# EFFICIENT MULTI-TASK REINFORCEMENT LEARNING VIA SELECTIVE BEHAVIOR SHARING

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

## ABSTRACT

The ability to leverage shared behaviors between tasks is critical for sample efficient multi-task reinforcement learning (MTRL). Prior approaches based on parameter sharing or policy distillation share behaviors uniformly across tasks and states or focus on learning one optimal policy. Therefore, they are fundamentally limited when tasks have conflicting behaviors because no one optimal policy exists. Our key insight is that, we can instead share exploratory behavior which can be helpful even when the optimal behaviors differ. Furthermore, as we learn each task, we can guide the exploration by sharing behaviors in a task and state dependent way. To this end, we propose a novel MTRL method, Q-switch Mixture of policies (QMP), that learns to selectively shares exploratory behavior between tasks by using a mixture of policies based on estimated discounted returns to gather training data. Experimental results in manipulation and locomotion tasks demonstrate that our method outperforms prior behavior sharing methods, highlighting the importance of task and state dependent sharing. Videos are available at https://sites.google.com/view/qmp-mtrl.

## 1 INTRODUCTION

Imagine we are learning to acquire a diverse but relevant set of skills at the same time, such as cooking various recipes. Certain skills may require interacting with the same appliances such as fridges and microwaves, responding to similar events (*e.g.* oven timer goes off or water boils), and navigating common paths across the kitchen. We as humans can naturally exploit the shared information from learning each skill. Specifically, we excel at leveraging the *behavioral overlap* when learning relevant skills together and therefore can accelerate multi-task learning through mutual exploration.

Can we replicate this stunning yet natural capability of humans — learning while sharing overlapped behaviors, and devise a framework that allows for efficiently acquiring a set of skills simultaneously? Prior work in behavior sharing for multi-task reinforcement learning (MTRL) has explored sharing a policy, directly across tasks or as a constraint to learn similar behaviors (Teh et al., 2017; Ghosh et al., 2018). However, these methods are designed to share behavior uniformly between tasks, which fundamentally limits them to acquire skills that require different optimal behaviors from the same state. To address such limitation, a more flexible MTRL framework is needed, where an agent can selectively learn to share behaviors from different tasks only when the optimal task behaviors coincide and avoid sharing when they conflict.

Thus, our goal is to achieve a generally applicable behavior sharing approach for efficient MTRL, capable of working over task sets even with conflicting behaviors. Our key insight is that an agent's behavior across tasks can be helpful for exploration during training, even if the final policies diverge. For example, while learning multiple recipes, our household robot can learn to check the refrigerator and pantry for ingredients first instead of exploring the entire kitchen. To this end, we propose to share exploratory behaviors across tasks.

There are two key challenges in sharing exploratory behaviors for MTRL. First, the agent must decide which behaviors to share and when. For different states, other task policies would be disparately helpful or harmful. Moreover, as the agent learns its task, the effect of behavior sharing from other tasks must reduce. Second, an effective mechanism is needed to incorporate exploratory behaviors from other tasks, instead of optimal behaviors as was done in prior works.

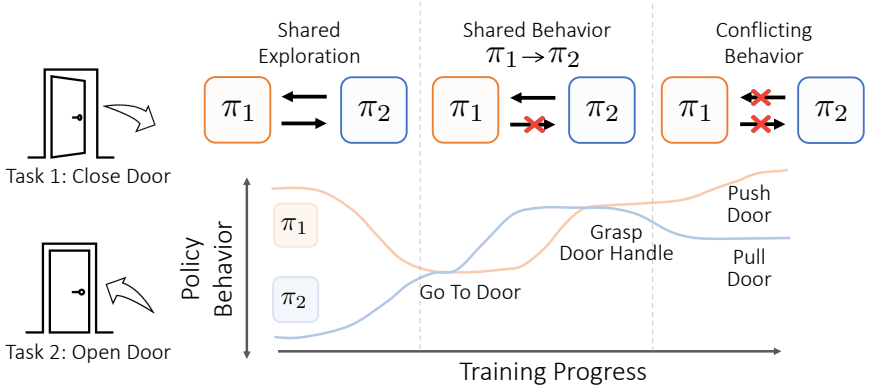



Figure 1: The policies for two relevant tasks $\pi_1$ and $\pi_2$ can benefit from selective behavior sharing even if the optimal behavior looks different. Initially, the policies can benefit from shared exploration to learn to go to the door. As $\pi_2$ learns to grasp the door handle first, it can share its behavior with $\pi_1$. Finally, the policies diverge to pushing and pulling the door, and can no longer share behavior.

To address these challenges, we aim to devise a framework which enables behavior sharing that is responsive to different states, non-uniform over other tasks, and adaptive to training progress. To this end, we propose Q-switch Mixture of Policies (QMP), a novel MTRL framework where each task policy selectively explores the actions proposed by other task policies. First, we use a task and state-dependent mixture of all task policies as the behavioral policy for each task. Second, we define the mixture gating function or Q-switch, which utilizes each task's learned action-value or Q-function to determine which proposed behaviors to share and when. This enables us to selectively share behaviors across tasks and be robust to conflicting behaviors in the task set.

We show empirically in manipulation and navigation tasks that our method outperforms uniform behavior sharing methods, demonstrating the importance of task and state dependent sharing. We further show that our method is complementary with parameter sharing, a popular MTRL strategy, demonstrating the effectiveness of behavior sharing in conjunction with other forms of MTRL.

