

Appendix

Result videos: <https://sites.google.com/view/qmp-mtrl>

Table of Contents

List of Tables	13
List of Figures	13
A Code Submission	14
B Environment Details	14
B.1 Multistage Reacher	14
B.2 Maze Navigation	15
B.3 Meta-World Manipulation	15
C Additional Results and Analysis	16
C.1 Multistage Reacher Per Task Results	16
C.2 QMP Behavior Sharing Analysis	16
C.3 Qualitative Visualization of Behavior-Sharing	17
C.4 Results on Non-Conflicting Fully-Shareable Task Set	18
D Implementation Details	18
D.1 Hyperparameters	19
D.2 No-Shared-Behaviors	19
D.3 Fully-Shared-Behaviors	19
D.4 DnC	19
D.5 QMP (Ours)	19
D.6 Online UDS	20

List of Tables

1	Subgoal Positions for Each Task in Multistage Reacher	14
2	QMP Hyperparameters	19

List of Figures

12	Ten Tasks Defined for the Maze Navigation	15
8	Success Rates for Individual Tasks in Multistage Reacher	16
9	Mixture Probabilities Per Task	17
10	Visualized QMP rollouts during training	17
11	Average Return in Walker2d	18

A Code Submission

In the supplementary submission, we provide the complete code to reproduce all the experiments, including QMP (ours) and baselines on all the environments.

B Environment Details

B.1 Multistage Reacher

We implement our multistage reacher tasks on top of the Open AI Gym (Brockman et al., 2016a) Reacher environment by defining a sequence of 3 subgoals per task which are specified in Table 1. For all tasks, the reacher is initialized at the same start position with a small random perturbation sampled uniformly from $[-0.01, 0.01]$ for each coordinate. The observation includes the agent’s proprioceptive state and how many sub-goals have been reached but not subgoal locations, as they must be inferred from the respective task’s reward function.

We set up the tasks to ensure that we can evaluate behavior sharing when the task rewards are qualitatively different (see Figure 3a):

- For every task except Task 3, the reward function is the default gym reward function based on the distance to the goal, plus an additional bonus for every subgoal completed.
- For Task 1, the reward is shifted by -2 at every timestep.
- Task 3 receives only a sparse reward of 1 for every subgoal reached.
- Task 4 has one fixed goal set at its initial position.

	Subgoal 1	Subgoal 2	Subgoal 3
T_0	(0.2, 0.3, 0.5)	(0.3, 0, 0.3)	(0.4, -0.3, 0.4)
T_1	(0.2, 0.3, 0.5)	(0.3, 0, 0.3)	(0.4, 0.3, 0.2)
T_2	(0.3, 0, 0.3)	(0.4, 0.3, 0.2)	(0.4, -0.3, 0.4)
T_3	(0.3, 0, 0.3)	(0.4, -0.3, 0.4)	(0.2, 0.3, 0.5)
T_4	initial	initial	initial

Table 1: Coordinates of subgoal locations for each task in Multistage Reacher. Shared subgoals are highlighted in the same color. For Task 4, the only goal is to stay in the initial position.

QMP-Domain: Section 6.2 ablates the importance of an adaptive and state-dependent Q-switch by replacing it with a domain-dependent distribution over other tasks based on apparent task similarity. Specifically, to define the mixture probabilities for QMP-Domain in Multistage Reacher, we use the domain knowledge of the subgoal locations of the tasks to determine the mixture probabilities. We use the ratio of *shared sub-goal sequences* between each pair of tasks (not necessarily the shared subgoals) over the total number of sub-goal sequences, 3, to calculate how much behavior must be shared between two tasks. For that ratio of shared behavior, we distribute the probability mass uniformly between all task policies that share that behavior. For Task 4, the conflicting task, we do not do any behavior sharing and only use π_4 to gather data.

