Appendix

A M-step for tasks with distinct transition functions P_τ

Here we extend the improvement in the M-step to the more general case, where we do not require tasks to share transition dynamics $P_\tau(\cdot|s,a)$. Specifically, we aim to show how a policy π trained on multiple tasks $\tau \in \mathcal{T}$ can improve the summed performance $\sum_{\tau\in\mathcal{T}} G_\tau(\pi)$. Note that we omit the cluster identifiers here for brevity, as clusters are treated independently in the M-step and a generalization to the overall objective follows trivially. We first define for the current policy π an auxiliary MDP $\mathcal{M}(\pi)$ as $(\boldsymbol{S}, \boldsymbol{A}, P_\mathcal{M}, R_\mathcal{M}, \gamma)$ with $\boldsymbol{S}, \boldsymbol{A}$ and γ as given by the tasks $\mathcal{T}$, $R_\mathcal{M}(s,a) = \mathbb{E}_{\tau\sim\mathcal{T}}[R_\tau(s,a)]$ and

$$P_\mathcal{M}(s'|s,a) = \frac{\mathbb{E}_{\tau\sim\mathcal{T}}\left[P_\tau(s'|s,a)V_\tau^\pi(s')\right]}{\mathbb{E}_{\tau\sim\mathcal{T}}[V_\tau^\pi(s')]}$$

where $V_\tau^\pi(s')$ is given by the cumulative discounted return of policy π on task τ starting in state s'. Note that this MDP has an evaluation $V_\mathcal{M}^\pi$ of π equivalent to the expected value of π:

$$\begin{aligned}
V_\mathcal{M}^\pi(s) &= \mathbb{E}_{\tau\in\mathcal{T}}\left[V_\tau^\pi(s)\right] \\
&= \mathbb{E}_{\tau\in\mathcal{T}}\left[\mathbb{E}_{a\sim\pi}\left[R_\tau(s,a) + \mathbb{E}_{s'\sim P_\tau(\cdot|s,a)}\left[V_\tau^\pi(s')\right]\right]\right] \\
&= \mathbb{E}_{a\sim\pi}\left[\mathbb{E}_{\tau\in\mathcal{T}}\left[R_\tau(s,a)\right] + \mathbb{E}_{\tau\in\mathcal{T}}\left[\mathbb{E}_{s'\sim P_\tau(\cdot|s,a)}\left[V_\tau^\pi(s')\right]\right]\right] \\
&= \mathbb{E}_{a\sim\pi}\left[\mathbb{E}_{\tau\in\mathcal{T}}\left[R_\tau(s,a)\right] + \sum_{s'\in\boldsymbol{S}} \mathbb{E}_{\tau\in\mathcal{T}}\left[P_\tau(s'|s,a)V_\tau^\pi(s')\right]\right] \\
&= \mathbb{E}_{a\sim\pi}\left[R_\mathcal{M}(s,a) + \mathbb{E}_{s'\sim P_\mathcal{M}(\cdot|s,a)}\left[V_\mathcal{M}^\pi(s')\right]\right]
\end{aligned}$$

Therefore, a policy improvement based on $V_\mathcal{M}^\pi$ will improve the overall objective. In practice, one could re-weight sampled transitions based on an estimated importance ratio $\frac{P_\mathcal{M}(s'|s,a)}{\mathbb{E}_{\tau\in\mathcal{T}}[P_\tau(s'|s,a)]}$. However, we empirically found that our approach also works without such a re-weighting, as our approach can also simply assign tasks with distinct dynamics to different clusters.

B Experiment Details

In addition to the details provided here, the implementation of all experiments can be found in the supplementary material.

B.1 Grid World Experiments

In the first discrete task set we use a one-dimensional state-chain with 51 states, in which the agent starts in the middle and receives a reward for moving toward either the left or right end. As a reward we use $r = \frac{1}{|x_\text{ag} - x_\text{goal}|}$ where x_ag is the position of the agent and x_goal is the goal position (either the left or right end of

the chain). We give a reward of $r = 20$ if the goal position is reached. Depending on the task, the reward is given every 2, 4, 8 or 16 steps, or only at the goal position, and otherwise replaced by $r = 0$.