The main contributions of this paper can be summarized as follows:

- We identify and empirically demonstrate the limitation of existing behavior sharing methods in multi-task reinforcement learning, which are designed to share behaviors uniformly between tasks.
- We present a method, Q-switch Mixture of policies (QMP), that learns to selectively share behaviors to accelerate exploration and therefore improve the sample efficiency.
- We design and conduct experiments in manipulation and navigation domains, which demonstrate that our proposed method can learn to effectively share behavior between tasks for accelerated exploration and highlight the important of *selective* behavior sharing.

## 2 Related Work

### 2.1 Multi-Task Reinforcement Learning for Diverse Task Families

Multi-task learning in diverse task families is susceptible to "negative transfer" between dissimilar tasks that hinders training. Prior works combat this by measuring task relatedness through validation loss on tasks (Liu et al., 2022) or influence of one task to another (Fifty et al., 2021), contributions from each of the individual tasks to the agent's updates (Hessel et al., 2019), or the similarities in task gradients (Yu et al., 2020). These works aim to avoid negative transfer while our work aims to improve sample efficiency through selective sharing.

### 2.2 EXPLORATION IN MULTI-TASK REINFORCEMENT LEARNING

To facilitate exploration in multiple tasks, various methods have been proposed. Bangaru et al. (2016) proposed to encourage agents to increase their state coverage by providing an exploration bonus. Yet, such exploration bonus is not task-directed and therefore may not be efficient. Zhang & Wang (2021) studied sharing information between agents to encourage exploration under tabular MDPs with a regret guarantee. Leveraging a policy from other tasks as an exploration policy has been explored in (Kalashnikov et al., 2021). However, it requires predefined task affinities and is unable to adapt through the course of training. Exploration in multi-task imitation learning also has been studied in (Mandlekar et al., 2020). While we share the motivation of improving exploration in multi-task reinforcement learning with the abovementioned methods, our proposed method does not require pre-defined task similarity measure or exploration bonus; instead, it is designed to learn to selectively share behaviors while directly learning from tasks.

### 2.3 SHARING IN MULTI-TASK REINFORCEMENT LEARNING

There are multiple mostly complementary avenues of how to share information in MTRL, including sharing data, sharing parameters or representations, and sharing behaviors. In offline MTRL, prior works show that selectively sharing data between tasks can be helpful (Yu et al., 2021; 2022). Our work similarly shows that selective sharing is important but in the online setting and in sharing behaviors. Sharing parameters across policies is powerful technique that can speed up MTRL by learning shared representations (Xu et al., 2020; D'Eramo et al., 2020). Many works in this area define architectures that are flexible and suitable for multitask learning (Yang et al., 2020; Sodhani et al., 2021; Misra et al., 2016; Perez et al., 2018; Devin et al., 2017) and can be easily combined with other types of information sharing. Most similar to our work, Teh et al. (2017) and Ghosh et al. (2018) share behaviors between multiple policies through policy distillation and regularization to the distilled policy. However, unlike our work, they share behavior uniformly between policies and aim to learn one optimal policy.

### 2.4 USING Q-FUNCTIONS AS FILTERS

Yu et al. (2021) uses Q-functions to filter which data should be shared between tasks in a multi-task setting. In the imitation learning setting, (Nair et al., 2018) and (Sasaki & Yamashina, 2020) use Q-functions to filter out low quality demonstrations so they are not used for training. In both cases, the Q-function is used to evaluate some data that can be used for training. In contrast, our method uses a Q-function to evaluate different policies that may be helpful for exploration.

## 3 PROBLEM FORMULATION

Multi-task learning aims to learn multiple related tasks jointly so that the knowledge from one task can be leveraged by other tasks to improve their performance (Zhang & Yang, 2021). Specifically, for multi-task reinforcement learning (MTRL), leveraging shared behaviors between tasks is critical to improving sample efficiency. However, prior behavior sharing approaches are limited to task sets that do not require conflicting behaviors. Therefore, our key problem is to leverage behavior sharing such that it can achieve sample-efficient MTRL in a variety of task families.

### 3.1 MULTI-TASK REINFORCEMENT LEARNING WITH BEHAVIOR SHARING

We aim to simultaneously learn a multi-task family consisting of $T$ tasks. Each task $T_i$ is a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}_i, \rho_i, \gamma)$, with shared state space $\mathcal{S}$, action space $\mathcal{S}$, transition probabilites $\mathcal{T}$, and discount factor $\gamma$. The reward function $\mathcal{R}_i$ and initial state distribution $\rho_i$ varies by task. All tasks share an agent and environment, meaning the state and action space should have consistent semantic meaning across all tasks. We do not make the assumption that optimal task behaviors coincide. Since, the optimal behaviors, $\pi_1^*(s)$ and $\pi_2^*(s)$, of two tasks can be conflicting, then $R_1(s, \pi_2^*(s)) < R_1(s, \pi_1^*(s))$ and $R_2(s, \pi_1^*(s)) < R_2(s, \pi_2^*(s))$ any action will be sub-optimal for Task 1, Task 2, or both. Thus any form uniform behavior sharing is limited.