Each Task T_i consists of 3 sub-goal sequences $\{S_0, S_1, S_2\}$ (e.g. [initial \rightarrow Subgoal 1], [Subgoal 1 \rightarrow Subgoal 2], and [Subgoal 2 \rightarrow Subgoal 3]). For each sequence $s \in \{S_0, S_1, S_2\}$, we divide equally the contribution of each task T_j ’s policy π_j that shares the sequence s (i.e. if T_0 and T_1 both contain sequence s , where we use the notation $\mathbb{1}(s \in T_i)$ as the indicator function for whether Task T_i contains sequence s , then π_0 and π_1 both have $\frac{1}{2}$ contribution for s). Each sequence contributes equally to the overall mixture probabilities for Task i (i.e. all policies that shares sequence S_i contributes in total $\frac{1}{3}$ to the mixture probability for the behavior policy of Task T_i). Thus, the contribution probability of Policy π_j to Task T_i is:

$$p_{j \rightarrow i} = \sum_{s \in \{S_0, S_1, S_2\}} \frac{1}{3} \cdot \frac{\mathbb{1}(s \in T_j)}{\sum_k \mathbb{1}(s \in T_k)}$$

$$\pi_i^{mix} = \sum_j p_{j \rightarrow i} \pi_j$$

Reusing notation for mixture probabilities, we have,

$$\begin{aligned}\pi_0^{mix} &= \frac{2}{3}\pi_0 + \frac{1}{3}\pi_1 \\ \pi_1^{mix} &= \frac{1}{3}\pi_0 + \frac{2}{3}\pi_1 \\ \pi_2^{mix} &= \frac{5}{6}\pi_2 + \frac{1}{6}\pi_3 \\ \pi_3^{mix} &= \frac{1}{6}\pi_2 + \frac{5}{6}\pi_3 \\ \pi_4^{mix} &= \pi_4\end{aligned}$$

B.2 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 to reach 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 observation consists of the agent’s current position and velocity. But, it lacks the goal location, which should be inferred from the dense reward based on the distance to the goal. The layout we used is LARGE_MAZE which is an 8×11 maze with paths blocked by walls. The complete set of 10 tasks is visualized in Figure 12 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 timestep in order to prevent the agent from exploiting the reward by staying near the goal. The episode terminates either as soon as the goal is reached by having $dist < 0.5$ or when 600 timesteps have passed.



Figure 12: Ten tasks defined for the Maze Navigation. The start and goal locations in each task are shown in green and red spots, respectively, and an example path is shown in green.

B.3 Meta-World Manipulation

We reproduce the Meta-world environment proposed by [Yu et al. \(2021\)](#) using the Meta-world codebase ([Yu et al. 2019](#)), where the door and drawer are both placed side-by-side on the tabletop for all tasks (see Figure 3c). The observation space consists of the robot’s proprioceptive state, the drawer handle state, the door handle state, and the goal location, which varies based on the task. Unlike [Yu et al. \(2021\)](#), we additionally remove the previous state from the observation space so the policies cannot easily infer the current task, making it a challenging multi-task environment. The environment also uses the default Meta-World reward functions which is composed of two distance-based rewards: distance between the agent end effector and the object, and distance between the object and its goal location. We use this modified environment instead of the Meta-world benchmark because our problem formulation of simultaneous multi-task RL requires a consistent environment across tasks.

C Additional Results and Analysis

C.1 Multistage Reacher Per Task Results

Additional results and analysis on Multistage Reacher are shown in Figure 8. QMP outperforms all the baselines in this task set, as shown in Figure 4. Task 3 receives only a sparse reward and, thus, can benefit the most from shared exploration. We observe that QMP gains the most performance boost due to selective behavior-sharing in Task 3. The No-Shared-Behavior baseline is unable to solve Task 3 at all due to its sparse reward nature. The other baselines which share uniformly suffer at Task 3, likely because they also share behaviors from other conflicting tasks, especially Task 4. We explore this further in the following Section C.2.

For all tasks, QMP outperforms or is comparable to No-Shared-Behavior, which shows that selective behavior-sharing can help accelerate learning when task behaviors are shareable and is robust when tasks conflict. Fully-Shared-Behavior especially underperforms in Tasks 2 and 3, which require conflicting behaviors upon reaching Subgoal 1, as defined in Table 1. In contrast, it excels at the beginning of Task 0 and Task 1 as their required behaviors are completely shared. However, it suffers after Subgoal 2, as the task objectives diverge.

