Universal benchmark for actuation dynamics adaptation in reinforcement learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Enabling reinforcement learning (RL) agents to adapt to changing environment dynamics is crucial for robustness. Consider the case where a robot's motors and gears change their behavior due to wear and tear over time, or where an old used-up part gets replaced. Current literature primarily emphasizes resilience against observation noise, distractions in the environment or shifts in physical properties of the world. However, the problem of continual shifts or sudden changes in actuation dynamics is relatively unexplored. To facilitate systematic research in that regard, we contribute a *Universal Benchmark for Actuation Dynamics Adaptation (UBADA)*[1]. We present a universal set of wrappers compliant with the Gymnasium API standard, providing a multitude of challenges with continual (serial) and multi-task (parallel) learning scenarios of changing action dynamics. We showcase the problem on visual and low-dimensional proprioceptive inputs, with dense or sparse rewards, utilizing the state-of-the-art learning algorithms Soft-Actor-Critic (SAC) and Data-regularized Q (DrQ).

1 Introduction

In current research to systematically investigate transfer capabilities of a reinforcement learning (RL) agent, one typically asks questions like the following: "The agent can *open a door*, but can it *open a window*?" [62, 61, 50, 47, 58]. Benchmarks like Meta-World [62] or CompoSuite [37] contain 10, 50 or up to 256 different tasks like those used in the example above. Another perspective on transfer learning involves exploring generalization and robustness to perturbations in the observation [18]. For example, one could ask: "The agent can *open a door*, but can it also *open the door if the floor is distracted with colorful dots*?".

With our contribution, we emphasize an often neglected angle on robustness and the ability to adapt to changes in the agent's embodiment. For example, we would ask: "The agent can *open a door*, but could it also *open a door if it were weaker*?". To address such questions, we change the dynamics of the system by manipulating the agent's action effect.

To illustrate, assuming you are reading this work on a computer. Navigate to your system settings and invert the scrolling direction of your trackpad or mouse. With the switched scrolling direction, return to the document and resume reading. You will notice that adjusting to the new system behavior might require a few trials, but overall you can adapt relatively quickly, and you do not need to re-learn to operate the trackpad or mouse from scratch.

A possible motor psychological explanation for how humans adapt to new dynamics involves a shift of attention to the relevant sensory modalities. Humans have efficient innate abilities to handle

[1] `https://github.com/xxx/xxx`

Submitted to ICML 2024 Workshop: Automated Reinforcement Learning: Exploring Meta-Learning, AutoML, and LLMs. Do not distribute.

conflicting sensory stimuli. When using the inverted trackpad or computer mouse, humans suppress the attention on haptic and proprioceptive senses and focus on the unfamiliar visual stimulus on the screen. The unfamiliar visual stimuli are sensory conflicts that emerge when novel experiences contradict expectations that have been formed through lifelong learning [32]. See Figure 1 for an illustration of the inverted computer mouse and Section A in the Appendix for more examples.

The example with the inverted mouse is rather constructed, but it is a good proxy for many realistic cases in robotics. For example, wear and tear of motors can lead to fluctuations in their effectiveness over time. Replacing a motor or a joint yields different dynamics that have to be relearned. Even in the case of a malfunctioning part, the robot should be able to adapt and compensate for the missing functionality.

In order to address such problems or similar ones in reinforcement learning (RL), significant effort is being devoted to improving robustness, (zero-shot) adaptation and overall generalization [59, 18, 49, 24]. However, current RL benchmarks and methods primarily focus on addressing noise, augmentation, perturbation, or distractions at the observation level, often neglecting changes in actuation dynamics.

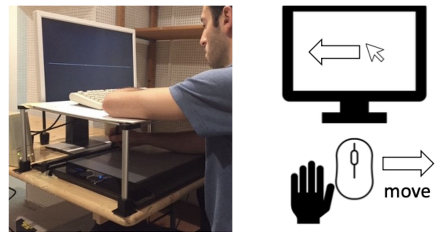


Figure 1: Psychological experiments requiring subjects to learn novel unexpected dynamics [13, 31, 32], e.g., moving the hand to the right causes the mouse cursor to move left. Illustration adapted from Liesner et al. [32].

To bring greater attention to this aspect, we propose our main contribution: a *Universal Benchmark for Actuation Dynamics Adaptation (UBADA)*, designed to analyze the adaptation capabilities of RL agents when confronted with dynamic changes in the agent's actuation. UBADA can be considered a set of universal wrappers for environments following the Gymnasium API standard [53]. It is *universal*, in the sense that it is not limited to specific hand-crafted environments, and it is independent of the type of sensory input. Therefore, it can convert a variety of publicly available standard environments into a challenging task for the continual learning, curriculum learning (serial transfer) or multi-task learning (parallel transfer) setup. In addition, it can be used to extend existing benchmarks by increasing the number of available tasks. The focus on adaptation and robustness and the universal nature of our benchmark facilitates research into AutoRL techniques aimed at optimizing agents for generality and applicability [4, 38]. The second contribution of this work involves evaluating the transfer capabilities of two state-of-the-art RL algorithms, namely SAC [16] and DrQ [60]. Therefore, we utilize environments from Gymnasium MuJoCo [53, 52] (goal-conditioned and non goal-conditioned) and DM Control [54].

2 Related work

Our contribution is a benchmark for changing action dynamics. Therefore, it is related to transfer learning, multi-task and meta learning, continual and curriculum learning. Another relevant field of research is RL in modified environments.

2.1 Transfer, multi-task and continual learning benchmarks

Henderson et al. [19] regard the Arcade Learning Environment (ALE) [3] as the primary benchmark for evaluating multi-task learning in domains with discrete actions. However, for continuous actions they observe a lack of a standardized evaluation environments for multi-task learning at that time. To fill this gap, they introduce a benchmark [19], aiming to facilitate a systematic comparison of multi-task, transfer, and lifelong learning in continuous domains.

The creators of Meta-World [62] emphasize the possible occurrence of negative transfer between tasks in the ALE. Hence, they motivate their work by the idea that positive transfer between different tasks should be possible. They propose a set of related yet diverse robotics tasks that share the same robot, action space, and workspace. It became a very popular benchmark used to study multi-task and meta learning [61, 63, 47, 50].

In robotics, more benchmarks comprising a multitude of environments exist [14, 22, 54, 58, 37]. For example, CompoSuite [37] introduces a series of tasks where a robotic arm manipulates individual objects to achieve objectives while navigating obstacles. This framework facilitates the evaluation of RL approaches in learning the compositional structure of tasks and their ability for compositional generalization to unseen tasks. Avalanche RL [35, 5] provides support for a continuous stream of diverse environments. Additionally, CORA [40] encompasses metrics and baselines designed to assess various aspects of continual RL. These aspects include catastrophic forgetting, plasticity, generalization, and sample-efficient learning. The benchmarks are based on diverse environments such as ALE [3, 25, 45, 43], ProcGen [8, 21], NetHack [28, 44], ALFRED [46] (built upon AI2-THOR [26]). While many efforts in continual learning tend to emphasize catastrophic forgetting, Continual World [58] advocates for the significance of forward transfer. Another notable contribution is BabyAI [7], which involves the creation of multiple grid-world environments of increasing difficulty. Here, the primary focus is on investigating grounded language learning.