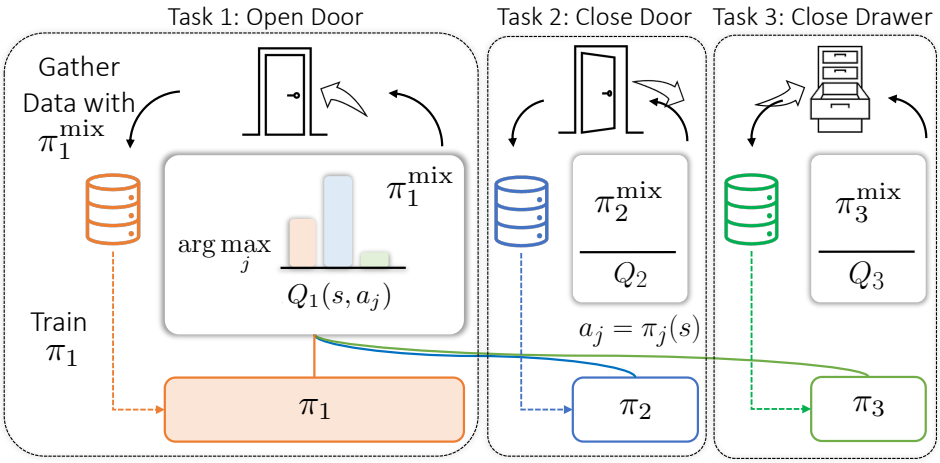



Figure 2: Our method (QMP) trains a policy for each task and shares behavior by gathering training data using a task-specific mixture of policies. The mixture is gated by the Q-switch, which picks the policy that has the highest Q-value for the current task and state. We approximate this by evaluating sampled actions from each policy.

We parameterize the multitask policy as $T$ policies $\{\pi_1, \pi_2, \cdots, \pi_T\}$, one for each task, where each policy returns an action distribution conditioned on a state input. The goal is to maximize the average expected return of all tasks, $\frac{1}{T}\sum_{i=1}^{T} \mathbb{E}_{\tau\sim(\pi_i,\mathcal{T},\rho_i)}[\sum_t \gamma^t R_i(s_t, a_t)]$, sample efficiently; so the approach must make use of behavior sharing between policies.

## 4 QMP: Q-switch Mixture of Policies

To solve this problem, we propose to *selectively share behavior for accelerated exploration*. Two practical challenges arise from this goal.

- **When to share behaviors.** The agent must share behaviors between policies where they seem beneficial and avoid sharing behaviors that it knows are conflicting or irrelevant. To this end, we need a mechanism to determine when we should or should not share behaviors between each pair of policies.
- **How to share behaviors.** Once we determine that sharing behaviors between a pair of policies would benefit learning, we need a mechanism for effectively sharing exploration behavior. This mechanism should allow the transfer of behaviors between policies while still being robust to possible conflicting behaviors.

Our proposed framework is specifically designed to tackle these two challenges. We define a Q-switch used to determine which policy to share behaviors with at any state (*when to share*) and use this to create a task and state dependent mixture of policies used to gather training data. An effective mixture will share helpful and relevant exploratory behavior for the current task which can then be incorporated into the current task policy (*how to share*). An illustration our proposed framework is shown in Figure 2. In the following, we first discuss when we should share behaviors in Section 4.1, describe how to share behaviors in Section 4.2, and then define our method in Section 4.3.

### 4.1 Deciding When to Share Behavior

Whether or not sharing behaviors between two policies, from $\pi_j$ to $\pi_i$, is beneficial depends on the task at hand, Task $i$, the environment state $s$, and the behavior of the other policy at that state $\pi_j(s)$. For example, two policies may share only a small segment of behavior or may initially benefit from shared exploration of a common environment but eventually diverge into more task-specific behavior. Therefore, we should selectively share behaviors based on how well we expect $\pi_j$ will do for task $i$ at state $s$.

In this work, we propose to utilize learned critics (*i.e.* , learned Q function) of $i$, $Q_i(s, a)$, which estimates the expected discounted return of the policy after taking action $a$ at state $s$. We use $Q_i$ to score each policy $\pi_j$ evaluating a sampled action from that policy $Q_i(s, a_j \sim \pi_j(s))$. This can be viewed as an unbiased but high variance estimation of the expected Q value of the action distribution $\pi_j(s)$. Then, for each task, we can decide whether to share behaviors from any other policy $\pi_j$ based on this score.

### 4.2 Effective Mechanism for Sharing Exploratory Behavior

To share exploratory behavior, we propose to use other policies directly to gather training data for the current task. Then, the behaviors with high task reward will be incorporated into the task policy through the training process. On the other hand, harmful or irrelevant behaviors will be discarded and therefore will not worsen the performance of the task policy. To allow for state-dependent behavior sharing, we use a mixture of all task policies to gather training data for each task. That is, for each task $i$, we define a mixture policy $\pi_i^{mix}$ over all the task policies $\pi_j$ with some gating function which we will explain in Section 4.3.

This allows us to activate multiple policies at different points of each episode, therefore allowing us to introduce selective behavior sharing later on. Importantly, this mixture of policies is used only to gather training data, and is not trained directly as a mixture; instead, each task policy $\pi_i$ is trained with data gathered for the current task by the mixture of policies. Thus each $\pi_i$ is trained to maximize the expected returns for a particular task $i$. The only difference to conventional RL is that the training dataset is generated by $\pi_i^{mix}$ and not $\pi_i$ directly, thus benefiting from mutual exploration without being limited by tasks with conflicting behaviors. Even so, gathering training data with harmful or irrelevant behaviors can slow down learning by wasting valuable environment steps. Thus, the choice of the gating function is critical to sample efficient learning.