For our corner grid-world task set we use a 2D-grid-world with edge length 7 and three goal positions per corner (as depicted in Figure 1). The agent always starts in the center and receives a reward based on the distance to the target $r = \frac{1}{||x_{\text{ag}} - x_{\text{goal}}||_2}$, with $|| \cdot ||_2$ being the Euclidean norm. A reward of $r = 10$ is given when the agent reaches the goal position.

In both tasks we use tabular Q-Learning with ϵ-greedy exploration. We start with $\epsilon_0 = 0.2$ and decay the value as $\epsilon_t = \epsilon_0^{\gamma_\epsilon^t}$ with $\gamma_\epsilon = 1 - 1 \times 10^{-6}$. We use a learning rate of $\alpha = 0.2$ to update the value estimates, as from the perspective of a single agent the environment can be regarded as stochastic. Further, we use a discount factor of $\gamma = 0.9$ and $T_M = 500$ training steps per policy in each M-step and evaluate each policy on each task for three episodes during the E-step, using the greedy policy without exploration.

B.2 Pendulum

In our pendulum tasks we use a modified version of the `Pendulum` environment provided in OpenAI gym [3]. This environment consists of a single pendulum and the goal is to balance it in an upright position. The observation consists of the current angle θ, measured from the upright position, and current angular velocity represented as $(\sin\theta, \cos\theta, \dot{\theta})$. The reward for each time step is $r_t = -(\theta^2 + 0.1\dot{\theta}^2 + 0.001a^2)$, with a being the torque used as action. Every episode starts with a random position and velocity. To provide a set of tasks we vary the length of the pendulum in $\{0.7, 0.8, ..., 1.3\}$.

Hyperparameters Hyperparameters for our EM-TD3 and multi-head TD3 were tuned on the pendulum task set by grid search over learning rate $\alpha = \{1 \times 10^{-2}, 3 \times 10^{-3}, 1 \times 10^{-3}\}$, batch-size $b = \{64, 128\}$ and update-rate $u = \{1, 3, 5\}$, specifying the number of collected time-steps after which the value-function is updated. We increased the network size for multi-head TD3, so that it overall had more parameters than EM-TD3. This is done to eliminate a potential advantage of our approach stemming from a higher representational capacity. The tuned hyperparameters are given in Table 1. To represent the value functions and policies we use fully connected multi-layer perceptrons (MLPs) with two hidden layers with 64 units each. As activations we use ReLU on all intermediate layers, and tanh activations on the output. The values are then scaled to the torque limits per dimension. In EM, SP and PPT we use a separate network for each policy. For our multi-head baseline we share the hidden layers between tasks, but use separate input and output layers per task. Additionally, we increase the size of the first hidden layer to 96 in the multi-head approach, such that it has a similar total number of parameters as our EM approach. For SP and PPT we reuse the hyper-parameters from our EM approach. For Surgery we use a network with a separate output layer per policy and similarly increase the size

of the layers and reuse the other parameters from the Multi-Head approach, as we found them to behave similarly. During the M-step, we train the agent for 5×10^4 steps per policy and during the E-step we evaluate each agent on each task by running 20 episodes without added exploration noise.

Table 1: Hyperparamters for pendulum experiments.

Hyperparameter	EM-TD3	Multi-head TD3	Surgery
learning-rate α	3×10^{-3}	3×10^{-3}	3×10^{-3}
batch-size b	128	128	128
update-rate u	1	1	1
policy-update-frequency	3	3	3
n - EM	4	-	-
network size	$4 \cdot (64, 64, 1)$	$(9 \cdot 96, 64, 9 \cdot 1)$	$(128, 128, 1)$
exploration noise σ	0.05	0.05	0.05
exploration noise clipping	$[-0.5, 0.5]$	$[-0.5, 0.5]$	$[-0.5, 0.5]$
target policy smoothing noise σ	0.1	0.1	0.1
buffer-size	2e6 per policy	2e6 per task	2e6 per task
decay γ	0.99	0.99	0.99
T_M	5×10^4	-	-