2.2 Environment modification benchmarks

The perceived dynamics of an environment can be changed by directly modifying the agent's observation. For tasks derived from pixel inputs, this could involve the incorporation of color and background distractions as valuable benchmarks [18, 49]. Another approach is a straightforward data augmentation technique, as proposed by Yarats et al. [60]. The authors utilize perturbations on the input observations commonly employed in computer vision tasks to regularize the value function. With DrQ they show, that this augmentation approach proves to be particularly effective in enhancing the performance of the SAC algorithm [16]. Not necessarily focused on RL from pixels, in their study, Khraishi and Okhrati [23] also delve into the application of data augmentation techniques in RL to enhance model performance and promote diversity in training data.

Another way to modify the perceived dynamics of an environment is by using action noise. Action noise plays an important role for exploration behavior [20, 12], as in Deep Deterministic Policy Gradient (DDPG) [33] and Twin Delayed DDPG (TD3) [15]. It is commonly generated by utilizing the Ornstein–Uhlenbeck process [55] or drawing from uncorrelated Gaussian distributions. In the case of SAC [16], where the policy is stochastic, Gaussian noise is implicitly introduced through the sampling procedure. In our work, the modification of actuation dynamics serves a different purpose than exploration: We focus on analyzing the adaptability of agents to dynamic changes.

Langlois and Everitt [29] have the human-in-a-loop aspect in mind and analyze how an agent behaves if its intended actions are overridden or manipulated by the environment or another agent. That is also related to the work on safety in RL by Leike et al. [30].

van Seijen et al. [56] want to measure how quickly an agent that is trained on task A changes its policy after it is placed in task B. They specifically experiment with maintaining identical transition dynamics between tasks A and B, while introducing a local difference in the reward function. For that purpose they introduce the local change adaptation (LoCA) regret metric.

Similar to our method, the CARL benchmark makes environments configurable and facilitates training agents to generalize across different instances (contexts) of the same environment. This supports research into AutoRL techniques aimed at optimizing agents for broad generality [4, 38].

Furthermore, there is a trend in research to modify or perturb system parameters, such as body part size or gravity, in continuous control tasks [19, 11].

The work that is most closely related to ours is the Real-World Reinforcement Learning (RWRL) Challenge Framework [11]. It provides a diverse suite of tasks to change the morphology of the agent or the physics of the world, like the friction of the ground. Related to the action effect it allows action offsets, action noise, repetitive actions, and action drops. However, it exclusively includes proprioceptive observations and does not allow for visual observations, and it is constrained by a limited, predefined set of environments. While the concepts from RWRL could in principle be applied to novel environments, doing so within the RWRL system may pose considerable challenges. In our approach, we address these drawbacks to provide a more general solution which is implemented as a set of universal wrappers that modify actions. Furthermore, our approach facilitates extensive configurability. For instance, action effects can be altered selectively for individual action dimensions, e.g., specific joints in a robot arm, rather than modifying the action as a whole.

3 Background

Reinforcement learning builds on Markov Decision Processes (MDP), defined as tuples $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$. The behavior of the world is determined by the transition probability $P(s'|s, a)$, whereas the policy $\pi(a|s)$ generates an action $a \in \mathcal{A}$ based on inputs $s \in \mathcal{S}$, to maximize the cumulative return, the sum of discounted rewards $\sum_{t=0} \gamma^t r_t$, with $r_t = R(s_t, a_t, s_{t+1})$. The particularity of this work is that we introduce a modification of the action $p(\bar{a}|a)$ which changes over time. Hence, in the transition dynamics $P(s'|s, a)$ the action is manipulated by our wrappers to yield $P(s'|s, \bar{a})$. The outcome could be similar or totally diverge from the old one.

An extended background section on transfer learning in RL, multi-task RL and continual RL is provided in the Appendix B.

4 Manipulate the action effect

UBADA is designed around the Gymnasium API standard [53]. It consists of a variety of wrappers that manipulate the action effect of the agent within its environment. They are universally applicable insofar as they are independent of the selected base environment and independent of whether pixels or state features are processed. Regardless of the action effect wrapper selected, the action space remains the same as in the original base environment. The wrappers provide versatility by allowing toggling on and off, seamless combination with each other, and the capability to alter the action effect of one or all action dimensions. Additionally, they are configurable. This design grants considerable freedom in tailoring the experimental setup according to specific preferences.

Note that the possible modifications for discrete actions are limited due to their inherent limited flexibility, the focus is on continuous actions.

The action effect wrappers are listed in the following. An overview including a brief description and motivation can be found in Table 1 in the Appendix. In general, all wrappers are configurable by the choice of the action dimension which is modified and if applicable by another specific parameter which is mentioned in the following.

`InvertAction` Consider the concept of the backward bicycle, where attempting a left turn of the handlebar leads to movement in the opposite, or right, direction (cf. Appendix A). This wrapper inverts the continuous action, simply by switching the sign of one or all action dimensions.

`ScaleAction` This wrapper scales the continuous action by multiplying either one or all action dimensions with a scalar value. A scenario where this might be relevant is to simulate decaying motor efficiencies over a robot's lifetime. The scaling factor is configurable.

`OffsetAction` Similarly, this wrapper adds a constant scalar value to one or all dimensions of the continuous action. This might simulate an inadequate calibration of a system, reflecting a systematic error. The offset is configurable.

`NoiseAction` This wrapper adds a random value to either one or all dimensions of the continuous action. The random value is drawn from a Gaussian distribution with a mean $\mu = 0$ and a user-defined standard deviation σ. While being applied at the environment level, it can be perceived as analogous to Gaussian action noise commonly utilized in algorithms like DDPG or TD3. In a robotics context, this could simulate the presence of random errors. The choice of σ is configurable.

`SineNoiseAction` Expanding upon the `NoiseAction` wrapper, this implementation adds a sine offset to the Gaussian noise. A single parameter within the wrapper specifies both the standard deviation σ and the amplitude of the sinusoidal wave. This allows for analyzing the impact of both an unpredictable component (Gaussian noise) and a predictable element (sine offset) on the action. The choice of σ is configurable.

`ZeroAction` It randomly sets one or all dimensions of the continuous action to zero for a defined number of steps. Again, in a robotics context, this wrapper might correspond to loose contacts or sporadic complete engine failures. The probability of a dimension (or the whole action) being zeroed and the duration of this zeroing effect are both configurable.

`RepeatAction` Similar to `ZeroAction`, but instead of setting to zero this wrapper repeats one or all dimensions of the continuous or discrete action for a defined number of steps. Proposed by

Machado et al. [36] this effect is also known as sticky actions, though it is usually not possible to repeat only one dimension of the action. The probability of a dimension (or the whole action) being repeated and the duration of this repetition are both configurable.

`SwapAction` This wrapper swaps either one dimension of the continuous or discrete action with another randomly picked one or shuffles randomly all dimensions. The changed order of dimensions is kept for the whole period of time this wrapper is activated. Naively, one can imagine interchanged cables wrongly connecting the motors in a robot system.

An additional `DynamicsHintObservation` wrapper can be applied optionally, it augments the observation space with a one-hot encoding which functions as a task identifier and informs the agent about the applied and active action effect wrapper.

5 Action effect benchmark

In the context of RL, the term *task* can sometimes be subject to varying interpretations. In our definition, each dynamic modification is considered a distinct task, or context, even though the underlying environment remains constant.