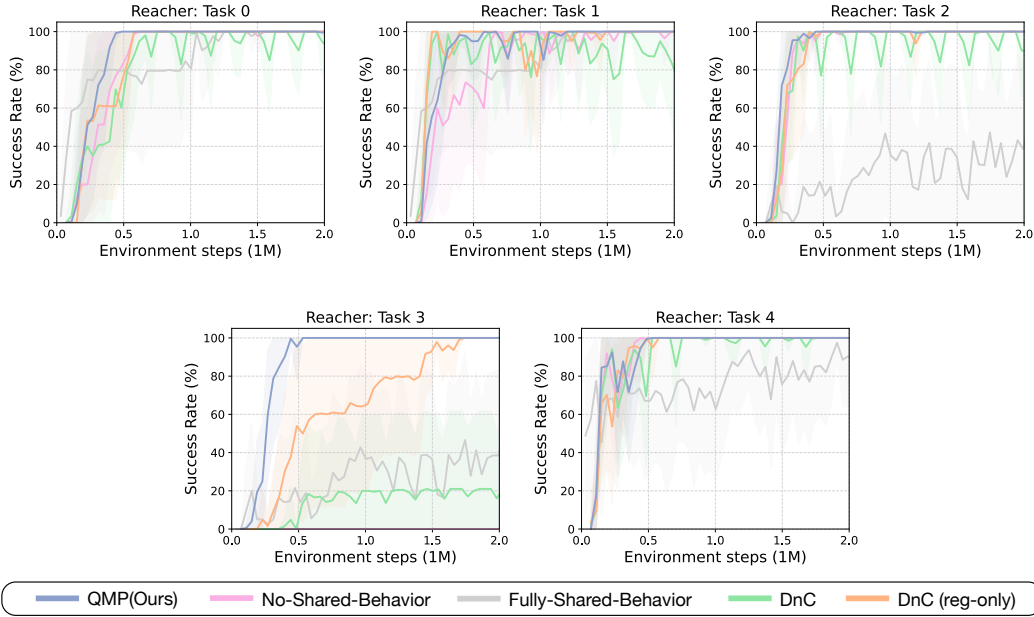


Figure 8: Success rates for individual tasks in Multistage Reacher. Our method especially helps in learning Task 3, which requires extra exploration because it only receives a sparse reward.

C.2 QMP Behavior Sharing Analysis

QMP learns to not share from conflicting tasks: We visualize the mixture probabilities per task of other policies in Figure 9 for Multistage Reacher, highlighting the conflicting Task 4 in red. Throughout the training, we see that QMP learns to generally share less behavior from Policy 4 than other policies in Tasks 0-3 and shares the least total cross-task behavior in Task 4. This supports our claim that the Q-switch can identify conflicting behaviors that should not be shared. We also note that Task 3 has a relatively larger amount of sharing than other tasks. The sparse reward nature of Task 3 makes it benefit the most from exploration via selective behavior-sharing from other tasks.

Behavior-sharing reduces over training: Figure 9 shows that the total amount of behavior-sharing decreases over the course of training in all tasks, which demonstrates a naturally arising preference in the Q-switch for the task-specific policy as it becomes more proficient at its own task.

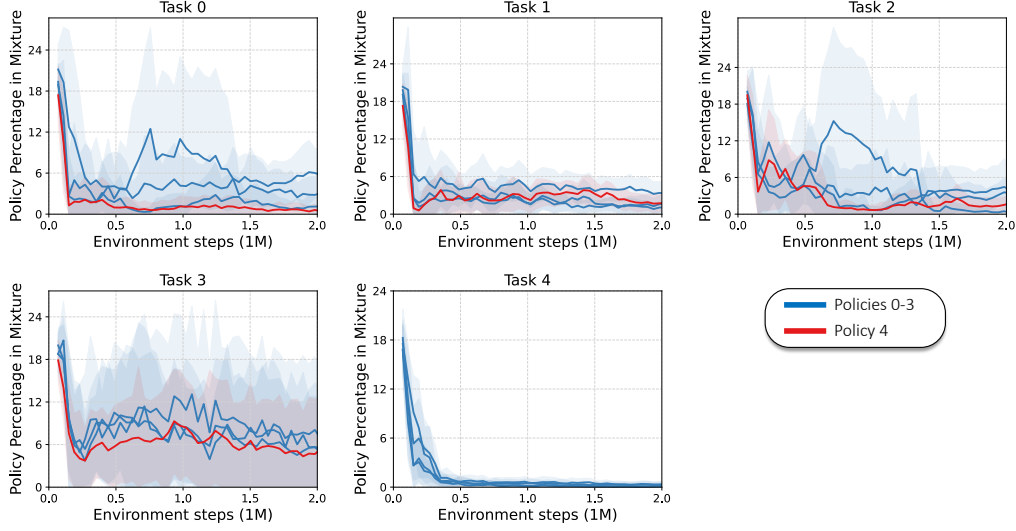


Figure 9: Mixture probabilities per task of other policies over the course of training for Multistage Reacher. The conflicting task Policy 4, which requires staying stationary, is highlighted in red.