B.3 BipedalWalker

For the BipedalWalker tasks we look at two different sets of tasks. The first set of tasks consists of different reward functions with mostly similar environments, inspired by track and field events. The tasks are jumping up, jumping a long distance, runs for different distances and a run with obstacles. In all tasks a reward of $-\epsilon||a||_1$ is given to minimize the used energy. The position of the hull of the bipedal walker is denoted as (x, y). In the jump up task a reward of $y - |x|$ is given upon landing, and $\epsilon = 3.5 \times 10^{-4}$. For the long jump task a reward of $x - x_0$ is given upon landing, with x_0 being the hull position during the last ground contact, $\epsilon = 3.5 \times 10^{-4}$. The three runs consist of a sprint over a length of 67 units, with $\epsilon = 3.5 \times 10^{-4}$, a run over 100 units, with $\epsilon = 3.5 \times 10^{-4}$, and a long run over 200 units with $\epsilon = 6.5 \times 10^{-4}$. The hurdles task is identical to the long run, but every 4 units there is an obstacle with a height of 1. Additionally, a reward of $0.1\dot{x}$ — a reward proportional to the velocity of the agent in the x-direction — is given during the run and hurdle tasks, to reward movement to the right.

The second set of tasks consists of varying obstacles and robot parameters. We vary the length of the legs in $\{25, 35, 45\}$ and either use no obstacles, or obstacles with a spacing of 2 or 4 units apart and height of 1. This results in a total of 9 tasks. Here we use the standard reward for the BipedalWalker task

$r = 4.3\dot{x} - 5|\theta| - ||a||_1$ with θ being the angle of the walker head. Additionally, in all experiments $r = -100$ is given if the robot falls over or moves to far to the left.

Hyperparameters Hyperparameters for our EM-TD3 and multi-head TD3 approaches were tuned on the track and field task set by grid search over $\alpha = \{1 \times 10^{-3}, 3 \times 10^{-4}, 1 \times 10^{-4}\}$, batch-size $b = \{100, 1000\}$ and update-rate $u = \{1, 3, 5\}$, u specifying the number of collected time-steps after which the value-function is updated. We reuse the optimal parameters found here on the task set with varying leg lengths and obstacles. For the SP and PPT baselines we reused the parameters from EM-TD3. We increased the network size for multi-head TD3, so that it overall had more parameters than EM-TD3. All hyperparameters are given in Table 2. For Surgery we reuse the parameters from the Multi-Head approach, as we found them to behave similarly. During the M-step, we train the EM agent with 2×10^5 steps per policy and during the E-step we evaluate each agent on each task by running 20 episodes without added exploration noise.

Table 2: Hyperparameters for BipedalWalker experiments.

Hyperparameter	EM-TD3	Multi-head TD3	Surgery
learning-rate	1×10^{-3}	1×10^{-3}	1×10^{-3}
batch-size	1000	1000	1000
update-rate	3	5	5
policy-update-frequency	3	3	3
n - EM	4	-	-
network size	$4 \cdot (400, 300, 1)$	$(6 \cdot 400, 400, 6 \cdot 1)$	$(800, 600, 1)$
exploration noise σ	0.1	0.1	0.1
exploration noise clipping	$[-0.5, 0.5]$	$[-0.5, 0.5]$	$[-0.5, 0.5]$
target policy smoothing noise σ	0.2	0.2	0.2
buffer-size	$5e6$ per policy	$5e6$ per task	$5e6$ per task
decay γ	0.99	0.99	0.99
T_M	2×10^5	-	-

B.4 Atari

To test our approach on a more complex task, we evaluate it on a subset of the Atari games. The set of chosen games consists of Alien, Assault, BankHeist, ChopperCommand, DemonAttack, JamesBond, MsPacman, Phoenix, RiverRaid, SpaceInvaders, WizardOfWor and Zaxxon. As stated above, this task set is similar to the set of games used in [30], but without tasks requiring a large amount of exploration to save computation time.