In this section we provide experimental results using two environments: `HalfCheetah-v4` based on work by Wawrzyński [57] and available through Towers et al. [53] and `walker-walk-v0` from DM Control [54], utilized via Towers et al. [53] and Tai et al. [51] with proprioceptive inputs. Extended results for more kinds of environments (goal-conditioned, visual observation inputs) showcasing the versatility of the wrapper approach are provided in Appendix G (cf. Table 3 for an overview). We use the learning algorithms SAC [16] for proprioceptive and DrQ [60] for visual input observations. For goal-conditioned environments we combine SAC with Hindsight Experience Replay (HER) [1]. We use implementations provided by Kostrikov [27] (JAXRL2) and Raffin et al. [42] (Stable-Baselines 3). Hyperparameters are kept default as provided in their implementations (cf. Section D in the Appendix). In general, results are obtained by averaging across five runs with different random initialization.

On the one hand, our intent is to demonstrate transfer capabilities of general purpose agents while learning under changing conditions. Although very interesting, we do not investigate how state-of-the-art approaches specifically designed for multi-task RL [61, 47, 50] or continual RL [25, 43] behave under action dynamic changes. We also do not analyze the effect of different replay buffer sizes which possibly influences learning and transfer capabilities in the continual learning case.

On the other hand, the purpose of the following experiments and those in Section G in the Appendix is to illustrate the universal applicability of action modifications across various environments. In some cases the wrappers yield fundamentally different modifications. Not all modifications are supposed to be transferable to real-world scenarios. Still, they can be useful to reveal the limits of the agent, in particular with regard to dimensions of robustness and transfer capabilities. An extended discussion on the classification of the different modifications is provided in Section E in the Appendix.

5.1 Sequential training setup (continual learning)

5.1.1 Experimental setting

The use of wrappers on top of the actual environments functions like a modular system. Changes can be combined with each other as desired and triggered at any time step. For the sequential experiments, we first use the initial environment and then switch the dynamic of the environment by toggling the respective action effect wrapper after a defined number of steps. For the environments `walker-walk-v0` and `HalfCheetah-v4` we train each task $\mathcal{T}_i$ for $T_{\mathcal{T}} = 500000$ and $T_{\mathcal{T}} = 1000000$ steps, respectively. The choice of $T_{\mathcal{T}}$ is made to ensure that, at the very least, the unmodified environment can be trained to a reasonable degree. We only consider sequences where $T_{\mathcal{T}}$ is constant across all tasks. For example, if we consider the initial dynamic and nine modifications of the `walker-walk-v0` environment, the sequence has $N = 10$ tasks and in total we train for $T = N \cdot T_{\mathcal{T}} = 5000000$ steps. The i-th task is trained during the interval $t \in [(i-1) \cdot T_{\mathcal{T}}, i \cdot T_{\mathcal{T}}]$.

Unlike Powers et al. [40] we do not cycle through the sequence of tasks multiple times. (Powers et al. [40] also only cycle multiple times in the general case, however, to compute forward transfer and forgetting they use one cycle.)

While it is possible to have multiple wrappers activated simultaneously, for a clearer understanding of the impact of the individual wrappers, we focus on a scenario where, at each time step, only one wrapper is active. Moreover, within our experiments we do not combine multiple types of wrappers, e.g., `ScaleAction` and `OffsetAction`. Possibilities are extensive.

5.1.2 Metrics

The metrics considered and their implementation were mainly based on the work of Lopez-Paz and Ranzato [34], Díaz-Rodríguez et al. [10], Wołczyk et al. [58] and Powers et al. [40].

Let $R_t(\pi, \mathcal{T}_i)$ be the episodic return, the undiscounted sum of rewards received over an episode, under the policy π on task $\mathcal{T}_i$ at time step t. To enhance the understanding of results and to account for different attainable returns in different environments we define a performance measure $\rho_i(t)$ by normalizing the return by its maximum value over all time steps and over all tasks. Note, within a sequence each task is derived from the same environment, hence we use the maximum over all tasks which differs from Powers et al. [40]. When comparing performances of two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$ within one task sequence we generally assume $i < j$.

$$\rho_i(t) = \frac{R_t(\pi, \mathcal{T}_i)}{\max\limits_{1 \le t \le T, 1 \le k \le N} R_t(\pi, \mathcal{T}_k)} \tag{1}$$

Average performance The performance at any time step averaged across all tasks. We consider its final value $\mathrm{P}(t = T)$ for evaluation.

$$\mathrm{P}(t) = \frac{1}{N} \sum_{i=1}^{N} \rho_i(t) \tag{2}$$

Forward transfer Measures the influence that learning a task $\mathcal{T}_i$ has on the performance of a future task $\mathcal{T}_j$. It can occur when the model is able to perform zero-shot learning [10], e.g., Powers et al. [40] refer to the corresponding metric as zero-shot forward transfer.

$$FT = \frac{2}{N(N-1)} \sum_{j=2}^{N} \sum_{i=1}^{j-1} FT_j \quad \text{where} \quad FT_j = \rho_j(t = i \cdot T_{\mathcal{T}}) - \rho_j(t = (i-1) \cdot T_{\mathcal{T}}) \tag{3}$$

Backward transfer In contrast to the findings reported by Wołczyk et al. [58], where they observe no occurrence of backward transfer in their scenario, we find the prospect intriguing for our specific purposes. Differences between tasks in our context may be more nuanced, implying that learning a modification of an environment could potentially enhance performance on the initial task in a retrospective manner.

To be precise, we actually use positive backward transfer following the argumentation of Díaz-Rodríguez et al. [10] and Wołczyk et al. [58].

$$BT = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} BT_i \quad \text{where} \quad BT_i = \max\{0, \rho_i(t = j \cdot T_{\mathcal{T}}) - \rho_i(t = (j-1) \cdot T_{\mathcal{T}})\} \tag{4}$$

Forgetting It represents the decrease of performance from a learned task during later tasks.

$$F = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} F_i \quad \text{where} \quad F_i = \max\{0, \rho_i(t = (j-1) \cdot T_{\mathcal{T}}) - \rho_i(t = j \cdot T_{\mathcal{T}})\} \tag{5}$$

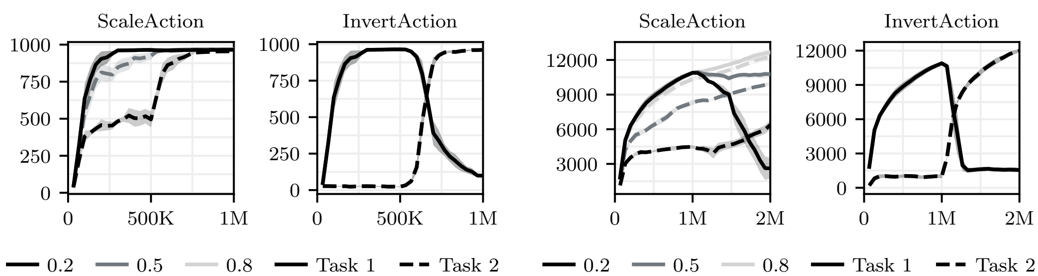
In alignment with the argumentation presented in Powers et al. [40] and akin to the approach taken by Lopez-Paz and Ranzato [34], unlike Chaudhry et al. [6] we use the performance right before training on a new task rather than the maximum value observed so far. That approach corresponds to a more task isolated view, hence Powers et al. [40] denotes their metric isolated forgetting. Wołczyk et al. [58] consider the decrease of performance after ending the training on the whole task sequence. They compute $F_i = \rho_i(t = i \cdot T_{\mathcal{T}}) - \rho_i(t = T)$, which corresponds to a more cumulative effect of forgetting.