622 C.3 Qualitative Visualization of Behavior-Sharing

623 We qualitatively analyze behavior sharing by visualizing a rollout of QMP during training for the
 624 Drawer Open task in Meta-World Manipulation (Figure 10). We see that it (i) switches between all 4
 625 task policies as it approaches the drawer (as they all bring the robot end effector physically closer to
 626 the drawer), (ii) uses drawer-specific policies as it grasps the drawer-handle, and (iii) uses Drawer
 627 Open and Door Open policies as it pulls the drawer open. This suggests that in addition to ignoring
 628 conflicting behaviors, QMP is able to identify helpful behaviors to share. We note that QMP is not
 629 perfect at policy selection throughout the entire rollout, and it is also hard to interpret these shared
 630 behaviors exactly because the policies themselves are only partially trained, as this rollout is from the
 631 middle of training. However, in conjunction with the overall results and analysis, this supports our
 632 claim that QMP can effectively identify shareable behaviors between tasks.

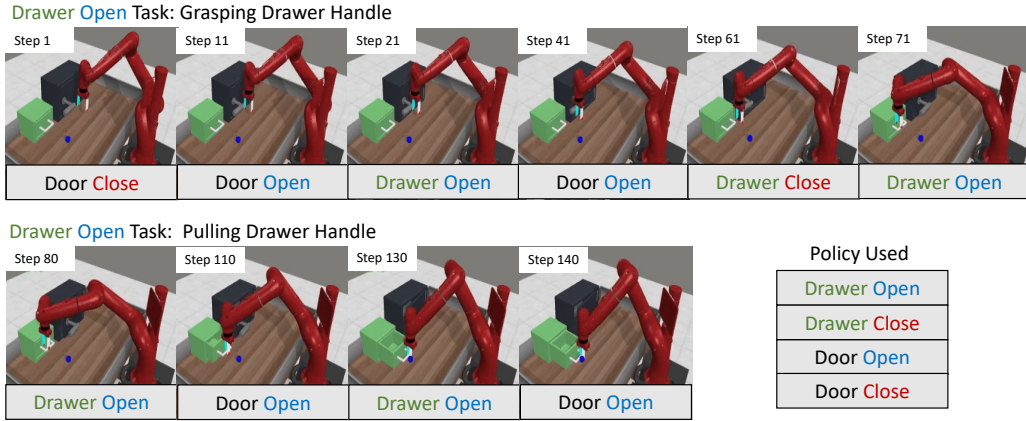


Figure 10: We visualize a QMP rollout during training (before the policy fully learns the task) for the Drawer Open task in Meta-World Manipulation by labeling each transition to a different task policy. For clarity, we first subsample the episode timesteps by 10. We qualitatively split the episode into three subtasks: (i) reaching the drawer (top row; Steps 1-60), (ii) grasping the drawer handle (top row; Steps 61-80), and (iii) pulling the drawer handle (bottom row). QMP uses all policies to approach the handle and then only the drawer-specific policies to grasp the handle as the agent nears the handle. To learn to pull the drawer open, QMP uses only Drawer Open and Door Open policies.

633 C.4 Results on Non-Conflicting Fully-Shareable Task Set

634 Walker2D is a 9 DoF bipedal walker agent with the multi-task set containing 4 locomotion tasks
 635 proposed in [Lee et al. \(2019\)](#): walking forward, walking backward, balancing, and crawling under
 636 a ceiling. Each of these tasks involves different gaits or body positions to accomplish successfully
 637 without any obviously identifiable shared behavior in the optimal policies. Behavior sharing can
 638 still be effective during training to aid exploration and share helpful intermediate behaviors, like
 639 balancing. However, there is no obviously identifiable conflicting behavior either in this task set.
 640 Because each task requires a different gait, it is unlikely for states to recur between tasks and even
 641 less likely for states that are shared to require conflicting behaviors. For instance, it is common for all
 642 policies to struggle and fall at the beginning of training, but all tasks would require similar stabilizing
 643 and correcting behavior over these states.