### 4.3 Putting it all together: Q-switch

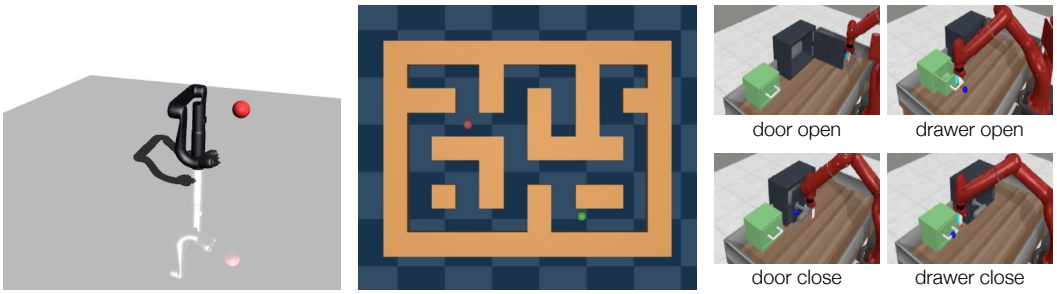
We now use our Q function metric to define a gating function. For simplicity we define the gating function Q-switch $\text{QS}_i(s, \pi_j)$ as the indicator function over the index of the best scored policy.

$$\pi_i^{mix}(s) = \sum_j^T \text{QS}_i(s, \pi_j) \cdot \pi_j(s)$$

$$\text{QS}_i(s, \pi_j) = \mathbb{1}\{j = \arg\max_k Q_i(s, a_k \sim \pi_k(s))\}$$

In other words, the mixture picks the policy with the best Q function score. The Q-switch is state and task dependent, uses the current task's Q function to predict which behaviors are most helpful, and is quickly adaptive to changes in other policies as the policies are constantly being evaluated. While there are more sophisticated ways of scoring the policies or defining the mixture probabilities, we found that this simple method works well in practice while requiring minimal hyperparameter tuning or computational overhead.

In summary, we aim to propose a method that allows for selectively sharing behavior to accelerate exploration. To this end, our method consists of a behavior sharing mechanism that uses a mixture of all task policies to gather training data for each task along with a Q-switch (how to share), a gating function based on the Q function of the current task (when to share). The Q-switch mixture of policies, $\pi_i^{mix}$ enables task and state dependent behavior sharing guided by $Q_i$, while the task specific training of $\pi_i$ allows it to benefit from multitask exploration without being limited by conflicting behaviors in other policies. Any helpful behaviors seen in exploration will be incorporated into $\pi_i$ through optimization while harmful or irrelevant behaviors, which represent error in $Q_i$, will be used to update and correct $Q_i$ and thus the Q-switch. In addition, while at the beginning of training we expect a uniform mixture over all policies on average, as $\pi_i$ is trained on task $i$ and becomes more proficient, the Q-switch will develop a stronger preference for $\pi_i$, therefore naturally sharing behavior less as $\pi_i$ becomes more specialized and requires less exploration.




(a) Multistage Reacher (b) Maze Navigation (c) Meta-World Manipulation

Figure 3: **Environments & Tasks**: (a) Multistage Reacher. The first sub-goal location is in red. (b) Maze Navigation. The agent (green spot) navigates through the maze to reach the goal (red spot). (c) Meta-World Manipulation. Consisting of tasks door open, door close, drawer open, drawer close.

## 5 EXPERIMENTS

We design environments and tasks to evaluate our proposed method, detailed in Section 5.1. The description of baselines and the variants of our proposed methods can be found in Section 5.2. Finally, in Section 5.3, we present and analyze the experimental results to answer a series of research questions as well as discuss the limitations of our proposed method.

### 5.1 ENVIRONMENTS

To evaluate our proposed method, we propose multi-task designs in locomotion and manipulation environments visualized in Figure 3. We define a set of tasks in each environment using the same agent and a consistent state space. Furthermore, each task set includes tasks with either conflicting or irrelevant behavior to test our hypothesis that our method can be robust to these cases.

**Multistage Reacher** This environment contains 5 tasks where the agent must learn to control a 6 DoF Jaco arm to reach multiple goals. In 4 out of the 5 tasks, there are 3 stages where the agent must reach different goals to progress on to the next stage. These goals are fixed for every task and there are some coinciding segments between tasks where the agent has an opportunity to share behavior. In the 5th task the agent's goal is to stay at its initial position for the goal episode. The observation space includes the Jaco arm state and the current stage number, but not the goal location, so the 5th task directly conflicts the other tasks. All tasks receive reward inversely proportional to the distance to the goal, except Task 3 which receives sparse reward when reaching a goal. The environment is simulated in the MuJoCo physics engine (Todorov et al., 2012).

**Maze Navigation** In this 2D environment, the point mass agent has to control its velocity to navigate through the maze and reach the goal where both start and goal locations are fixed. The observation consists of the agent's current position and velocity, both continuous, but lacks the goal location which should be inferred from the dense reward based on the distance to the goal. Based on the environment proposed in Fu et al. (2020), we defined 10 tasks with different start and goal locations. The optimal paths for different tasks coincide at some parts, either in the same direction or opposite direction, so the agents are required to share behaviors only for the common subtasks while avoiding conflicts.

**Meta-World Manipulation** We use a subset of 4 tasks from the the Meta-World multitask task set (Yu et al., 2019) and a modified environment proposed in Yu et al. (2021). This environment places the door and drawer objects next to each other on the table top so that all 4 tasks (door open, door close, drawer open, drawer close) are solvable in the same environment. The observation space consists of the robot proprioceptive state, the drawer handle state, the door handle state, and the goal location. While there are no directly conflicting behaviors, policies should learn to share behaviors when interacting with the same object while ignoring irrelevant behavior from policies that only interact with the other object.

Further details on task setup and implementation are in Appendix Section A.2.