Also note, to emphasize two different concepts, we explicitly distinguish positive backward transfer and forgetting, although Equation 4 and 5 are quite related [10].

5.1.3 One dynamic switch

In this scenario, a SAC agent trains for $T_{\mathcal{T}} = T/2$ steps on a task 1, $\mathcal{T}_1$, and continues training on task 2, $\mathcal{T}_2$, for the same number of steps. $\mathcal{T}_1$ is the unmodified environment. $\mathcal{T}_2$ comes from the same environment, but with modified action dynamics. The modification is defined by the type of action wrapper, the action dimension to be modified, and if applicable a wrapper specific configuration value. In Figure 2 we use the `InvertAction` and the `ScaleAction` wrapper. We modify only action dimension 0. For `ScaleAction` we consider three scaling factors $\{0.2, 0.5, 0.8\}$.

With the one dynamic switch setup and the selection of action effect modifications we showcase the different metrics defined in Section 5.1.2. While the `InvertAction` switch is a perfect example for (catastrophic) forgetting ($F = 0.92$), we cannot observe any forgetting of the first task after the `ScaleAction` switch (cf. Figure 2a). It applies analogously for the forward transfer. Just considering individual task returns, the agent is not able to transfer knowledge from the baseline environment to the same environment with the `InvertAction` modification. `ScaleAction` allows for forward transfer, correlating with the manifestation of the switch. The lower the scaling value, the higher the contrast between tasks. For scaling values 0.8, 0.5 and 0.2 we have $FT = 0.96$, $FT = 0.89$ and $FT = 0.49$, respectively. To analyze positive backward transfer we consider the `HalfCheetah-v4` environment (cf. Figure 2b). For a `ScaleAction` switch and lower scaling values we can observe positive backward transfer, while forgetting emerges otherwise. For scaling values 0.8, 0.5 and 0.2 we have $BT = 0.12$, $BT = 0.02$, and $BT = 0.0$, respectively.



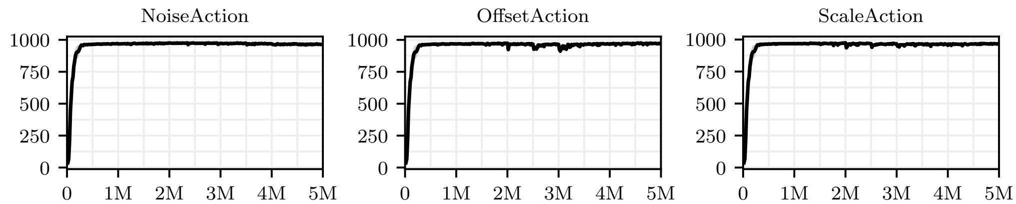
(a) `walker-walk-v0` environment. (b) `HalfCheetah-v4` environment.

Figure 2: Per task evaluation returns for the sequential training setup with one dynamic switch. To establish the switch we use the `InvertAction` and the `ScaleAction` wrapper. We modify only action dimension 0. For `ScaleAction` we consider three scaling factors $\{0.2, 0.5, 0.8\}$. The switch happens at $T/2$.

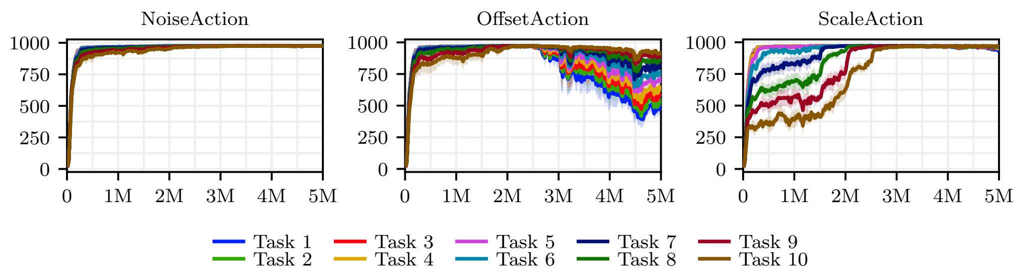
5.1.4 Continual adaptation to new dynamics

In this scenario a SAC agent trains for $T_{\mathcal{T}} = T/10$ steps on the unmodified action dynamic, task $\mathcal{T}_1$. Then nine times for $T_{\mathcal{T}} = T/10$ steps on modified action dynamics, varied by the wrapper value, tasks $\{\mathcal{T}_2, \mathcal{T}_3, \dots, \mathcal{T}_{10}\}$. We construct the sequence of tasks with increasing difficulty. One could think of a curriculum learning schedule. For `NoiseAction` tasks are determined by the standard deviation for the Gaussian noise in $\{0.1, 0.2, \dots, 0.9\}$, for `OffsetAction` the action offset is in $\{-0.1, -0.2, \dots, -0.9\}$ and for `ScaleAction` the scaling factor is in $\{0.9, 0.8, \dots, 0.1\}$.

Consider solely the training return in Figure 3a, the agent can adapt to the action effect changes smoothly without larger collapses. In that regard there is no difference between the three action effect wrapper. Now consider the evaluation returns for each individual task at every time step in Figure 3b. Differences in forward transfer and forgetting stand out. While the forward transfer in the `NoiseAction` and `OffsetAction` experiments mostly happens during the training of task $\mathcal{T}_1$, in the `ScaleAction` experiment it occurs on a wider time span, i.e., during the training of $\mathcal{T}_1$ to $\mathcal{T}_5$. Nevertheless, overall according to Equation 3 all three experiments have the same total $FT = 0.19$. As the metric is averaged across tasks and training periods, it cannot provide an isolated view on forward transfer. Observing Figure 3b or the distribution of FT_j (cf. Equation 3) is more appropriate. Towards the end of the training sequence the agent shows to some degree forgetting of the earlier tasks in the context of the `OffsetAction` experiment. The longer it is been since the training of a task, the more is forgotten. Interestingly, this notion is not observed in the `NoiseAction` and `ScaleAction` experiments.



(a) Train returns.



(b) Evaluation returns.

Figure 3: Returns for the sequential training setup with continual adaptation. A SAC agent is trained on the `walker-walk-v0` environment. After first training on the unmodified action dynamic, action effect changes occur every $T/10$ steps. For `NoiseAction` tasks are determined by an action wrapper value in $\{0.1, 0.2, \ldots, 0.9\}$, for `OffsetAction` in $\{-0.1, -0.2, \ldots, -0.9\}$, for `ScaleAction` in $\{0.9, 0.8, \ldots, 0.1\}$. The modification is done on action dimension 0.

5.2 Parallel training setup (multi-task learning)

5.2.1 Experimental setting

The main difference compared to the sequential setup is that multiple dynamics are to be learned simultaneously and that task identifiers are passed to the agent for this purpose utilizing the `DynamicsHintObservation` wrapper. For the environments `walker-walk-v0` and `HalfCheetah-v4` we train the agent for $T = 1000000$ and $T = 2000000$ steps, respectively. The steps eventually trained per task $\mathcal{T}_i$ are evenly distributed. In general we consider only a pair of tasks, hence $\{\mathcal{T}_1, \mathcal{T}_2\}$.

