize more training samples. This is particularly evident in tasks such as 'pick-place,' 'push,' and 'peg-insert-side'.

Table 4. Ablation study on the reset mechanism

	Algorithm	
Task set	SMT w/o reset	SMT
$\mathcal{C}_{\text{easy}}$	88.7±7.1	99.3±0.3
$\mathcal{C}_{\text{difficult}}$	43.8±31.8	68.0±16.1
$\mathcal{C}$	70.7 ±17.4	86.8 ±8.6

Table 5. Ablation study on the scheduling method

	Scheduling Method		
Task Set	Easy Tasks First	Random	Hard Tasks First(Ours)
$\mathcal{C}_{\text{easy}}$	88.7±7.1	89.8±4.3	99.3±0.3
$\mathcal{C}_{\text{difficult}}$	23.8±36.4	26.2±44.5	68.0±16.1
$\mathcal{C}$	62.7±17.4	62.6±21.3	86.8±8.6

easy tasks within the MT10 benchmark comprising 'reach', 'door-open', 'drawer-open', 'window-open', and 'window-close' tasks, whereas the remaining tasks form the set of difficult tasks $\mathcal{C}_{\text{difficult}}$. The ablation result shows that the performance is robust against m and B_1 and is relatively dependent on κ and M. The reason for the dependence of κ is that the amount of samples used to solve a given task is determined by the value of κ, underscoring the importance of setting an appropriate κ. More ablation studies are provided in Appendix G

Next, we conducted ablation studies on the reset mechanism. Table 4 shows the average success ratio based on the presence or absence of the reset mechanism. With the reset mechanism, we have 22 % performance increase. The result shows that the reset mechanism is important in our SMT framework.

Finally, to verify the importance of the "Hard Tasks First" scheduling scheme, we conducted experiments on various scheduling methods, including random task selection, easy tasks first, and hard tasks first. Table 5 shows the average success ratio for each scheduling method. As shown, scheduling with the hard tasks first outperforms other scheduling approaches, highlighting the effectiveness of our method.

6. Related Works

Multi-task RL. Multi-task learning has emerged as a critical domain within machine learning, aiming to develop algorithms that excel across a broad spectrum of tasks. This endeavor has proven to be particularly pivotal in RL, where the goal is to construct models capable of mastering diverse tasks (Wilson et al., 2007; Pinto & Gupta, 2017; Zeng et al., 2018; Hausman et al., 2018; Yang et al., 2020). Unlike single-task learning, multi-task RL thrives on exploiting the shared knowledge between tasks to foster generalization, a concept that has also seen significant adoption in computer vision to improve feature extraction and generalization (Zhang et al., 2014; Dai et al., 2016; Liu et al., 2019). The primary challenge lies in optimizing the knowledge transfer across tasks to boost learning efficiency without

Algorithm 1 Scheduled Multi-Task Training (SMT)

```
1: Initialize policy network $\pi_\theta$, $Q$-value network $Q_\psi$, task embedding network $E_\varphi$
2: Initialize replay buffers $\mathcal{D}_i$ for each task $\mathcal{T}_i \in \mathcal{C}$
3: $(\mathcal{P}_u, \mathcal{P}_s) \leftarrow (\varnothing, \varnothing)$
4: Randomly sample $K$ tasks to form $\mathcal{P}_t$
5: $\mathcal{P}_m \leftarrow \mathcal{C} \setminus \mathcal{P}_t$
6: Set a budget for each task in $\mathcal{P}_t$ as $\kappa B_{\text{total}}$.
7: for $B \leftarrow B_{\text{total}}, B_{\text{total}} - 1, \ldots, B_1 + 1$ do
8:   for all $\mathcal{T}_i \in \mathcal{P}_t$ do
9:     Interact with the environment and store data in $\mathcal{D}_i$
10:    if Training performance of $\mathcal{T}_i > M$ then
11:      Move $\mathcal{T}_i$ to $\mathcal{P}_s$
12:    end if
13:    if $\mathcal{T}_i$ exhausts its budget then
14:      if Training performance of $\mathcal{T}_i < m$ then
15:        Move $\mathcal{T}_i$ to $\mathcal{P}_u$
16:      else
17:        Move $\mathcal{T}_i$ to $\mathcal{P}_m$
18:      end if
19:    end if
20:  end for
21:  Update $\theta$, $\psi$, and $\varphi$ using the data in $\{\mathcal{D}_i\}$
22:  if $B_{\text{total}} - B \equiv T_{\text{reset}} - 1 \pmod{T_{\text{reset}}}$ then
23:    Evaluate the performance of $\pi_\theta$ on each tasks by rolling out $n_{\text{eval}}$ trajectories.
24:    Randomly reinitialize $\theta$ and $\psi$
25:  end if
26:  while $|\mathcal{P}_t| < K$ do
27:    Sample a task $\mathcal{T}_i$ with the lowest evaluation performance, move it to $\mathcal{P}_t$, and set its budget to $\kappa B$.
28:  end while
29: end for
30: for $B \leftarrow B_1, B_1 - 1, \ldots, 1$ do
31:   for all $\mathcal{T}_i \in \mathcal{P}_u$ do
32:     Interact with the environment and store data in $\mathcal{D}_i$
33:   end for
34:   Update $\theta$, $\psi$, and $\varphi$ using the data in $\{\mathcal{D}_i\}$
35: end for
```

succumbing to negative transfer, where progress in one task may inadvertently degrade performance in others.

Addressing Negative Transfer. Addressing negative transfer is paramount to the success of multi-task RL. Strategies to mitigate this issue encompass a range of techniques. (i) Distillation methods leverage policy distillation to amalgamate insights from multiple tasks into a unified model, albeit typically requiring a separate network for each task, thus increasing the resource overhead (Rusu et al., 2016a; Parisotto et al., 2016; Teh et al., 2017). (ii) Modular networks employ distinct modules for different tasks, with potential task-specific routing to manage parameter sharing and minimize task interference. This method allows for strategic parameter sharing, curbing the adverse effects of negative transfer (Rusu et al., 2016b; Devin et al., 2017; Andreas et al., 2017; Haarnoja et al., 2018a; Yang et al., 2020; Sun et al., 2022). (iii) Gradient-based techniques focus on analyzing gradient signals across tasks to identify and eliminate elements that could lead to negative transfer. Although promising, the variability and noise inherent in task gradients can complicate their effective utilization for this purpose (Zhang & Yeung, 2013; Chen et al., 2018; Hu et al., 2019; Yu et al., 2020). In contrast to the previous approaches, we train a task-embedding network alongside a novel learning algorithm specifically designed to counteract negative transfer. By quantifying the negative effects and dynamically adjusting the learning process, our method offers a stable and effective solution to the challenges of multi-task RL.