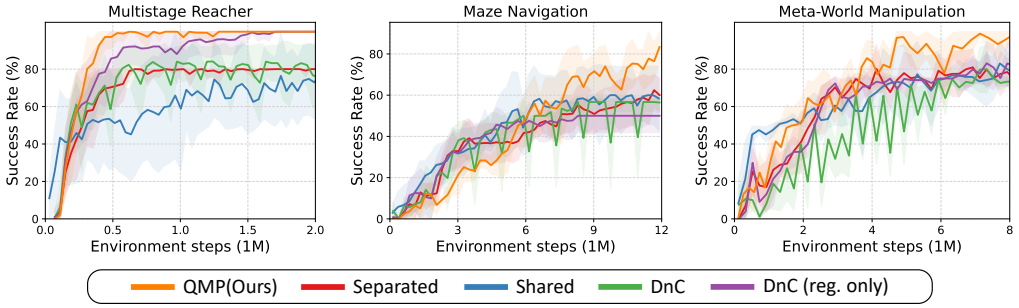



Figure 4: Comparison of average multitask success rate over training for different behavior sharing approaches. We take the average over 10 evaluation episodes per task. The plot and shaded region are mean and standard deviation over 5 seeds for each method unless otherwise specified.

### 5.2 Baselines and Ablations

We used Soft Actor-Critic (SAC) Haarnoja et al. (2018) as our basic training algorithm for all models. Also, within any model consisting of multiple agents, all agents have the same architectures for their policies and Q functions, respectively, with the same training hyperparameters including learning rates, batch sizes, and number of gradient updates. We do not include task ID's in the observation space because our problem formulation assumes a consistent state space.

- **Separated** consists of $T$ RL agents where each agent is assigned with one task and trained to solve it without any information sharing with other agents. In every training iteration, each agent collects the data for its own task and uses it for training.
- **Shared** is a single SAC agent that learns one shared policy for all tasks. For the fairness of comparison, we adjusted the size of the networks, batch size, and number of gradient updates to match those of other models with multiple agents.
- **Divide-and-Conquer RL (DnC)** proposed in Ghosh et al. (2018) uses an ensemble of policies that shares behaviors through policy distillation and regularization. We modified the method for multi-task learning by assigning each of $T$ policies to a task and evaluating only the task-specific policy.
- **DnC (Regularization Only)** is DnC without policy distillation which we found to sometimes perform better in multitask settings.
- **QMP (Ours)** learns $T$ policies with behavior sharing via Q-switch and mixture of policies.
- **QMP-Uniform** replaces the Q-switch in our method with a uniform distribution over policies to verify the importance of selective and adaptive behavior sharing.
- **QMP-Domain** replaces the Q-switch with a hand crafted, fixed policy distribution based on domain knowledge of the task set (ie. common sub-goals in Multistage Reacher).

For further details on baselines and implementation, please refer to Appendix Section A.3.

### 5.3 Results

In this section, we analyze experimental results to answer the following questions: (1) How does our method compare with other forms of behavior sharing? (2) Can our method effectively identify when and which tasks to share behaviors with? (3) Is exploration sharing complementary to other forms of multitask learning?

#### 5.3.1 How exploration sharing compares to other forms of behavior sharing?

To verify QMP as an effective and sample efficient form of behavior sharing, we compare against several behavior sharing baselines on 3 environments, Multistage Reacher (5 tasks), Maze Navigation (10 tasks), and (Meta-World Manipulation (4 Tasks) in Figure 4. Overall, QMP outperforms other methods in terms of sample efficiency and final performance.

In the MultistageReacher environment, our method reaches 100% success rate at 0.5 million environment steps, while DnC (reg.), the next best method, takes 3 times the number of steps to fully converge. The rest of the methods never reach full success rate, which the Shared baseline performing the worse, verifying the challenge of the conflicting tasks in this environment.

In the Maze Navigation environment, we test the scalability of our method to the larger 10 task set. Our method successfully solves 8 tasks on average out of 10, while other methods plateau at around a 60% success rate.

In the Meta-World Manipulation environment, our method reaches over 90% success rate after 8 million environment steps while other methods plateau at around 80%. This is significant because this task set contains a majority of irrelevant behavior: between policies interacting with the door versus the drawer, and between pulling on the object handles versus pushing. The fact that QMP can still outperform other methods validates our intuition that shared behaviors can be helpful for exploration even if the optimal behavior is different. We also note that the Shared baseline performs very well in the beginning but quickly plateaus. The initial performance is likely due to the shared policy also benefiting from shared internal representation across, whereas the other methods learn separate policies for each task.

#### 5.3.2 Can our method effectively identify when and which tasks to share behaviors with?

We analyze the effectiveness of the Q-switch in identifying when to share behaviors in two ways. First, we visualize the average proportion that each policy is selected for each task across training for the Reacher Multistage environment in Figure 5. This is a task level statistic to analyze whether QMP is sharing behaviors between similar tasks and avoiding behavior sharing between conflicting or irrelevant tasks. As we expect, Task 4 utilizes the least behavior from other policies (bottom row) and policy 4 shares the least with other tasks (rightmost column), because Task 4, to stay at initialization, conflicts with all the other goal reaching tasks. Or the remaining tasks, Task 0 and 1 share the most similar goal sequence so it is intuitive that they should also benefit from shared exploration. Finally Task 3, unlike the other tasks, receives only sparse reward and therefore relies heavily from shared exploration. In fact QMP demonstrates the greatest advantage in this task (Figure 7).