In our multi-task experiments we solely rely on one-hot task identifiers. For example, Sodhani et al. [47] communicate tasks from Meta-World [62] to the agent through language descriptions. They show that similarities between these descriptions can be exploited for learning. Applying this idea to modifications in action effects would yield interesting experiments, which we defer to future research endeavors.

5.2.2 Metrics

Again, $R_t(\pi, \mathcal{T}_i)$ is the episodic return under the policy π on task $\mathcal{T}_i$ at time step t. We define a performance measure $\rho_i(t)$ by normalizing the return by its maximum value over all time steps and over all tasks. As in the sequential case, Equation 1 applies.

Average performance Similar to the setup for sequential training, we measure the performance at any time step averaged across all tasks. We consider its final performance $\mathrm{P}(t = T)$ for evaluation. Equation 2 applies.

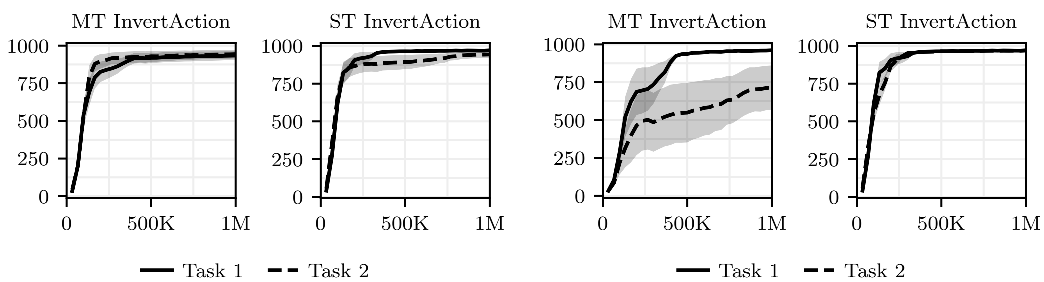
Parallel transfer In the sequential learning setup Chaudhry et al. [6] define an intransigence metric for measuring forward transfer by comparing the maximum return for a task trained independently to the return achieved while it was trained sequentially [58, 40].

For the multi-task learning setup we propose an analogous metric to assess parallel transfer. We compare the performance ρ_i^b for a task trained independently (single-task, baseline) to the performance ρ_i achieved while it was trained in parallel to other tasks (multi-task). We assume both, the single-task and the multi-task run, train for the same number of steps. ρ_i and ρ_i^b are both normalized by the overall (single-task or multi-task condition) maximum value.

$$PT = \frac{1}{N}\sum_{i=1}^{N} PT_i \quad \text{where} \quad PT_i = \rho_i(t = T) - \rho_i^b(t = T) \tag{6}$$

5.2.3 Pair-wise multi-task

We train a SAC agent in parallel on two tasks in the `walker-walk-v0` environment, i.e., on the unmodified action dynamic and on a modification of such. We compare the performances of the two tasks trained in a multi-task setting to their baseline performances achieved in a single-task setting. While we observe hardly any parallel transfer if the `InvertAction` modification is done on only one dimension (cf. Figure 4a, $PT = -0.02$), if it is done on all action dimensions the agent cannot cope with both tasks in parallel. Figure 4b illustrates an instance of negative parallel transfer ($PT = -0.14$). In all our experiments (cf. Table 11 and 12 in the Appendix) we were not able to observe positive parallel transfer which might be due to not using a specific multi-task learning algorithm.



(a) Modification on action dimension 0.

(b) Modification on all action dimensions.

Figure 4: Evaluation returns for a SAC agent trained on the `walker-walk-v0` environment. It independently trains on two tasks in parallel (multi-task, MT) and individually (baseline, single-task, ST). Tasks correspond to the unmodified action dynamic (solid lines) and a `InvertAction` modification (dashed lines).

6 Conclusions

We propose a *Universal Benchmark for Actuation Dynamics Adaptation (UBADA)* comprising a set of universal wrappers adhering to the Gymnasium API standard. UBADA can modify arbitrary environments and turn them into challenges for both continual (serial) and multi-task (parallel) learning scenarios. It focuses specifically on adaptation to changing action dynamics. With our experiments we utilize these challenges to advance the understanding and evaluation of RL agents' transfer capabilities under continual and sudden dynamic changes. Research on adaptability to changing dynamics is crucial for robustness and eventually real-world applications. With our benchmark, we place a clear emphasis on dynamic changes associated with variations in action effects, and we are confident that UBADA can aid systematic research on that perspective of robustness.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[2] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap. Distributed Distributional Deterministic Policy Gradients. In *International Conference on Learning Representations*, 2018.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] C. Benjamins, T. Eimer, F. Schubert, A. Mohan, S. Döhler, A. Biedenkapp, B. Rosenhahn, F. Hutter, and M. Lindauer. Contextualize Me - The Case for Context in Reinforcement Learning. Transactions on Machine Learning Research, 2023.

[5] A. Carta, L. Pellegrini, A. Cossu, H. Hemati, and V. Lomonaco. Avalanche: A PyTorch library for deep continual learning. *Journal of Machine Learning Research*, 24(363):1–6, 2023.

[6] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr. Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[7] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning, 2019.

[8] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging Procedural Generation to Benchmark Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2048–2056. PMLR, 2020.

[9] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry. Gymnasium robotics, 2023.

[10] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni. Don't forget, there is more than forgetting: New metrics for Continual Learning. *CoRR*, abs/1810.13166, 2018.

[11] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. An empirical investigation of the challenges of real-world reinforcement learning, 2021.

[12] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink Noise Is All You Need: Colored Noise Exploration in Deep Reinforcement Learning. In *The Eleventh International Conference on Learning Representations*, 2022.

[13] J. P. Ebert and D. M. Wegner. Time warp: Authorship shapes the perceived timing of actions and events. *Consciousness and Cognition*, 19(1):481–489, 2010. ISSN 1053-8100. doi: 10.1016/j.concog.2009.10.002.

[14] B. Ellenberger. PyBullet gymperium, 2018/2019.

[15] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[17] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019.

[18] N. Hansen and X. Wang. Generalization in reinforcement learning by soft data augmentation. In *International Conference on Robotics and Automation*, 2021.

[19] P. Henderson, W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek. Benchmark Environments for Multitask Learning in Continuous Domains, 2017.

[20] J. Hollenstein, S. Auddy, M. Saveriano, E. Renaudo, and J. Piater. Action noise in off-policy deep reinforcement learning: Impact on exploration and performance. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

[21] M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. Transient Non-stationarity and Generalisation in Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2020.

[22] S. James, Z. Ma, D. Rovick Arrojo, and A. J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

[23] R. Khraishi and R. Okhrati. Simple Noisy Environment Augmentation for Reinforcement Learning, 2023.

[24] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A Survey of Zero-shot Generalisation in Deep Reinforcement Learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023. ISSN 1076-9757. doi: 10.1613/jair.1.14174.

[25] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

[26] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, A. Kembhavi, A. Gupta, and A. Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI, 2022.

[27] I. Kostrikov. JAXRL: Implementations of reinforcement learning algorithms in JAX, 2022.

[28] H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The NetHack Learning Environment. In *Advances in Neural Information Processing Systems*, volume 33, pages 7671–7684. Curran Associates, Inc., 2020.