7. Conclusion

In this work, we introduced the Scheduled Multi-Task Training (SMT), addressing the critical challenge of negative transfer in multi-task RL. By strategically prioritizing more complex tasks, SMT effectively enhances learning efficiency and performance across a diverse set of tasks. Our novel dynamic task prioritization strategy, guided by a metric for assessing task difficulty, ensures that training resources are allocated where they are most needed, thereby significantly improving learning outcomes. Moreover, the incorporation of a reset mechanism to periodically reinitialize key network parameters mitigates the simplicity bias, enhancing the adaptability and robustness of the learning process. Future work could explore the integration of SMT with other RL algorithms, further refinement of the task difficulty assessment metric, and the application of SMT to real-world multi-task learning problems. Additionally, investigating the impact of different reset mechanisms and task prioritization strategies on learning efficiency and adaptability presents an exciting avenue for research. Ultimately, the SMT algorithm opens up new possibilities for advancing the field of multi-task RL.

Impact Statement

One of the current limitations of RL is generalization. That is, the policy trained for a specific task has difficulty in performing well for a different task. Thus, training a policy that can perform well for multiple similar tasks is a vital issue in RL. In this paper, we proposed a new effective algorithm based on scheduling for multi-task RL, which mitigates the negative transfer among multiple tasks and enhances overall performance. Our result can contribute to application of RL to many real-world control problems that require solving multiple similar tasks.

Acknowledgments

This work was supported in part by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2022-0-00124, Development of Artificial Intelligence Technology for Self-Improving Competency-Aware Learning Capabilities, 50%) and in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2021R1A2C2009143 Information Theory-Based Reinforcement Learning for Generalized Environments, 50%).

References

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 166–175. PMLR, 2017.

Caruana, R. Multitask learning. *Mach. Learn.*, 28(1):41–75, 1997.

Chen, Z., Badrinarayanan, V., Lee, C., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 793–802. PMLR, 2018.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1724–1734. ACL, 2014.

Dai, J., He, K., and Sun, J. Instance-aware semantic segmentation via multi-task network cascades. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 3150–3158. IEEE Computer Society, 2016.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 2169–2176. IEEE, 2017.

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P., Bellemare, M. G., and Courville, A. C. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1582–1591. PMLR, 2018.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pp. 6244–6251. IEEE, 2018a.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865. PMLR, 2018b.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. A. Learning an embedding space for transferable robot skills. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Hu, H., Dey, D., Hebert, M., and Bagnell, J. A. Learning anytime predictions in neural networks via adaptive loss balancing. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 3812–3821. AAAI Press, 2019.

Kim, W., Shin, Y., Park, J., and Sung, Y. Sample-efficient and safe deep reinforcement learning via reset deep ensemble agents. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10-16, 2023*, 2023.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held*

December 3-6, 2012, Lake Tahoe, Nevada, United States, pp. 1106–1114, 2012.

Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *CoRR*, abs/1805.00909, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016a.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016b. URL http://arxiv.org/abs/1509.02971.

Liu, B., Liu, X., Jin, X., Stone, P., and Liu, Q. Conflict-averse gradient descent for multi-task learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 18878–18890, 2021.

Liu, S., Johns, E., and Davison, A. J. End-to-end multi-task learning with attention. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 1871–1880. Computer Vision Foundation / IEEE, 2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P., and Courville, A. C. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16828–16847. PMLR, 2022.

Parisotto, E., Ba, L. J., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Pinto, L. and Gupta, A. Learning to push by grasping: Using multiple tasks for effective learning. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 2161–2168. IEEE, 2017.

Rusu, A. A., Colmenarejo, S. G., Gülçehre, Ç., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. Policy distillation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016a.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *CoRR*, abs/1606.04671, 2016b.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 1889–1897. JMLR.org, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Schwarzer, M., Ceron, J. S. O., Courville, A., Bellemare, M. G., Agarwal, R., and Castro, P. S. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pp. 30365–30380. PMLR, 2023.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

Sun, L., Zhang, H., Xu, W., and Tomizuka, M. Paco: Parameter-compositional multi-task reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21495–21507, 2022.

Sun, X., Panda, R., Feris, R., and Saenko, K. Adashare: Learning what to share for efficient deep multi-task learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Sutton, R. S. and Barto, A. G. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 978-0-262-19398-6.

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 4496–4506, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 5026–5033. IEEE, 2012.

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pp. 1015–1022. ACM, 2007.

Yang, R., Xu, H., Wu, Y., and Wang, X. Multi-task reinforcement learning with soft modularization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 2019.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., and Funkhouser, T. A. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pp. 4238–4245. IEEE, 2018.

Zhang, Y. and Yeung, D. A regularization approach to learning task relationships in multitask learning. *ACM Trans. Knowl. Discov. Data*, 8(3):12:1–12:31, 2013.

Zhang, Z., Luo, P., Loy, C. C., and Tang, X. Facial landmark detection by deep multi-task learning. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*, volume 8694 of *Lecture Notes in Computer Science*, pp. 94–108. Springer, 2014.

Zintgraf, L. M., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: A very good method for bayes-adaptive deep RL via meta-learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

A. MT10 and MT50 Environments

In our experiment, we make two benchmarks MT10 and MT50 that contain 10 and 50 distinct environments from the 50 environments of Meta-World (Yu et al., 2019), respectively. Figure 7 and 8 illustrates every task in the MT10 and MT50. Note that the illustrated figures are all derived from rendered images within the Meta-World environment (Yu et al., 2019).