Our implementation is based on the IQN implementation in the Dopamine framework [8, 6]. As hyperparameters we use the default values recommended by Dopamine for Atari games, except the changes listed below: Due to the different action spaces, we use a separate replay buffer for each game, as well as a separate output layer, both for our EM, multi-head and PPT approaches. We reduce the size of the replay buffer to 3×10^5 compared to 1×10^6 in the original paper, to reduce the memory demand. We use the normal NatureDQN network, but scale the size of the layers to ensure that each approach has a similar number of parameters. For our EM approach, we use $T_M = 2.5 \times 10^5$ trainings steps per M-step, and evaluate all policies on all tasks for 27000 steps in the E-step, using the greedy policy without random exploration. In both EM and the multi-head approach, we record how many transitions were performed in each M-Step and sample the task with the least transitions as next training task. This is done to ensure a similar amount of transitions and training steps per game, as episode lengths vary. This approach was proposed in [30].

C Additional Results

C.1 Bipedal Walker

In Figure 9 the assignments for 4 randomly chosen trials on the track and field task set are shown. We can see that in all trials the runs over different distances are grouped together with the long jump task. This is likely due to these tasks aligning well, as they both favor movements to the right. It is possible to learn the hurdles task with the same policy as the runs, due to the available LIDAR inputs. The hurdle task therefore sometimes switches between policies, but usually is learned by a separate policy. The jump up task is very different from the other tasks, as it is the only one not to involve movement to the right, and is therefore assigned to a separate policy.

In Figure 10 the assignments for 4 randomly chosen trials on the leg-length and obstacle task set are shown. As illustrated by the good performance of the SP approach shown in Figure 5, it is possible to learn a nearly optimal behavior with a single policy here. This makes learning a meaningful clustering significantly harder and sometimes leads to a single policy becoming close to optimal on all tasks, as in Trial 2. In most other trials the task set is separated into two or three different clusters based on the different leg lengths.

C.2 Atari

In Figure 11 the assignments of all three trials of our approach on the Atari task set are shown. While we see a consistency in assignments, we cannot identify a clearly repeated clustering across trial. We assume this is due to the high diversity of tasks preventing the identification of clearly distinguishable clusters. This lack of clearly distinguishable clusters might also be the reason for failing to exceed the performance of PPT. Yet, the specialization of policies in our approach helps to avoid negative transfer as seen in Figure 6.