[29] E. D. Langlois and T. Everitt. How RL agents behave when their actions are modified. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11586–11594, 2021. doi: 10.1609/aaai.v35i13.17378.

[30] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg. AI Safety Gridworlds, 2017.

[31] M. Liesner, W. Kirsch, and W. Kunde. The interplay of predictive and postdictive components of experienced selfhood. *Consciousness and Cognition*, 77:102850, 2020. ISSN 1053-8100. doi: 10.1016/j.concog.2019.102850.

[32] M. Liesner, W. Kirsch, R. Pfister, and W. Kunde. Spatial action–effect binding depends on type of action–effect transformation. *Attention, Perception, & Psychophysics*, 82(5):2531–2543, 2020. ISSN 1943-393X. doi: 10.3758/s13414-020-02013-2.

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR*, 2016.

[34] D. Lopez-Paz and M. A. Ranzato. Gradient Episodic Memory for Continual Learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[35] N. Lucchesi, A. Carta, V. Lomonaco, and D. Bacciu. Avalanche RL: A Continual Reinforcement Learning Library, 2022.

[36] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61(1):523–562, 2018. ISSN 1076-9757.

[37] J. A. Mendez, M. Hussing, M. Gummadi, and E. Eaton. CompoSuite: A Compositional Reinforcement Learning Benchmark, 2022.

[38] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, F. Hutter, and M. Lindauer. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. *Journal of Artificial Intelligence Research*, 74, 2022. ISSN 1076-9757. doi: 10.1613/jair.1.13596.

[39] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research, 2018.

[40] S. Powers, E. Xing, E. Kolve, R. Mottaghi, and A. Gupta. CORA: Benchmarks, Baselines, and Metrics as a Platform for Continual Reinforcement Learning Agents. In *Proceedings of The 1st Conference on Lifelong Learning Agents*, pages 705–743. PMLR, 2022.

[41] A. Raffin. RL baselines3 zoo, 2020.

[42] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. ISSN 1533-7928.

[43] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne. Experience Replay for Continual Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[44] M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Kuttler, E. Grefenstette, and T. Rocktäschel. MiniHack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[45] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & Compress: A scalable framework for continual learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

[46] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[47] S. Sodhani, A. Zhang, and J. Pineau. Multi-Task Reinforcement Learning with Context-based Representations, 2021.

[48] D. Starch. A demonstration of the trial and error method of learning. *Psychological Bulletin*, 7 (1):20–23, 1910. ISSN 1939-1455. doi: 10.1037/h0063796.

[49] A. Stone, O. Ramirez, K. Konolige, and R. Jonschkowski. The Distracting Control Suite – A Challenging Benchmark for Reinforcement Learning from Pixels, 2021.

[50] L. Sun, H. Zhang, W. Xu, and M. Tomizuka. PaCo: Parameter-Compositional Multi-Task Reinforcement Learning, 2022.

[51] J. J. Tai, M. Towers, and E. Tower. Shimmy: Gymnasium and PettingZoo wrappers for commonly used environments. Zenodo, 2023.

[52] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[53] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. de Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, 2023.

[54] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa. Dm$_c$ontrol: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: 10.1016/j.simpa.2020.100022.

[55] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical Review*, 36(5):823–841, 1930. doi: 10.1103/PhysRev.36.823.

[56] H. van Seijen, H. Nekoei, E. Racah, and S. Chandar. The LoCA regret: A consistent metric to evaluate model-based behavior in reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[57] P. Wawrzyński. A Cat-Like Robot Real-Time Learning to Run. In *Adaptive and Natural Computing Algorithms*, volume 5495, pages 380–390. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-04920-0 978-3-642-04921-7. doi: 10.1007/978-3-642-04921-7_39.

[58] M. Wołczyk, M. Zając, R. Pascanu, Ł. Kuciński, and P. Miłoś. Continual World: A Robotic Benchmark For Continual Reinforcement Learning, 2021.

[59] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images. 2019.

[60] D. Yarats, I. Kostrikov, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.

[61] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient Surgery for Multi-Task Learning, 2020.

[62] T. Yu, D. Quillen, Z. He, R. Julian, A. Narayan, H. Shively, A. Bellathur, K. Hausman, C. Finn, and S. Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning, 2021.

[63] L. Zintgraf, S. Schulze, C. Lu, L. Feng, M. Igl, K. Shiarlis, Y. Gal, K. Hofmann, and S. Whiteson. VariBAD: Variational Bayes-Adaptive Deep RL via Meta-Learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021. ISSN 1533-7928.

A More examples

A well known example for switched action dynamics is the backward bicycle, which refers to a modified bicycle that has its handlebars connected to the front wheel over a gear, so that turning the handlebars to the right makes the wheel turn to the left, and vice versa. This modification creates a counterintuitive steering mechanism where the bike responds opposite to the rider's expectations. Riding such a bike can be a challenging and it highlights how difficult it can be for individuals to adapt to new ways of thinking or doing things, even when presented with clear evidence of the need for change.

Note the difference to the mirror drawing or tracing task, a famous test in psychology for which participants have to draw or trace a figure (such as a shape or a pattern) on the drawing surface by looking only at the reflection of their hand in the mirror. The catch is that the visual-motor feedback is reversed due to the mirror, meaning that if the participant moves their hand to the right, the reflection in the mirror appears to move to the left on the drawing surface [48]. Though being closely related, in that case the action effect remains the same, instead only the observation is reversed.

B Extended background

B.1 Transfer Learning in RL

Transfer learning in the context of RL involves leveraging knowledge gained from one task to improve the learning or performance of a related but different task. This is especially relevant when the tasks share some underlying structure or features.

On a basic level one can distinguish two kinds of transfer: Positive transfer, where the knowledge gained on a source task enhances the learning or performance on the target task and negative transfer, where the transferred knowledge hinders performance on the target task. Obviously positive transfer is desired whilst negative transfer is a potential risk in transfer learning, and careful design is needed to mitigate it.

For our purpose it may also be beneficial to distinguish parallel and serial transfer. Parallel transfer implies the simultaneous transfer of knowledge or skills across multiple tasks. In an RL context, this might involve training on several tasks concurrently, with the expectation that knowledge gained in one task can benefit learning in others. This is the intention behind multi-task RL. The subtle difference in serial transfer is that future tasks are unknown, thus it refers to a sequential transfer of knowledge or skills. In the broad field of meta RL, this corresponds to transferring knowledge from a source task to an unknown target task. In the context of continual RL, serial transfer translates to the learning of a series of tasks one after the other. The scheduling of tasks is further specified in curriculum learning, where each task builds on the knowledge gained from previous ones, possibly with increasing difficulty or complexity.

In the serial transfer learning domain, specifically for the continual learning case one may further make a difference between forward and backward transfer. Forward transfer occurs when knowledge or skills acquired from learning on earlier tasks positively influence the learning or performance on future tasks. Backward transfer, on the other hand, occurs when knowledge or skills acquired from learning on later tasks positively influence the learning or performance on earlier tasks.

B.2 Multi-task RL

The objective of multi-task RL is to acquire a unified policy, denoted as $\pi(a|s, z)$, where z represents the task ID or an encoding thereof. This policy is designed to maximize the average expected return across all considered tasks. Task information can be communicated to the policy through various means, such as language or a one-hot task identification encoding, which is supplied in addition to the current state.