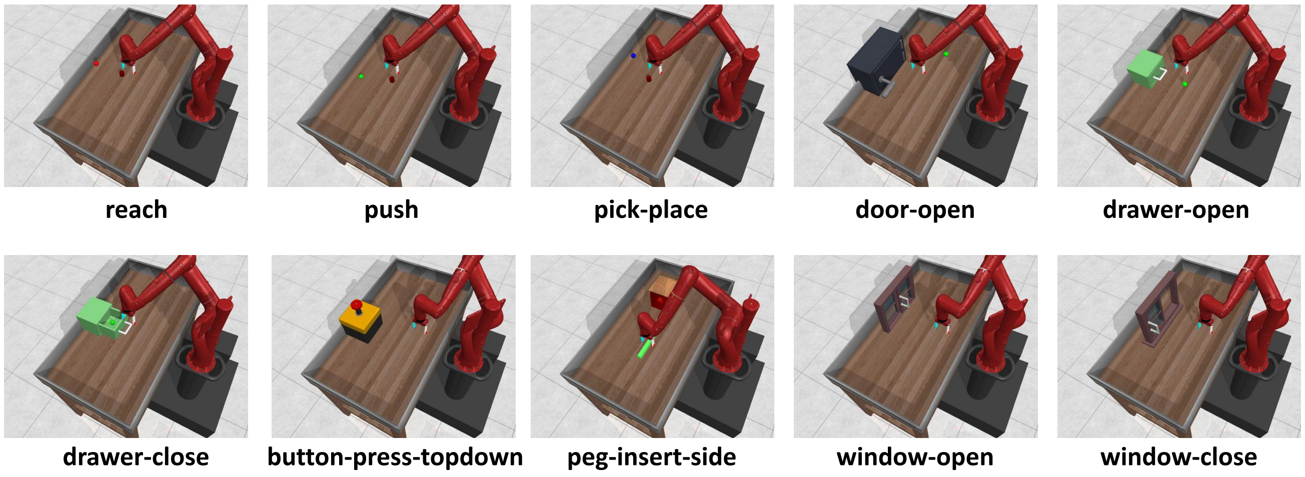


Figure 7. MT10: multi-task benchmark with 10 distinct tasks.

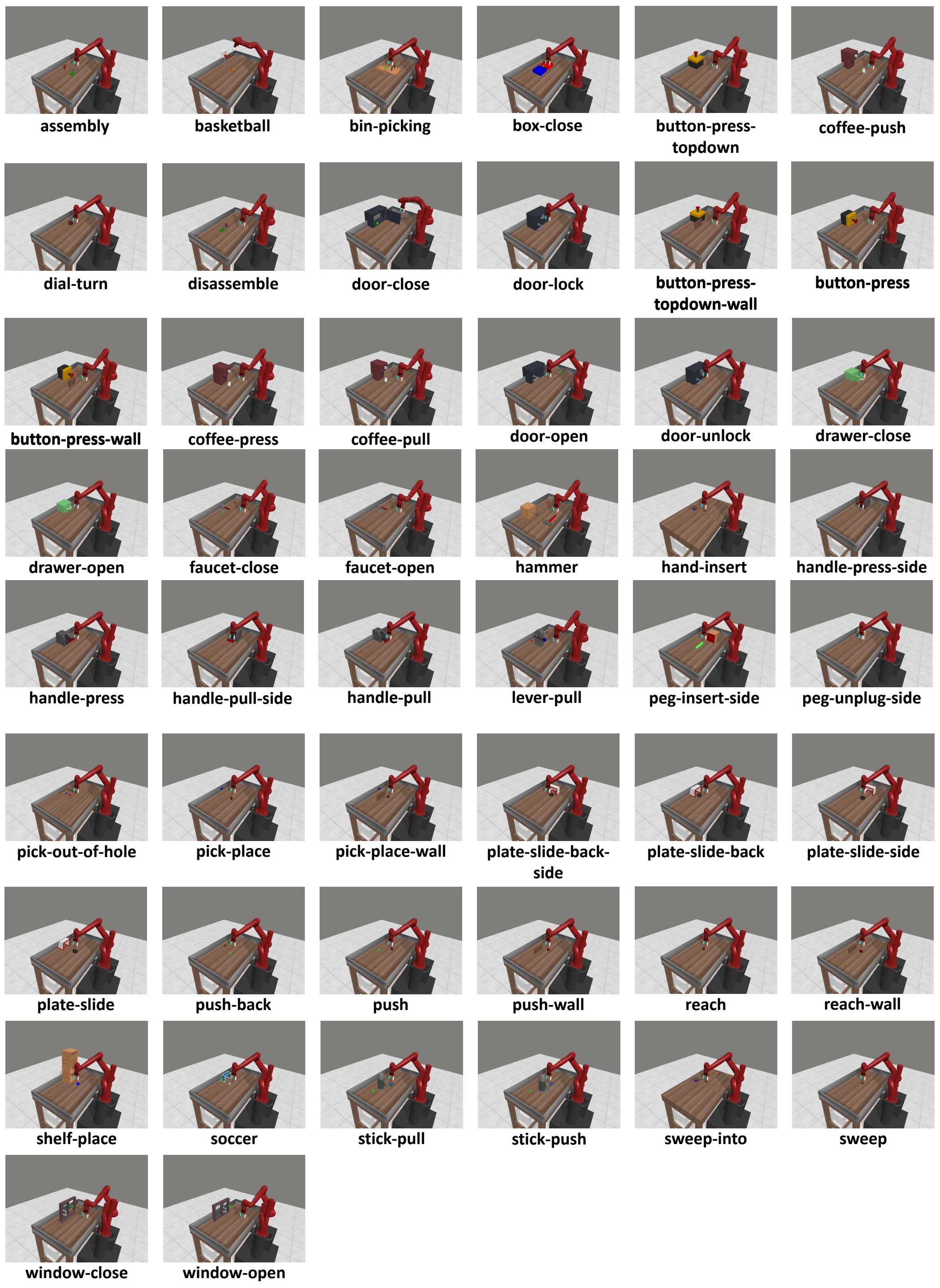


Figure 8. MT50: multi-task benchmark with 50 distinct tasks.