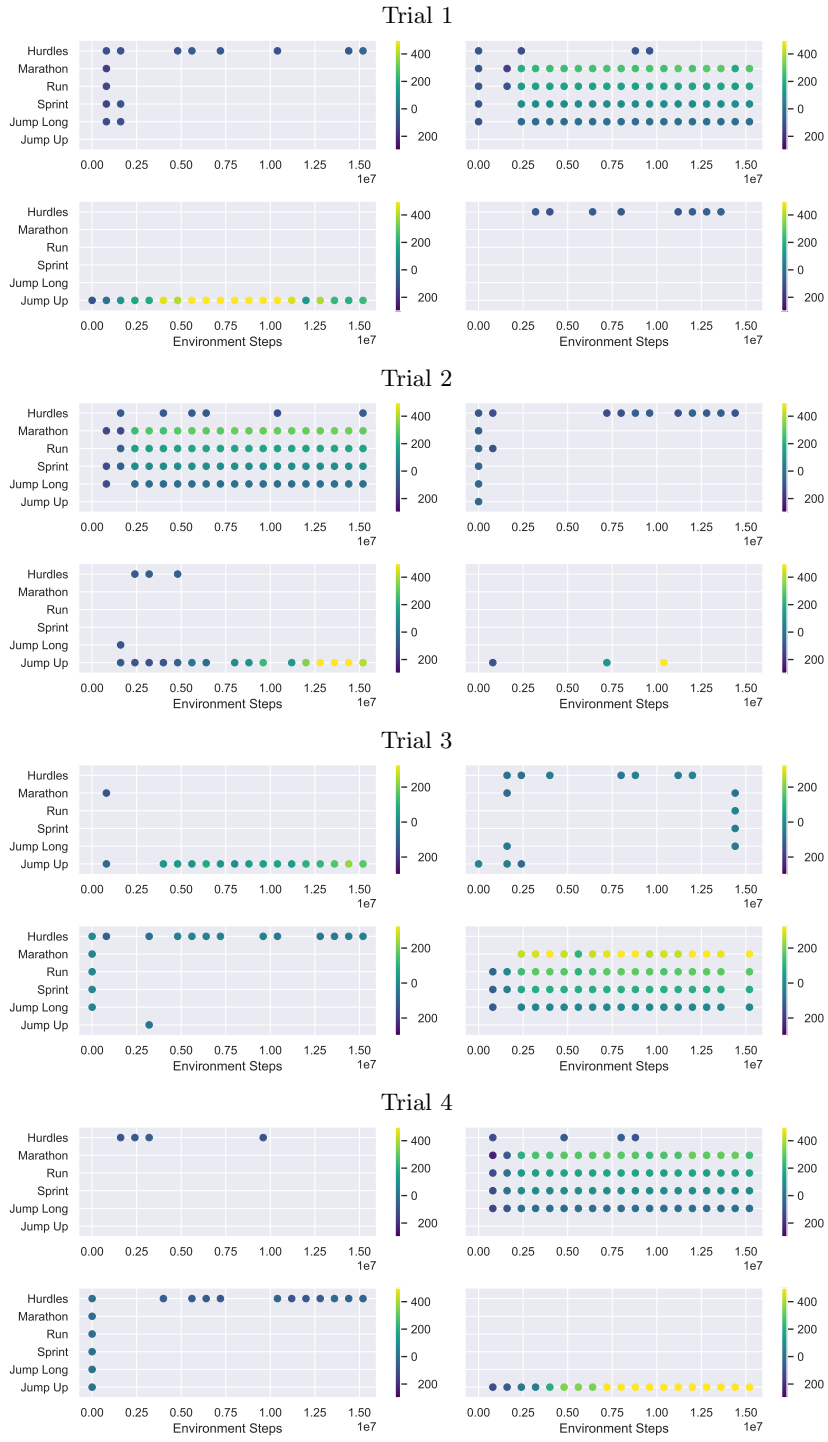

Fig. 9: Shown are the assignments from 4 randomly picked trials on the track and field BipedalWalker task set.