One seeks to understand how effectively the policy generalizes and performs across a spectrum of related tasks. Multi-task RL algorithms attempt to make use of shared knowledge and skills across these tasks, enhancing the efficiency of the learning process, i.e., learn a set of tasks more quickly and more proficiently than learning them independently. This can be considered a parallel transfer of knowledge. The constant access to all tasks is equivalent to ignoring non-stationarity, while continual

RL focuses on just that. Multi-task RL algorithms are commonly evaluated by considering their average performance across all training tasks, as opposed to meta RL, which utilizes separate test tasks for assessment.

B.3 Continual RL

Continual RL is a field of study dedicated to creating algorithms that can effectively adapt to changing environments, i.e., they are capable of handling non-stationarity. The goal is to develop agents that can progressively learn new skills and tackle unfamiliar tasks without neglecting what they have learned before. This ability to adapt continuously over extended periods is often referred to as lifelong learning or endless adaptation.

In the training process, a series of tasks is presented to the system. The transitions between these tasks may be seamless and unknown to the agent. When evaluating these systems, researchers are often interested in assessing the agent's ability to apply previously acquired knowledge to new tasks (forward transfer), as well as retaining knowledge when faced with new challenges (backward transfer), all while avoiding catastrophic forgetting. Forward and backward transfer can be considered as some kind of serial transfer as opposed to parallel transfer as in the multi-task setting.

C Action wrapper overview

Table 1: An overview of action effect wrappers including a brief description and motivation.

Wrapper	Description	Motivation
InvertAction	Inverts one/all dimension(s) of the continuous action.	backwards bicycle
ScaleAction	Scales one/all dimension(s) of the continuous action.	sim2real gap
OffsetAction	Adds an offset to one/all dimension(s) of the continuous action.	sim2real gap, systematic error
NoiseAction	Adds gaussian noise to one/all dimension(s) of the continuous action.	sim2real gap, random error
SineNoiseAction	Adds gaussian noise overlaid with a sine offset to one/all dimension(s) of the continuous action.	sim2real gap, random error with systematic component
ZeroAction	Randomly zeros one/all dimension(s) of the continuous action for a defined number of steps.	loose contact
RepeatAction	Randomly repeats one/all dimension(s) of the continuous/discrete action for a defined number of steps.	loose contact
SwapAction	Swaps one dimension of the continuous/discrete action with another (randomly picked) or shuffles all dimensions. Changed order of dimensions is kept.	interchanged cables

D Hyperparameters

We use implementations provided by Kostrikov [27] (JAXRL2) and Raffin et al. [42] (Stable-Baselines 3). We intended not to tune the hyperparamter and rather kept them the default values. An overview is provided in Table 2.

Additional remarks: For the `walker-walk-v0(pixel)` environment typically an action repeat value of 2 is used [17]. In DrQ, again for `walker-walk-v0(pixel)`, we use the standard choice in

JAXRL2 for the image encoder which is originated from D4PG [2]. For the reduction of the twinned critic we use the mean Q-value in DrQ and the minimum Q-value in SAC which is both the default configuration. For the goal-conditioned environments `FetchReach-v2` and `FetchPush-v2` we use SAC as the learner and use HER [1] to sample from the replay buffer. For HER we set the number of additional, virtual goals sampled per real goal to 4 which is again the default in Stable-Baselines 3. For the hidden dimensions of actor and critic we used feasible values in accordance to Raffin [41].

Table 2: Hyperparamters for the SAC base used in the experiments.

Parameter	**Value**
Buffer capacity	1000000
Batch size	256
Discount γ	0.99
Optimizer	Adam
Critic LR	0.0003
Actor LR	0.0003
Temperature LR	0.0003
Critic soft target update τ	0.005
Init temperature (SAC)	1.0
Init temperature (DrQ)	0.1
Hidden dims (general)	(256, 256)
Hidden dims (`FetchReach-v2`)	(64, 64)
Hidden dims (`FetchPush-v2`)	(256, 256, 256)

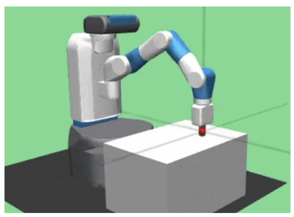
E Classification of the modifications

The purpose of the experiments in Section 5 and G is to illustrate the universal applicability of action modifications across varied environments, hence they are applicable for different problem statements. In some cases, the wrappers yield fundamentally different modifications. `NoiseAction` and `OffsetAction` might be considered comparable in their functionality, whereas `InvertAction` enables the investigation of a fundamentally different problem. They are intended to reveal the limits of the agent, in particular with regard to dimensions of robustness and transfer capabilities. It is not the sole aim to simulate realistic scenarios. For example it is hard to imagine a real-world scenario where the `SwapAction` wrapper seems appropriate. In Table 1 in the Appendix we provide the example of interchanged cables which is intended to elucidate the functionality but is arguably far-fetched. Still, in an idealized world it makes for an interesting case to investigate how the agent behaves under these circumstances.

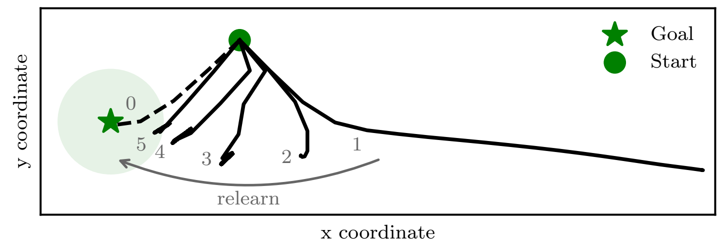
Based on the experimental results in Section 5 and Section G one may distinguish two main groups. First, `SwapAction` and `InvertAction`, although the modifications seem minimal, hardly any transfer is possible for the baseline agent. In the sequential setup it basically has to relearn from scratch (cf. Figure 5), possibly the return drops to the initial level (cf. Figure 2, respectively on the right). For the other modifications which may form the second group, namely `ScaleAction`, `OffsetAction`, `NoiseAction`, `SineNoiseAction`, `ZeroAction`, `RepeatAction`, transfer seems natural. Though, the experiments show that also with modifications from this group the adaptation can become arbitrary hard, depending on the configuration value and whether the modification is applied on only one dimension or all.

F Relearning in a new dynamic

Analogously to the one dynamic switch experiment in Section 5.1.3 we trained a SAC agent for $T_{\mathcal{T}} =$ 20000 steps to succeed on the `FetchReach-v2` environment (cf. Figure 5a). In that environment the agent is supposed to reach a sampled goal with a robotic arm. Then we modified the action dynamic using the `InvertAction` wrapper such that the action dimension which corresponds to the x coordinate is inverted. The agent trains for another $T_{\mathcal{T}} = 20000$ steps on the new dynamic. Figure 5b shows trajectories of the goal reaching robot arm before (stage 0) and after (stages 1-5) the dynamic switch. As expected, right after the switch (no relearning, stage 1) the trajectories are mirrored in the x coordinate, while the position on the y coordinate is intact. The agent seems to relearn the behavior corresponding to the x coordinate from scratch, respectively, it has to compensate for the mirroring which seems to gradually improve over the stages 2-5.



(a) `FetchReach-v2` environment.



(b) x-y trajectories.