Next, we look at the importance of an adaptive, state-dependent Q-switch by comparing our policy to two ablations where the gating function is replaced by a fixed sampling distribution over the policies in Figure 5. QMP-Uniform, which samples a policy uniformly, reaches around 60% success rate, lower than the worst performing behavior sharing baseline, demonstrating that a poor choice of Q-switch can significantly hinder learning in our framework. To define the mixture probabilities for QMP-Domain, we weighted the probability of selecting $\pi_j$ for Task $i$ by the number of shared sub-goal sequences between Task $i$ and $j$ (more details in Section A.2). QMP-Domain performs well initially but plateaus early. This shows the importance of task-dependent behavior sharing and well as a small performance gap due to state-dependent sharing. And while we were able to define this domain specific mixture for the Multistage Reacher task due to the restricted behavior range, it can be difficult to define for more complex tasks and even more difficult to define a state-dependent mixture.

#### 5.3.3 Is exploration sharing complementary to other forms of multitask reinforcement learning?

It is important that our method is compatible with other forms of multitask reinforcement learning that share different kinds of information, especially parameter sharing which is very effective at the beginning of training as we saw in the Meta-World Manipulation tasks. While we use completely separated policies for our main experiments, our method is not restricted to this architecture as long as the we are able to define $T$ task-specific policies. One direct way to do this that is also commonly used in multi-task learning is to parameterize the policy with a shared multihead network architecture, where each network head outputs the action distribution for a different task. We can easily run QMP with a shared multihead network architecture by using SAC on a multihead network and replacing the data collection policy with $\pi_i^{mix}$.

We compare the following methods on the Maze Navigation environment in Figure 6.

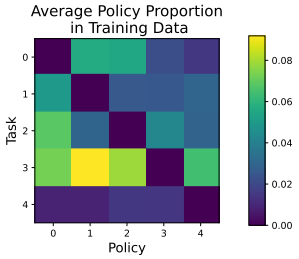



Figure 5: **Left**: Average proportion of training data for each task where each policy was selected on the Reacher Multistage task set. Each row $i$ represents the average mixture of policies used for Task $i$, with the task specific policies zeroed out to provide better contrast. **Right**: Our method enables behavior sharing that is responsive to states, non-uniform over other tasks, and adapts to the training progress of the agent.

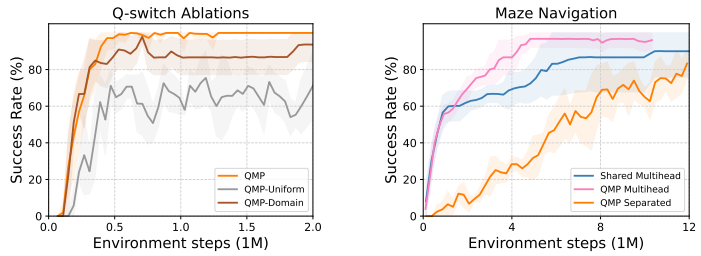



Figure 6: Combining our proposed method with multi-head shared architectures (in pink) outperforms both components on their own: multihead policy and our method without shared parameters.

- **Shared Multihead**: a multihead network trained with SAC
- **QMP Multihead**: a multihead network trained with SAC with QMP data collection
- **QMP Separated**: our original method with separate networks

We find that QMP Multihead greatly improves the performance over both the Shared Multihead baseline *and* QMP Separated, demonstrating the additive effect of these two forms of information sharing in MTRL. QMP Multihead is able to benefit from the initial sample efficient learning from the multihead architecture while also benefiting from the accelerated exploration of the mixture policies to keep learning even after the Shared Multihead baseline plateaus. This result demonstrates the compatibility and synergy between QMP and parameter sharing as key ingredients to sample efficient MTRL.

#### 5.3.4 Limitations

We empirically found that in some cases it is possible that QMP fails to learn a task because $\pi_i^{mix}$ *overshares* behaviors and fails to do enough task-guided exploration and instead the mixture looks almost uniform. This can happen if the task has sparse reward and conflicting behaviors or if there is a very large number of tasks. In either case, the Q function and $\pi_i^{mix}$ can get stuck in a local minimum where the Q-switch has little preference over policies and the resulting $\pi_i^{mix}$ is unable to get better data to update Q-switch. A possible solution is to develop a more sophisticated Q-switch that has a preference for the task-specific policy when the Q-function is overly noisy or uncertain.

## 6 Discussion

We propose Q-switch Mixture of policies (QMP), an approach for sample efficient MTRL that selectively shares exploratory behavior between tasks through a Q-switch-gated mixture of policies. Experimental results on manipulation and navigation tasks demonstrate that our proposed method effectively learns to share behavior between tasks for accelerated exploration and highlights the importance of *selective* behavior sharing. We further show that our method is complementary with parameter sharing, a popular MTRL strategy, demonstrating the effectiveness of behavior sharing in conjunction with other forms of MTRL.

Despite these encouraging results, we have also empirically identified the oversharing behavior of our proposed method, which can happen when learning from conflicting tasks with sparse rewards or when there is a very large number of tasks. Future work can address this limitation of our work through a more sophisticated mixture of policies that accounts for noise and uncertainty in the Q-function. Other promising directions include extending the idea of behavior sharing between policies and our proposed method to transfer learning or continual learning in reinforcement learning.

## REFERENCES

Sai Praveen Bangaru, JS Suhas, and Balaraman Ravindran. Exploration for multi-task reinforcement learning with deep generative models. *arXiv preprint arXiv:1611.09894*, 2016.

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

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, Jan Peters, et al. Sharing knowledge in multi-task deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *IEEE International Conference on Robotics and Automation*. IEEE, 2017.

Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. In *Advances in Neural Information Processing Systems*, 2021.