B. MT50 Full Experiment Result

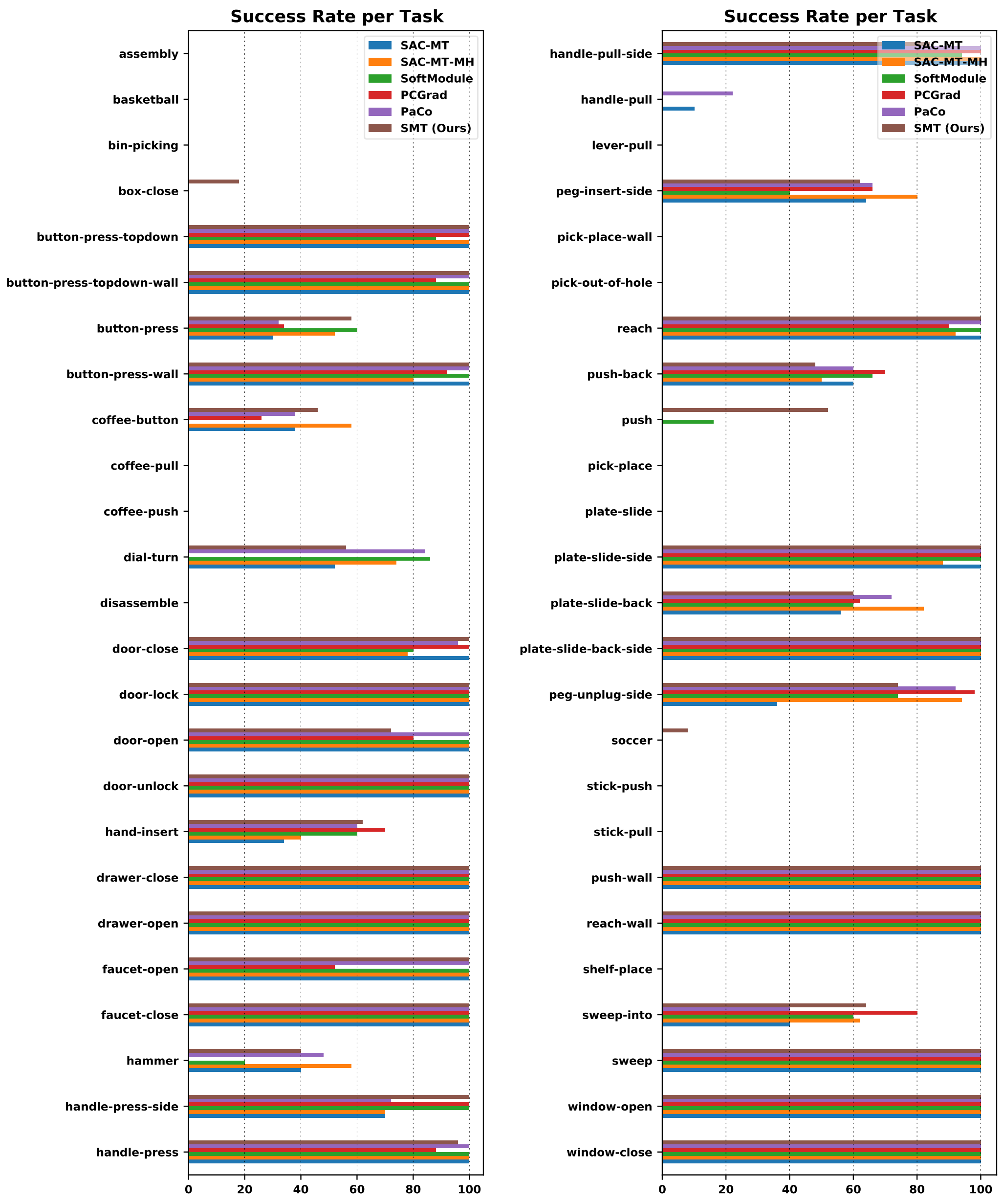


Figure 9. **Meta-World MT50 success rate.** We report the final success rates of the baselines and our method for training tasks of the Meta-World MT50 benchmark.

C. Hyperparameters

Table 6. **Hyperparameters of SMT**. Hyperparameters of SMT used for Meta-World MT10 and MT50 along with the notations in the manuscript.

Hyperparameters	**MT10**	**MT50**
Training steps B_{total}	2×10^7	1×10^8
Discount factor	0.99	0.99
Minibatch size per Task	256	256
Optimizer (all)	Adam	Adam
Optimizer (all) : learning rate	0.0003	0.003
Networks (MLP) : activation	ReLU	ReLU
Networks (MLP) : n. hidden layers	2	2
Networks (MLP) : hidden units	400	400
Networks (GRU) : hidden units	256	256
Dim(h)	4	8
Replay Buffer Size per Task	1×10^6	2×10^5
Target network update period	1	1
τ	0.005	0.005
Stage 1 Budgets B_1	1.7×10^7	8.5×10^7
Stage 2 Budgets B_2	3×10^6	1.5×10^7
κ	0.8	0.7
K	3	8
Reset Interval (time steps) T_{reset}	5×10^5	1×10^7
Scheduling Interval (time steps)	1×10^3	1×10^3

D. Practical Policy Implementation

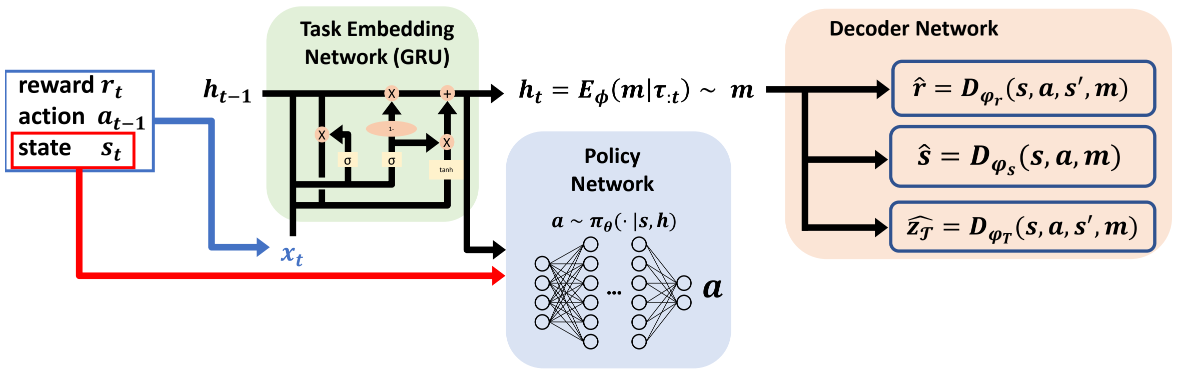

reward r_t
action a_{t-1}
state s_t
h_{t-1}
Task Embedding Network (GRU)
x_t
$h_t = E_\phi(m|\tau_{:t}) \sim m$
Policy Network
$a \sim \pi_\theta(\cdot|s,h)$
a
Decoder Network
$\hat{r} = D_{\varphi_r}(s,a,s',m)$
$\hat{s} = D_{\varphi_s}(s,a,m)$
$\widehat{z_\mathcal{T}} = D_{\varphi_\mathcal{T}}(s,a,s',m)$

Figure 10. Overall structure of networks: Policy network together with a task embedding network and three decoding networks.