644 In this environment, we found that QMP still outperforms No-Shared-Behavior or Fully-Shared-
 645 Behavior baselines, but DnC (Reg. only) works best (see Figure [11](#)). DnC benefits from an
 646 additional tunable hyperparameter for the policy regularization coefficient, which dictates the strength
 647 of behavior sharing. In non-conflicting task sets such as this environment, DnC can fully leverage
 648 behavior-sharing with a high regularization coefficient without harming any individual task policies,
 649 which would be the case if there are any conflicting behaviors. In contrast, QMP selectively and
 650 adaptively shares behaviors through the Q-filter, so the amount of shared behavior can be more
 651 conservative in comparison to a well-tuned DnC in non-conflicting task sets. This is a trade-off for
 652 our behavior-sharing algorithm that is more robust to possibly conflicting task behaviors and has
 653 fewer hyperparameters to tune. In purely non-conflicting and fully-shareable task sets, tuning the
 654 regularization strength in DnC is likely the best method. However, in task sets with the presence of
 655 conflicting behaviors or where the similarity in task behaviors is not clear or known a priori, we find
 656 QMP to be the best option as it is a more robust form of behavior sharing compared to all baselines.

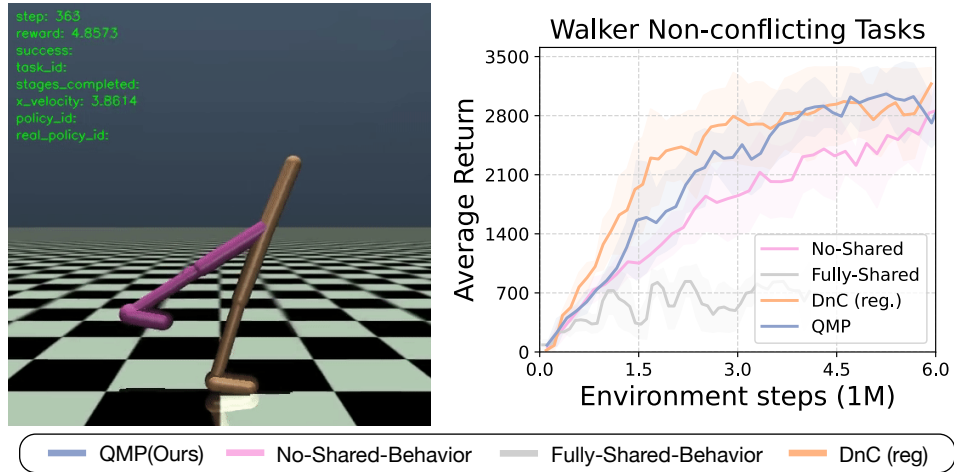


Figure 11: Average Return in Walker2d multi-task set which consists of 4 tasks: walk forward, walk backward, balance, and crawl. The return is averaged over 10 evaluation episodes per task and we report the learning curves over 3 seeds for each algorithm.

657 D Implementation Details

658 The SAC implementation we used in all our experiments is based on the open-source implementation
 659 from Garage ([garage contributors, 2019](#)). We used fully connected layers for the policies and Q-
 660 functions with the default hyperparameters listed in Table [2](#). For DnC baselines, we reproduced the
 661 method in Garage to the best of our ability with minimal modifications.

662 We used PyTorch ([Paszke et al., 2019](#)) for our implementation. We run the experiments primarily on
 663 machines with either NVIDIA GeForce RTX 2080 Ti or RTX 3090. Most experiments take around
 664 one day or less on an RTX 3090 to run. We use the Weights & Biases tool ([Biewald, 2020](#)) for
 665 logging and tracking experiments. All the environments were developed using the OpenAI Gym
 666 interface ([Brockman et al., 2016b](#)).

667 D.1 Hyperparameters

668 Table 2 details the list of important hyperparameters on all the 3 environments.

Table 2: QMP hyperparameters.

Hyperparameter	Multistage Reacher	Maze Navigation	Meta-World Manipulation
# Layers in π 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 π , Q and α	0.0003	0.0003	0.0015
Target update frequency	1	1	1
Target update tau (τ)	0.995	0.995	0.995
Discount factor (γ)	0.99	0.99	0.99

669 D.2 No-Shared-Behaviors

670 All T networks have the same architecture with the hyperparameters presented in Table 2.

671 D.3 Fully-Shared-Behaviors

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

680 This modification also applies to the Shared Multihead baseline, but with separate tuning for the
681 network size and learning rates. In Multistage Reacher, we used layers with hidden dimensions of
682 512 and 0.001 as the final learning rate. In Maze Navigation, we used 834 for hidden dimensions and
683 0.0003 for the learning rate.