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

The garage contributors. Garage: A toolkit for reproducible reinforcement learning research. `https://github.com/rlworkgroup/garage`, 2019.

Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. In *International Conference on Learning Representations*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

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*, 2019.

Dmitry Kalashnikov, Jake Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Scaling up multi-task robotic reinforcement learning. In *Conference on Robot Learning*, 2021.

Shikun Liu, Stephen James, Andrew J Davison, and Edward Johns. Auto-lambda: Disentangling dynamic task relationships. *Transactions on Machine Learning Research*, 2022.

Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. GTI: learning to generalize across long-horizon tasks from human demonstrations. In *Robotics: Science and Systems*, 2020.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE international conference on robotics and automation*, 2018.

Taewook Nam, Shao-Hua Sun, Karl Pertsch, Sung Ju Hwang, and Joseph J. Lim. Skill-based meta-reinforcement learning. In *International Conference on Learning Representations*, 2022.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Fumihiro Sasaki and Ryota Yamashina. Behavioral cloning from noisy demonstrations. In *International Conference on Learning Representations*, 2020.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2021.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. 2012.

Zhiyuan Xu, Kun Wu, Zhengping Che, Jian Tang, and Jieping Ye. Knowledge transfer in multi-task deep reinforcement learning for continuous control. In *Advances in Neural Information Processing Systems*, 2020.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. In *Advances in Neural Information Processing Systems*, 2020.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, 2019.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, 2020.

Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Sergey Levine, and Chelsea Finn. Conservative data sharing for multi-task offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Chelsea Finn, and Sergey Levine. How to leverage unlabeled data in offline reinforcement learning. In *International Conference on Machine Learning*, 2022.

Chicheng Zhang and Zhi Wang. Provably efficient multi-task reinforcement learning with model transfer. In *Advances in Neural Information Processing Systems*, 2021.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

# A APPENDIX

## A.1 ADDITIONAL RESULTS/ANALYSIS

Additional results and analysis on Multistage Reacher are shown in Figure 7. QMP outperforms all the baselines in this task set.

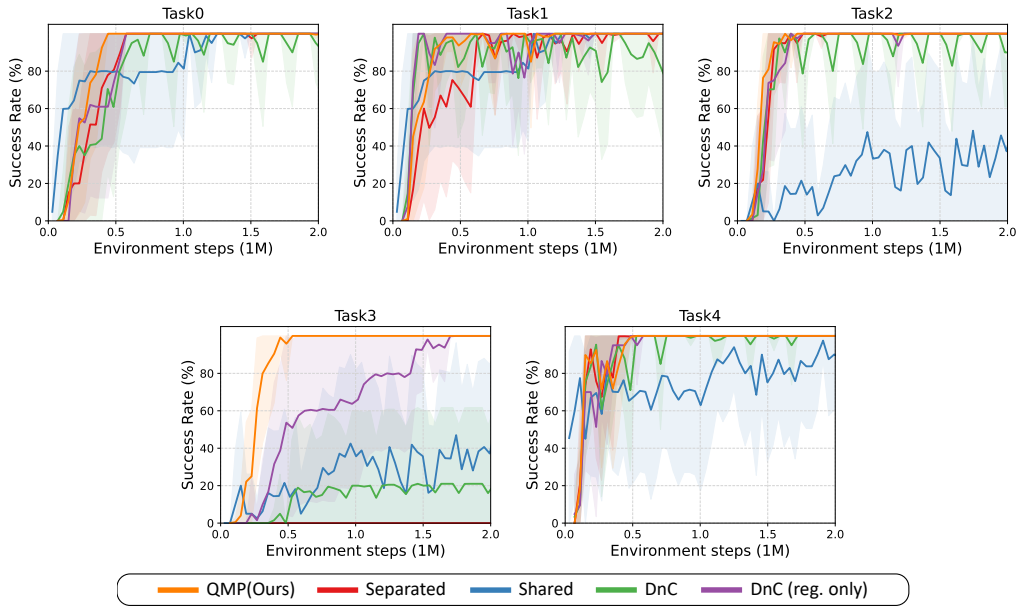



Figure 7: Success rates for individual tasks on Multistage Reacher. Our method especially helps in learning Task 3, which requires extra exploration because it only receives sparse reward.

## A.2 ENVIRONMENT DETAILS

**Multistage Reacher** We implement our multistage reacher tasks on top of the Open AI Gym (Brockman et al., 2016) Reacher environment by defining a sequence of 3 subgoals per task. The reward function is the default gym reward function based on the distance to the goal plus an additional bonus for every subgoal completed. Task 3 recieves only a sparse reward of 1 for every subgoal reached. Task 4 has one fixed goal set at its initial position.

**Maze Navigation** The layout and dynamics of the maze follow Fu et al. (2020), but since their original design aims to train a single agent reaching a fixed goal from multiple start locations, we modified it to have both start and goal locations fixed in each task, as in Nam et al. (2022). The start location is still perturbed with a small noise to avoid memorizing the task. The layout we used is `LARGE_MAZE` which is an $8\times11$ maze with paths blocked by walls. The complete set of 10 tasks are visualized in 8, where green and red spots correspond to the start and goal locations, respectively. The environment provides an agent a dense reward of $\exp(-dist)$ where $dist$ is a linear distance between the agent's current position and the goal location. It also gives a penalty of 1 at each time step, in order to prevent the agent from exploiting the reward by staying near the goal. The episode terminates as soon as the goal is reached, by having $dist < 0.5$, or 600 time steps are passed.