In our approach, we adopt a generalized policy, denoted as $\pi_\theta(a|s,h)$, which is parameterized by θ as depicted in Figure 10. Additionally, we utilize a task embedding network implemented by GRU (Cho et al., 2014), $m \sim E_\phi(h_t|\tau_{:t})$, which is parameterized by ϕ, along with three decoder networks: $D_{\varphi_r}(s,a,s',m)$ for reward reconstruction, $D_{\varphi_s}(s,a,m)$ for state reconstruction, and $D_{\varphi_\mathcal{T}}(s,a,s',m)$ for task prediction. Here, the proposed networks are shared across all tasks. At each time, the task embedding network takes the input of the history $\tau_{:t} = \{s_i, a_{i-1}, r_i\}_{i=1}^t$, and outputs the representation $m \sim E_\phi(h_t|\tau_{:t})$. Typically, tasks are distinguished by their IDs through a one-hot vector, as seen in approaches like SAC-MT. Nevertheless, with our proposed scheduling method, training is exclusively focused on the tasks within $\mathcal{P}_t$, so that the policy might not be exposed to the one-hot task IDs of other tasks not included in $\mathcal{P}_t$. Consequently, this could lead to diminished performance on tasks that are unseen in the evaluation period. Hence, the extraction of task-specific information becomes crucial. Moreover, unlike the VariBAD architecture, we can determine which tasks have been introduced, and to utilize on this for extracting task information, we've augmented VariBAD's original Decoder with a task prediction feature aimed at learning to predict the one-hot task ID.

E. Detailed Derivation of Equation (5)

The derivation is based on (Levine, 2018). Let us first assume that we have access to the true initial state distribution $\rho(s_0)$ and the true state transition probability $p(s_{t+1} \mid s_t, a_t)$. Given a trajectory τ, we introduce a binary random variable $\mathcal{O}$, where $\{\mathcal{O} = 1\}$ denotes the event that τ is optimal and $\{\mathcal{O} = 0\}$ denotes the event that τ is not optimal. We choose the distribution over $\mathcal{O}$ to be defined by the equation

$$p(\mathcal{O} = 1 \mid \tau) = \exp\left(\sum_{t=1}^{H} \gamma^{t-1}(r(s_t, a_t) - R_{\max})\right),$$

where $R_{\max}$ is the least upper bound of the reward function r. We assign a prior on the trajectory space, defined by the equation

$$p(\tau) = \rho(s_0) \prod_{t=1}^{H} \left[\mathcal{U}(a_t) p(s_{t+1}|s_t, a_t)\right],$$

where $\mathcal{U}$ is the uniform distribution on the action space $\mathcal{A}$. Then, the probability distribution of optimal trajectories can be written as

$$p(\tau \mid \mathcal{O} = 1) = \frac{p(\mathcal{O} = 1 \mid \tau)p(\tau)}{p(\mathcal{O} = 1)} = \frac{p(\mathcal{O} = 1 \mid \tau)p(\tau)}{\int p(\mathcal{O} = 1 \mid \tau)p(\tau)\, \mathrm{d}\tau}$$

The computation of the integral $\int p(\mathcal{O} = 1 \mid \tau)p(\tau)\, \mathrm{d}\tau$ is intractable, so instead we aim to find a policy π that produces optimal trajectories.

The probability distribution $q_\theta(\tau)$ of τ generated by a policy π_θ, which is parametrized by a parameter θ, can be written as

$$q_\theta(\tau) = \rho(s_0) \prod_{t=1}^{H} \left[\pi_\theta(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)\right].$$

Our goal is to find the optimal parameter θ^* that minimizes the KL divergence between $q_\theta(\tau)$ and $p(\tau \mid \mathcal{O}=1)$, that is,

$$\theta^* = \arg\min_\theta \mathrm{KL}(q_\theta(\tau) \parallel p(\tau \mid \mathcal{O}=1)).$$

Simplifying the equation, we obtain the following:

$$\begin{aligned}
&\mathrm{KL}\left(q_\theta(\tau) \parallel p(\tau \mid \mathcal{O}=1)\right)\\
&= \mathbb{E}_{\tau\sim q_\theta}\left[\log \frac{q_\theta(\tau)p(\mathcal{O}=1)}{p(\mathcal{O}=1 \mid \tau)p(\tau)}\right]\\
&= \mathbb{E}_{\tau\sim q_\theta}\left[\log \frac{\rho(s_0)\left[\prod_{t=1}^H \pi_\theta(a_t \mid s_t)p(s_{t+1} \mid s_t, a_t)\right]p(\mathcal{O}=1)}{\exp\left(\sum_{t=1}^H \gamma^{t-1}(r_t - R_{\max})\right)\rho(s_0)\left[\prod_{t=1}^H \mathcal{U}(a_t)p(s_{t+1} \mid s_t, a_t)\right]}\right]\\
&= \mathbb{E}_{\tau\sim q_\theta}\left[\log \frac{\left[\prod_{t=1}^H \pi_\theta(a_t \mid s_t)\right]p(\mathcal{O}=1)}{\exp\left(\sum_{t=1}^H \gamma^{t-1}(r_t - R_{\max})\right)\left[\prod_{t=1}^H \mathcal{U}(a_t)\right]}\right]\\
&= \mathbb{E}_{\tau\sim q_\theta}\left[\sum_{t=1}^H \log \pi_\theta(a_t \mid s_t) - \sum_{t=1}^H \gamma^{t-1} r_t\right]\\
&\quad + \mathbb{E}_{\tau\sim q_\theta}\left[\log p(\mathcal{O}=1) + \sum_{t=1}^H \gamma^{t-1}R_{\max} + \sum_{t=1}^H \log|\mathcal{A}|\right]
\end{aligned}$$

where $\mathcal{A}$ is the Lebesgue measure of $\mathcal{A}$. Note that $\log p(\mathcal{O}=1)$, $\sum_{t=1}^H \gamma^{t-1}R_{\max}$, and $\sum_{t=1}^H \log|\mathcal{A}|$ are constant with respect to τ, so we can remove the expectation.