684 D.4 DnC

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

694 D.5 QMP (Ours)

695 Our method also uses the default hyperparameters. We experimented with an optional ‘mixture
696 warmup period’ hyperparameter to decide when to start using the mixture of policies in exploration.
697 Before warmup, each agent collects data using its own policy as an exploration policy. We searched

over 3 values: 0, 50, or 100 iterations. We found the option of 0 warmup iterations to perform the best across all the environments.

Like in Baseline Multihead (Parameters-Only), the QMP Multihead architecture (Parameters+Behaviors) also required a separate tuning. Since QMP Multihead effectively has one network, we increased the network size in accordance with Baseline 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, respectively.

D.6 Online UDS

Yu et al. (2022) proposes an offline multi-task RL method (UDS) that shares data between tasks if their conservative Q value falls above the k^{th} percentile of the task data. Specifically, before training, you would go through all the tasks' data and share some data from Task j to Task i if the Task i Q value of that data is greater than $k\%$ of the Q values of Task i 's data. UDS does not require access to task reward functions like other data-sharing approaches. It simply re-labels any shared data with the minimum task reward, making it applicable to our problem setting as we also do not assume that reward relabeling is possible.

In order to adapt UDS to online RL, instead of doing data sharing once on the given multi-task dataset, we apply UDS data sharing before every training iteration to the data in the multi-task replay buffers. Concretely, we implement this on-the-fly for every batch of sampled data by sampling one batch of data from Task i 's replay buffer, β_i , and one batch of data from the other task's replay buffers $\beta_{j \neq i}$. Then following UDS, we would form the effective batch β_i^{eff} by sharing data from $\beta_{j \neq i}$ if it falls above the k^{th} percentile of Q values for β_i :

$$UDS_{\text{online}} : (s, a, r_i, s') \sim \beta_{j \neq i} \in \beta_i^{\text{eff}} \\ \text{if } \Delta^\pi(s, a) := \hat{Q}^\pi(s, a, i) - P_{k^{\text{th}}}[\hat{Q}^\pi(s', a', i) : s', a' \sim \beta_i] \geq 0$$

Note the differences here: (i) the 'data' used for data-sharing is the sampled replay buffer batch instead of the offline dataset, and (ii) we use the standard Q-function to evaluate data instead of the conservative Q-function since we are doing online (not offline) RL. We implement it this way as a practical approximation to avoid having to process the entire replay buffer every training iteration.

We use the same default hyperparameters as the other baseline methods. Additionally, we need to tune the sharing percentile k . For this, we tried 0th percentile (sharing all data) and 80th percentile, and chose the best-performing one.

In Figure 12, we report multiple sharing percentiles for UDS and for CDS (Yu et al., 2021) which assumes access to ground truth task reward functions which it uses to re-label the shared data. When the shared data is relabeled with task reward functions, thereby bypassing the conflicting behavior problem, online data sharing approaches can work very well. But when unsupervised, we see that online data sharing can actually harm performance in environments with conflicting tasks, with the more conservative data sharing approach (UDS $k=80$) outperforming sharing all data.

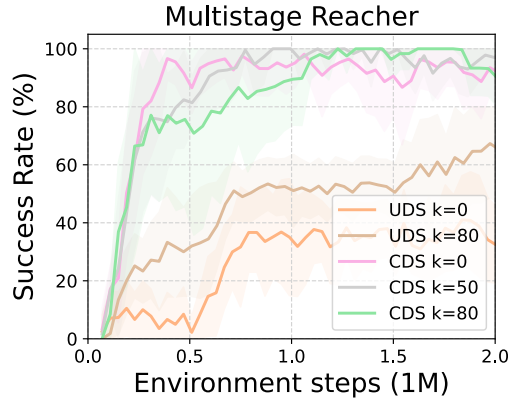


Figure 12: Various data sharing approaches on Multistage Reacher. Online data sharing is very efficient when given task reward functions (all CDS versions), but suffers without (all UDS versions).