**Meta-World Manipulation** We reproduce the Meta-world environment proposed by Yu et al. (2021) using the Meta-world codebase (Yu et al., 2019). We additionally remove the previous state from the observation space so the policies cannot easily infer the current task. We use this modified environment instead of the Meta-world benchmark because our method assumes a consistent environment across tasks, thus the door and drawer should both be on the tabletop for all tasks.

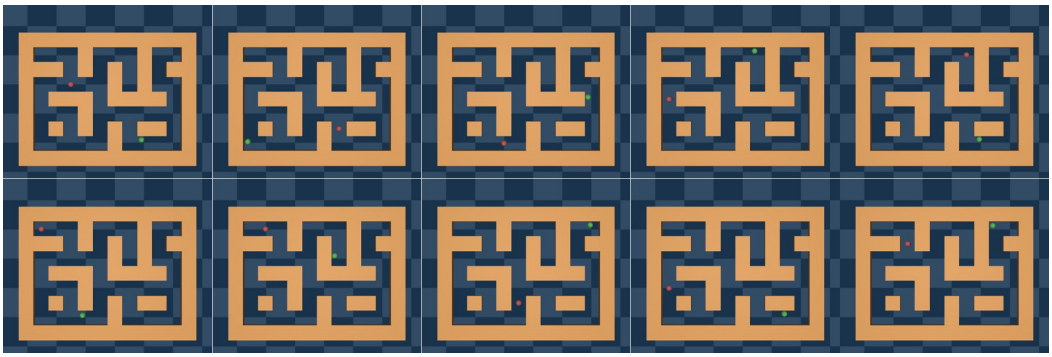


Figure 8: Ten tasks defined for the Maze Navigation. The start and goal locations in each task are shown in green and red spots, respectively. Best viewed in color.

### A.3 Implementation Details

The SAC we used in our experiments is based on the open-source implementation from Garage (garage contributors, 2019). We used fully connected layers for the policies and Q-functions with the default hyperparameters listed in Table 1. For DnC baselines, we reproduced the method in Garage to the best of our ability with minimal modifications.

#### A.3.1 Hyperparameters

Table 1 details the list of important hyperparameters on all the 3 environments.

Table 1: QMP hyperparameters.

| Hyperparameter | Multistage Reacher | Maze Navigation | Meta-World Manipulation |
|---|---|---|---|
| # Layers in $\pi$ and $Q$ | 2 | 2 | 2 |
| Activation function | tanh | tanh | tanh |
| Hidden dimension | 256 | 256 | 256 |
| Minimum buffer size (per task) | 10000 | 3000 | 10000 |
| # Environment steps per update (per task) | 1000 | 600 | 500 |
| # Gradient steps per update (per task) | 100 | 100 | 50 |
| Batch size | 32 | 256 | 256 |
| Learning rates for $\pi$, $Q$ and $\alpha$ | 0.0003 | 0.0003 | 0.0015 |
| Target update frequency | 1 | 1 | 1 |
| Target update tau ($\tau$) | 0.995 | 0.995 | 0.995 |
| Discount factor ($\gamma$) | 0.99 | 0.99 | 0.99 |

#### A.3.2 Separated

All $T$ networks have the same architecture with the hyperparameters presented in Table 1.

#### A.3.3 Shared

Since it is the only model with a single policy, we increased the number of parameters in the network to match others and tuned the learning rate. The hidden dimension of each layer is 600 in Multistage Reacher, 834 in Maze Navigation, and 512 in Meta-World Manipulation, and we kept the number of layers at 2. The number of environment steps as well as the number of gradient steps per update were increased by $T$ times so that the total number of steps could match those in other models. For the learning rate, we tried 4 different values (0.0003, 0.0005, 0.001, 0.0015) and chose the most performant one. The actual learning rate used for each experiment is 0.0003 in Multistage Reacher and Maze Navigation, and 0.001 in Meta-World Manipulation.

This modification also applies to the Shared Multihead baseline, but with separate tuning for the network size and learning rates. In Multistage Reacher, we used layers with hidden dimensions of 512 and 0.001 as the final learning rate. In Maze Navigation, we used 834 for hidden dimensions and 0.0003 for the learning rate.

#### A.3.4 DNC

We used the same hyperparameters as in Separated, while the policy distillation parameters and the regularization coefficients were manually tuned. Following the settings in the original DnC (Ghosh et al., 2018), we adjusted the period of policy distillation to have 10 distillations over the course of training. The number of distillation epochs was set to 500 to ensure that the distillation is completed. The regularization coefficients were searched among 5 values (0.0001, 0.001, 0.01, 0.1, 1), and we chose the best one. Note that this search was done separately for DnC and DnC with regularization only. For DnC, the coefficients we used are: 0.001 in Multistage Reacher and Maze Navigation, and 0.001 in Meta-World Manipulation. For Dnc with regularization only, the values are: 0.001 in Multistage Reacher, 0.0001 in Maze Navigation, and 0.001 in Meta-World Manipulation.

#### A.3.5 QMP

Our method also uses the default hyperparameters. It requires one additional hyperparameter to decide when to start using the mixture of policies in exploration. Before that, each agent will collect the data using its own policy as an exploration policy. This 'mixture warmup period' was searched over 3 values (0, 50, or 100 iterations), and the best option was chosen. In all environments, we found the period of 0 works the best.

As in Shared Multihead, the QMP Multihead also required a separate tuning. Since QMP Multihead has effectively one network, we increased the network size in accordance with Shared Multihead, and tuned the learning rate in addition to the mixture warmup period. The best performing combinations of these parameters we found are 0 and 0.001 in Multistage Reacher, and 100 and 0.0003 in Maze Navigation.