$$\begin{aligned}
&\mathrm{KL}\left(q_\theta(\tau) \parallel p(\tau \mid \mathcal{O}=1)\right)\\
&= \mathbb{E}_{\tau\sim q_\theta}\left[\sum_{t=1}^H \log \pi_\theta(a_t \mid s_t) - \sum_{t=1}^H \gamma^{t-1} r_t\right] + \log p(\mathcal{O}=1) + \sum_{t=1}^H \gamma^{t-1}R_{\max} + H\log|\mathcal{A}|
\end{aligned}$$

Since $\log p(\mathcal{O}=1)$, $\sum_{t=1}^H \gamma^{t-1}R_{\max}$, and $\sum_{t=1}^H \log|\mathcal{A}|$ are constant with respect to θ, we have

$$\begin{aligned}
\theta^* &= \arg\min_\theta \mathbb{E}_{\tau\sim q_\theta}\left[\log \pi_\theta(a_t \mid s_t) - \sum_{t=1}^H \gamma^{t-1} r_t\right]\\
&= \arg\max_\theta \mathbb{E}_{\tau\sim q_\theta}\left[\sum_{t=1}^H \gamma^{t-1} r_t - \log \pi_\theta(a_t \mid s_t)\right].
\end{aligned}$$

Note that for each $t = 1, 2, \ldots, H$,

$$\mathbb{E}_{\tau\sim q_\theta}\, lr[-\log \pi_\theta(a_t \mid s_t)] = \mathbb{E}_{s_t\sim q_\theta}\left[\mathbb{E}_{a\sim\pi_\theta(\cdot|s_t)}\left[-\log \pi_\theta(a \mid s_t)\right]\right] = \mathbb{E}_{s_t\sim q_\theta}[H(\pi_\theta(\cdot \mid s_t))],$$

where H is the entropy function. Therefore, we have

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau\sim q_\theta}\left[\sum_{t=1}^H \gamma^{t-1} r_t + \sum_{t=1}^H H(\pi_\theta(\cdot \mid s_t))\right].$$

Since the expectation can be replaced by the empirical mean of trajectories sampled from the MDP using the policy π_θ, the objective can be estimated even when we do not have access to the true initial state distribution $\rho(s_0)$ and transition probability $p(a_t \mid s_t)$.

Interestingly, the first term corresponds to the expected return and the second term corresponds to the expected entropy of policy $\pi_\theta(\cdot|s)$. The theory tells that we should consider both return(exploitation) and entropy(exploration) in order to find a policy that is optimal under the Bayesian framework.

F. A Simple Grid-World Example of Simplicity Bias

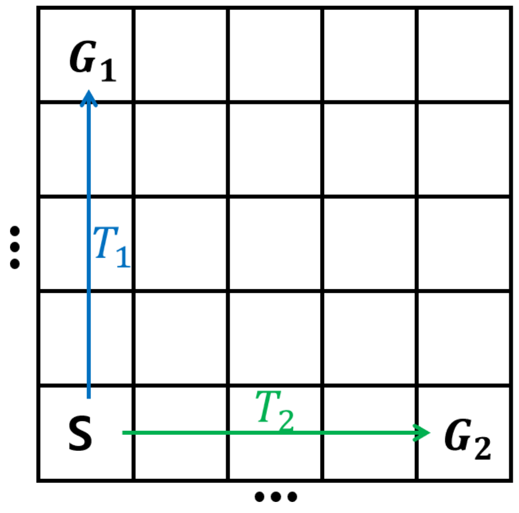


Figure 11. Simple Grid-world Toy example for Multi-task RL

To strengthen our motivation regarding "**simplicity bias**", we conducted additional experiments in a simple grid-world environment shown in Figure 11. This environment features two tasks, $\mathcal{T}_1$ and $\mathcal{T}_2$, each with the goal of reaching different endpoints, G_1 and G_2, respectively, starting from a common point S.

We conducted two separate experiments with this simple grid-world environment. In the first experiment, we designed two tasks: $\mathcal{T}_1$ and $\mathcal{T}_2$ so that $\mathcal{T}_1$ is easier with a dense reward setting and $\mathcal{T}_2$ is more challenging with a sparse reward setting, thereby introducing varying levels of task difficulty. The second experiment is designed so that both $\mathcal{T}_1$ and $\mathcal{T}_2$ are of sparse reward setting, aligning their difficulty levels. The result of these experiments is shown in the Figure 12. The result shows that overfitting to the relatively easier task complicates the learning process for the more difficult task. Indeed, the simplicity bias is a significant challenge within the realm of Multi-Task Reinforcement Learning (MTRL).

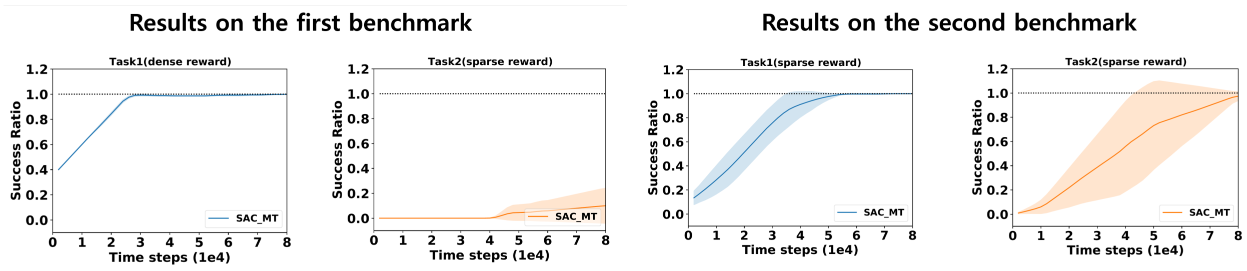


Figure 12. Result of two different experiments on the Simple Grid-world

G. Additional Ablation Studies

G.1. Ablation Study on the Hyperparameter κ

As seen in Table 3, performance varies significantly depending on the hyperparameter κ compared to other hyperparameters. The reason is as follows:

Consider the first main training phase B_1 . In our current sample time assignment method, the sample time assigned to a scheduled task at current time is $\kappa * (B_1 - B_{\text{used}})/K$, where B_{used} is the used sample time and K is the number of scheduled tasks. So, initially, $\kappa * B_1/K$ sample time is assigned to each scheduled task, and we want that this time is reasonably long to train a hard task successfully. Anyway, a task's successful training is continuously checked by measuring the latest five episode return average. So, an easy task ending quickly leaves the training pool, moves to the successful pool, and a new task is scheduled. However, with too large a κ, too difficult unsolvable tasks scheduled earlier will consume most of B_1 and there remains not many samples for easy tasks later. So, there exists a trade-off between hard tasks and easy tasks.

Our additional ablation study on κ, shown in Table 7 and Figure 13 clearly shows this trade-off. As κ increases, the performance of difficult tasks increases while that of easy tasks decreases. However, we observe that the performance is not so sensitive and there exist quite a wide range of soft spots over 0.8 to 0.9. Please note that our performance over the whole range of κ is better than other baselines as shown in the right figure of Figure 13.

Table 7. Ablation studies on the hyperparameter κ

	κ							
Task Set	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0
$\mathcal{C}_{\text{easy}}$	**99.8 ± 0.1**	99.6 ± 0.1	**99.8± 0.1**	99.3± 0.3	99.4± 0.3	95.6± 3.7	91.7± 6.4	90.8± 7.2
$\mathcal{C}_{\text{difficult}}$	42.1 ± 28.9	50.3±34.3	56.7±28.3	68.0±16.1	66.9±18.7	**69.8±19.6**	68.3±17.5	69.0±18.4
$\mathcal{C}$	76.7±16.2	79.8±14.5	82.6±9.8	**86.8±8.6**	86.4±12.7	85.3±9.8	82.3±10.3	82.1±9.9

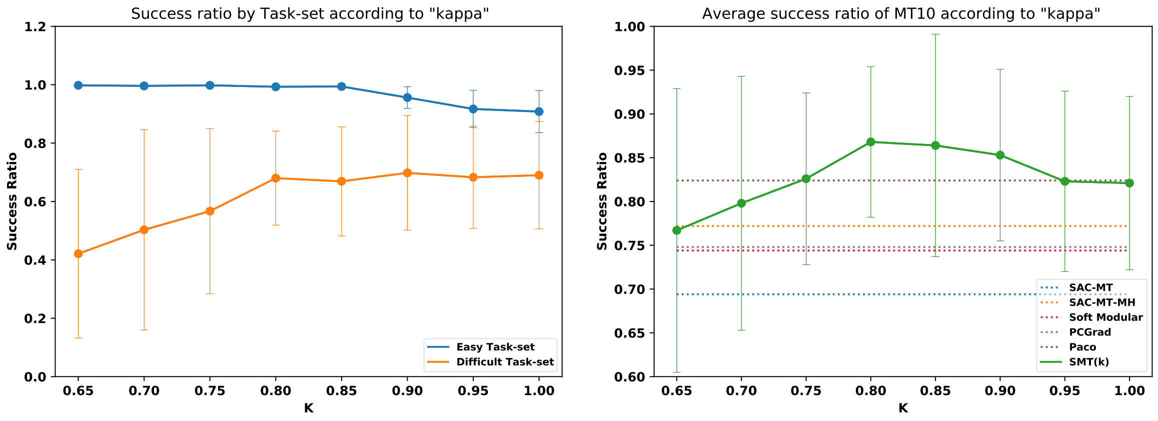


Figure 13. Results depending on the hyperparameter κ

G.2. Ablation Study on the Hyperparameter n_{eval}

We conducted further experiments with various values for the hyperparameter n_{eval} ranging from $[1, 5, 10, 20, 50]$, where the main paper adopted 5. The result of this experiment is shown in Table 8 and Figure 14. As evident from the result, the performance does not significantly change with variations in n_{eval}.

Table 8. Ablation studies on the hyperparameter n_{eval}

	n_{eval}				
Task Set	1	5	10	20	50
$\mathcal{C}_{\text{easy}}$	**99.8 ± 0.1**	99.3 ± 0.3	99.1± 0.4	99.4± 0.2	99.5± 0.2
$\mathcal{C}_{\text{difficult}}$	61.1±18.5	68.0±16.1	**70.2±14.4**	68.1±15.3	69.3±16.5
$\mathcal{C}$	84.3±10.9	86.8±8.6	**87.5±8.1**	86.9±8.3	87.4±9.0

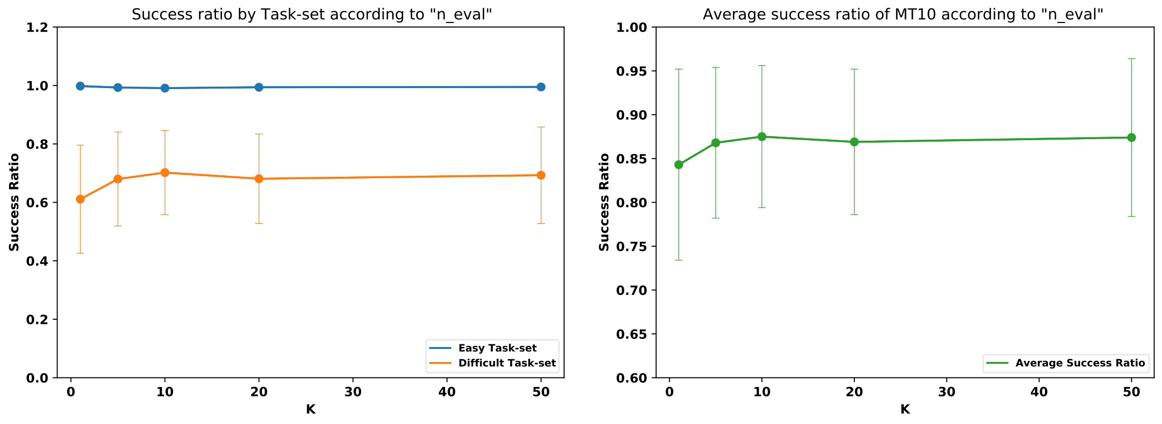


Figure 14. Result depending on the hyperparameter n_{eval}

G.3. Integrating of the SMT Framework Using Soft Modular Method

We applied our SMT framework, including the scheduling method and reset mechanism, to the Soft Modular approach and denoted this variant as SMT (Soft Modular). The results in the MT10 benchmark is presented in Table 9. From the result, we confirm that our SMT framework can be effectively integrated into existing MTRL algorithms.