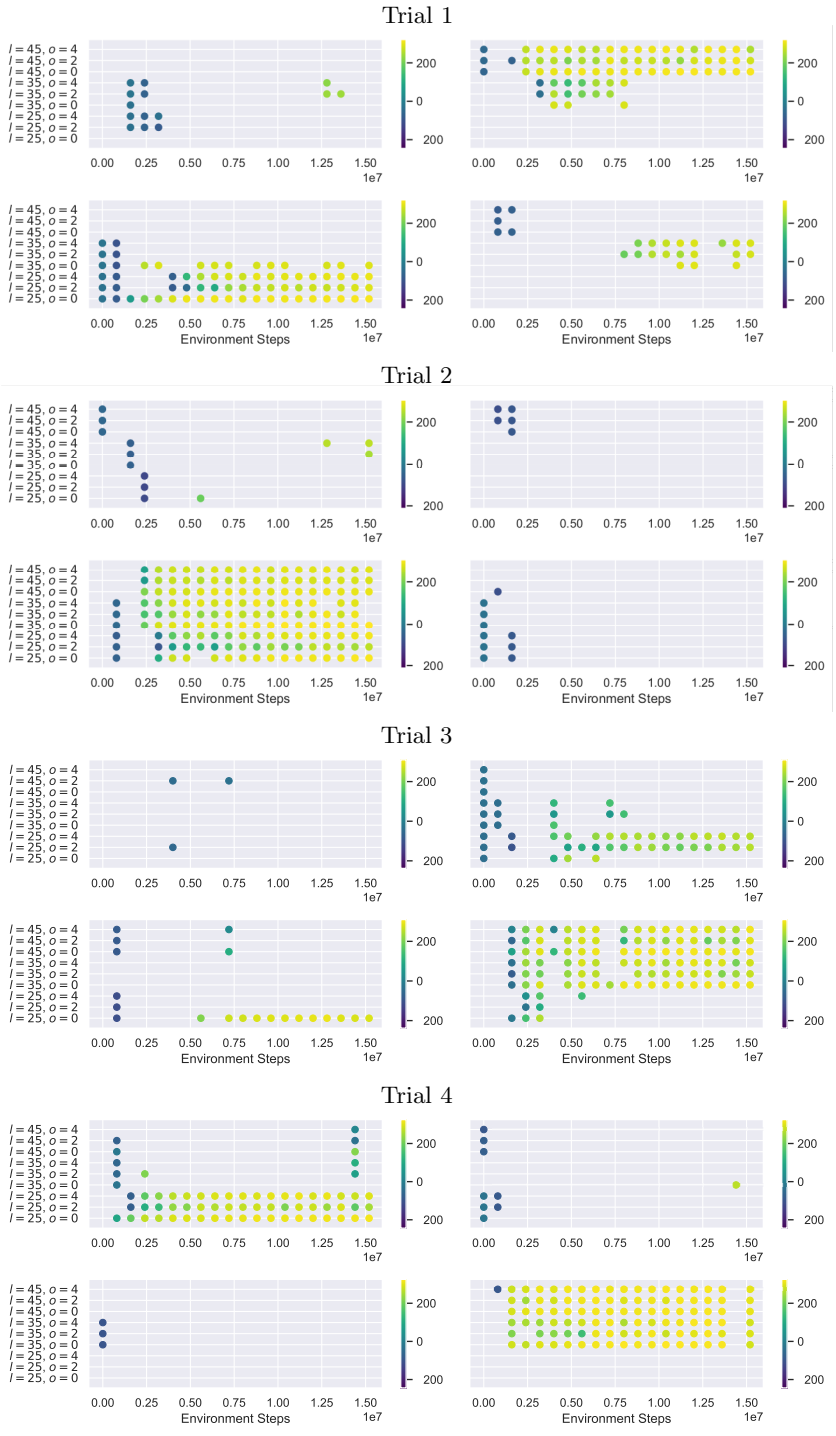

Fig. 10: Shown are the assignments from 4 randomly picked trials on the first BipedalWalker task set. l refers to the lenghts of the legs, o refers to the frequency of obstacles.

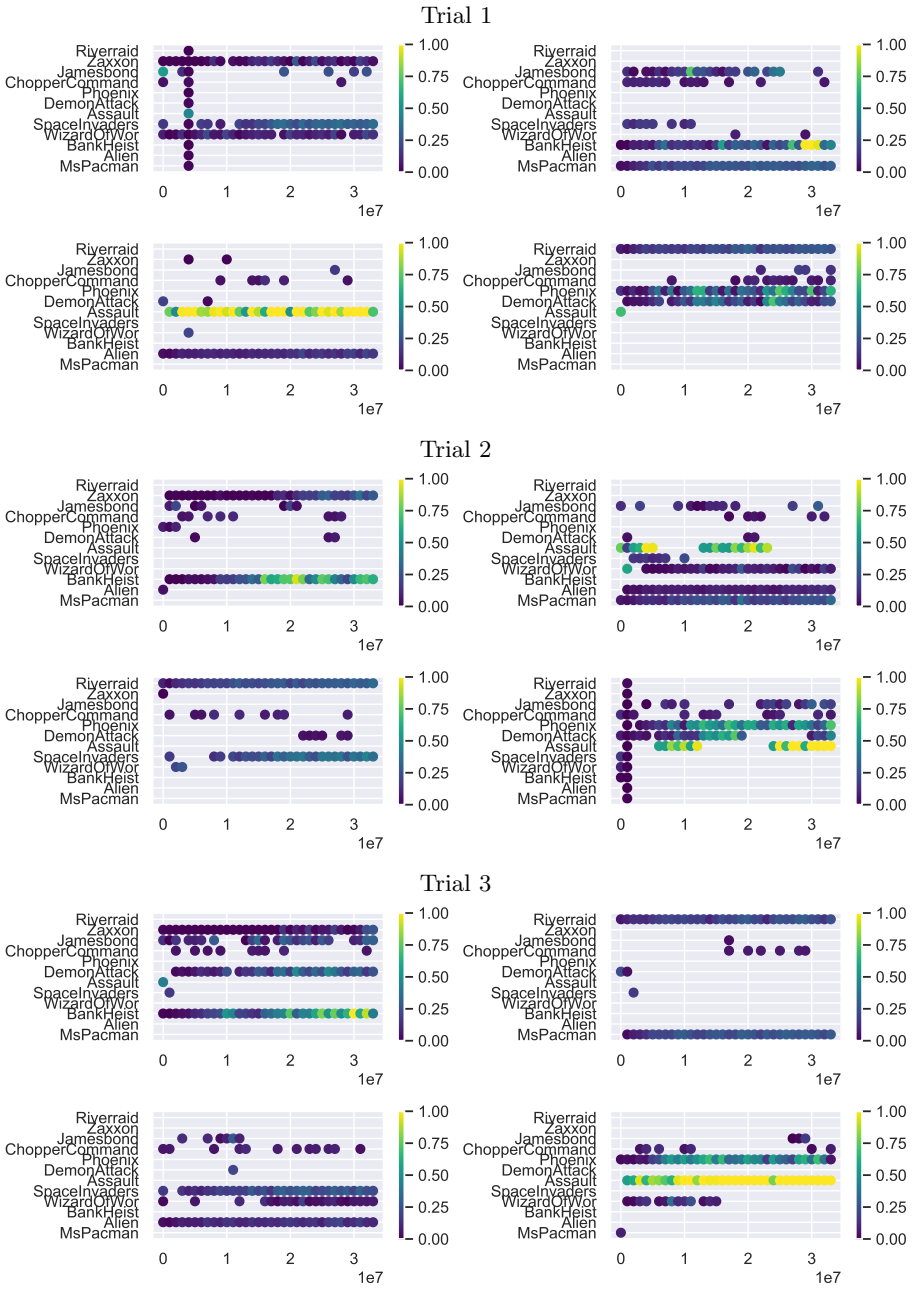

Fig. 11: Shown are the assignments of all three trial that were run on the set of Atari games. The color represents the human-normalized score per game.

D Performance Gap to Random Clusters

In Figure 8 we investigated the importance of the E-Step in our approach, by comparing to an ablation which randomly assigns tasks to policies at the start. These results showed that using random assignments performs worse, highlighting the importance of using related clusters of tasks. Here we will investigate how the difference between the return when using our EM method G_{EM} or random assignments G_{rand} changes depending on the number of tasks. When using a single policy or a policy for each task our method becomes identical to the baselines. We hypothesize that the difference should be maximal when using as many policies as there are true underlying clusters in the task set.

To test this hypothesis we perform experiments on our grid-world task set with 12 goals distributed to the four corners and show the return gap $G_{\text{EM}} - G_{\text{rand}}$ in Figure 12. The experiments confirm our hypothesis, showing that the return gap increases with the number of policies before reaching a maximum when it matches the true clusters at $n = 4$. Afterwards it starts to decrease.

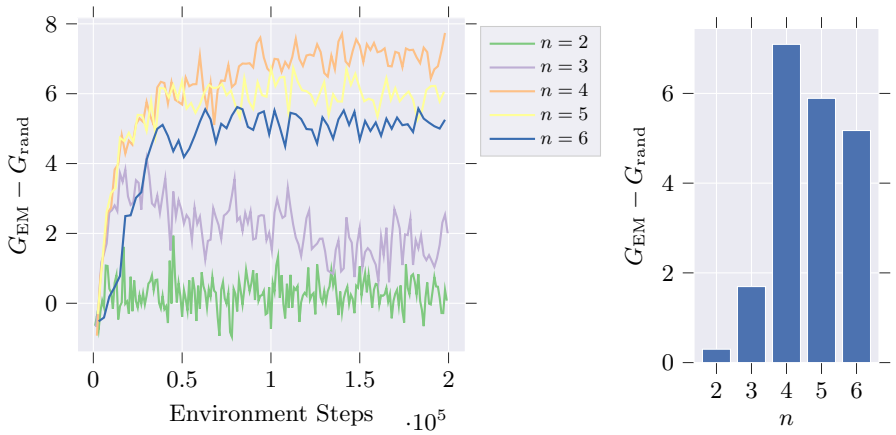

Fig. 12: Show is the difference between using random assignments or our EM approach for different numbers of policies. On the left the development during training is shown, on the right the average performance gap over the last 10% of the training is visualised.