Table 9. Results of the SMT Framework Using the Soft Modular Method

	Algorithm		
Task Set	Soft Modular	SMT(Ours)	SMT(Soft Modular)
$\mathcal{C}_{\text{easy}}$	99.7±0.1	99.3±0.3	99.6±0.2
$\mathcal{C}_{\text{difficult}}$	36.0±25.4	68.0±16.1	69.4±15.2
Average Success Ratio	74.4±10.5	86.8±8.6	87.5±9.9

H. Comprehensive Overview of Stage 1

We will add more explanation on the overall operation of our method in Figure 15. The brief explanation is as follows:

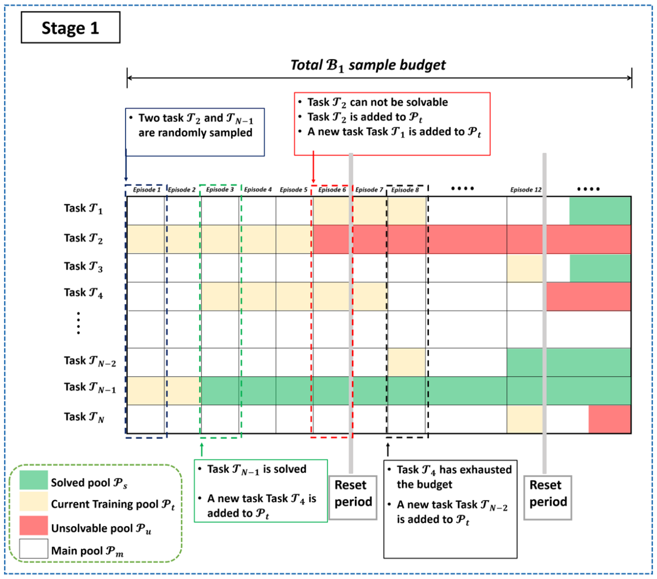

Stage 1
Total $\mathcal{B}_1$ sample budget
• Two task $\mathcal{T}_2$ and $\mathcal{T}_{N-1}$ are randomly sampled
• Task $\mathcal{T}_2$ can not be solvable
• Task $\mathcal{T}_2$ is added to $\mathcal{P}_t$
• A new task Task $\mathcal{T}_1$ is added to $\mathcal{P}_t$
Episode 1
Episode 2
Episode 3
Episode 4
Episode 5
Episode 6
Episode 7
Episode 8
Episode 12
Task $\mathcal{T}_1$
Task $\mathcal{T}_2$
Task $\mathcal{T}_3$
Task $\mathcal{T}_4$
Task $\mathcal{T}_{N-2}$
Task $\mathcal{T}_{N-1}$
Task $\mathcal{T}_N$
• Task $\mathcal{T}_{N-1}$ is solved
• A new task Task $\mathcal{T}_4$ is added to $\mathcal{P}_t$
Reset period
• Task $\mathcal{T}_4$ has exhausted the budget
• A new task Task $\mathcal{T}_{N-2}$ is added to $\mathcal{P}_t$
Reset period
🟩 Solved pool $\mathcal{P}_s$
🟨 Current Training pool $\mathcal{P}_t$
🟥 Unsolvable pool $\mathcal{P}_u$
⬜ Main pool $\mathcal{P}_m$

Figure 15. Overview of the Stage 1 in the SMT framework.

In Stage 1, we introduce four distinct pools: the training task pool $\mathcal{P}_t$, the main pool $\mathcal{P}_m$, the solved task pool $\mathcal{P}_s$, and the unsolved task pool $\mathcal{P}_u$. To understand how Stage 1 operates, let us consider the example in Figure 15, where we can simultaneously train up to $K = 2$ tasks.

(1) Initially, before Stage 1, all tasks reside within the main pool $\mathcal{P}_m$. At the beginning of Stage 1, we randomly select two tasks, say, $\mathcal{T}_2$ and $\mathcal{T}_{N-1}$, from the main pool $\mathcal{P}_m$ and transfer them to the training task pool $\mathcal{P}_t$. Concurrently, we allocate a budget and collect trajectories for each task in $\mathcal{P}_t$, which currently consists of two tasks: $\mathcal{T}_2$ and $\mathcal{T}_{N-1}$. We perform gradient updates based on the collected trajectories.

(2) Upon completion of two episodes, task $\mathcal{T}_{N-1}$ is deemed solved, as it surpasses our threshold M. It is thus moved to the solved task pool $\mathcal{P}_s$. To fill the vacancy in $\mathcal{P}_t$ caused by the departure of $\mathcal{T}_{N-1}$, another task is selected from the start of the 3rd episode. In this example, $\mathcal{T}_4$ is randomly selected from the main pool $\mathcal{P}_m$.

(3) Following the end of the fifth episode, the entire budget for $\mathcal{T}_2$ is expended. We then assess the solvability of $\mathcal{T}_2$ based on the threshold m.

In this scenario, the training return of $\mathcal{T}_2$ failed to exceed the threshold m, leading to its transfer to the unsolved task pool $\mathcal{P}_u$. Then, another task, $\mathcal{T}_1$, is randomly selected from the main pool $\mathcal{P}_m$ and transfer it to $\mathcal{P}_t$.

(4) After the end of the sixth episode, which marks the reset period, we evaluate all tasks using the trained policy by sampling n_{eval} trajectories. Subsequently, random task selection is finished (random task selection is only for the time before the first reset) and tasks are selected from the main pool $\mathcal{P}_m$ based on the evaluation metric defined by equation (5) of the paper under the principle of hard task first. Note that the evaluation metric computation period is synchronized with the reset period for implementation simplicity. After reset, scheduling is based on the renewed evaluation metric values.