Figure 5: Relearning behavior of a SAC agent in the `FetchReach-v2` environment. It is first trained on the default dynamic and then a switch is established using the `InvertAction` wrapper on the dimension 0 which corresponds to the x coordinate. Trajectories (in x-y coordinates) of the goal reaching robot arm before (stage 0, dashed trajectory) and after (stages 1-5, solid trajectory) the dynamic switch are shown.

G Extended results for experiments

In the Figures and Tables within this section we show extended results for a variety of task combinations, within the sequential and the multi-task setup. We present curves for training and evaluation returns. While the training curve only provides the overall view, evaluation returns for individual tasks allow for a more complete analysis, it allows for calculating the defined metrics in Section 5.

We consider five environments or variants of such: The first two are the goal-conditioned environments `FetchReach-v2` and `FetchPush-v2` initially developed by Plappert et al. [39] and currently maintained by de Lazcano et al. [9]. The third is `HalfCheetah-v4` based on work by Wawrzyński [57] and available through Towers et al. [53]. The fourth and fifth are `walker-walk-v0` from DM Control [54], utilized via Towers et al. [53] and Tai et al. [51], with, respectively, proprioceptive and visual observation inputs. We denote the latter as `walker-walk-v0(pixel)`.

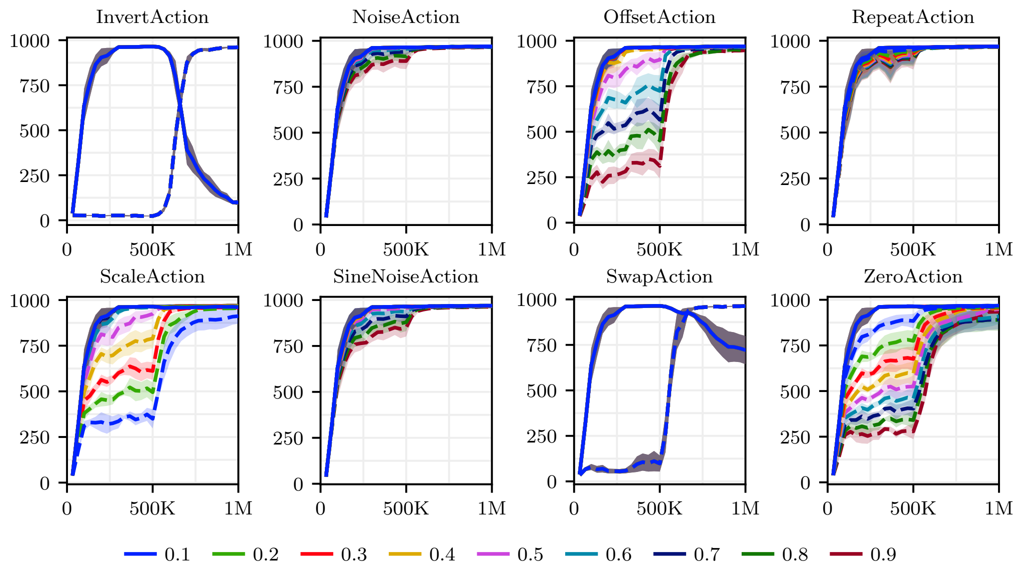
In the sequential training setup, for the environments `FetchReach-v2`, `FetchPush-v2`, `walker-walk-v0`, `walker-walk-v0(pixel)` and `HalfCheetah-v4` we train each task $\mathcal{T}_i$ for $T_{\mathcal{T}} = 100000$, $T_{\mathcal{T}} = 500000$, $T_{\mathcal{T}} = 500000$, $T_{\mathcal{T}} = 500000$ or $T_{\mathcal{T}} = 1000000$ steps, respectively. The choice of $T_{\mathcal{T}}$ is made to ensure that, at the very least, the unmodified environment can be trained to a reasonable degree. We only consider sequences of tasks where $T_{\mathcal{T}}$ is constant for each individual task.

In the multi-task training setup, for the environments `walker-walk-v0` and `HalfCheetah-v4` we train the agent for $T = 1000000$ and $T = 2000000$ steps, respectively. The steps eventually trained per task $\mathcal{T}_i$ are evenly distributed. In general we consider only a pair of tasks, hence $\{\mathcal{T}_1, \mathcal{T}_2\}$.

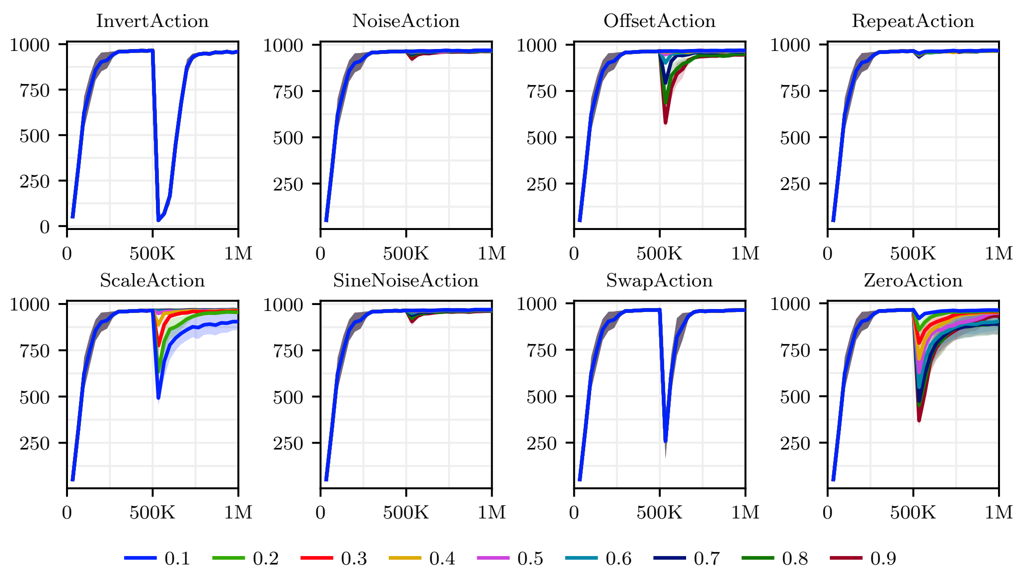
An overview for quicker access to all experiment variants is provided in Table 3.

Table 3: An overview of the extended experiments with references to the corresponding performance figures and metric tables.

Setting	Environment	Algorithm	Returns	Metrics
Sequential, one switch, dimension 0	`walker-walk-v0`	SAC	Figure 6	Table 4
Sequential, one switch, all dimensions	`walker-walk-v0`	SAC	Figure 7	Table 4
Sequential, one switch, dimension 0	`walker-walk-v0 (pixel)`	DrQ	Figure 8	Table 5
Sequential, one switch, all dimensions	`walker-walk-v0 (pixel)`	DrQ	Figure 9	Table 5
Sequential, one switch, dimension 0	`HalfCheetah-v4`	SAC	Figure 10	Table 6
Sequential, one switch, all dimensions	`HalfCheetah-v4`	SAC	Figure 11	Table 6
Sequential, one switch, dimension 0	`FetchReach-v2`	SAC+HER	Figure 12	Table 7
Sequential, one switch, all dimensions	`FetchReach-v2`	SAC+HER	Figure 13	Table 7
Sequential, one switch, dimension 0	`FetchPush-v2`	SAC+HER	Figure 14	Table 8
Sequential, one switch, all dimensions	`FetchPush-v2`	SAC+HER	Figure 15	Table 8
Sequential, one parallel switch, combine `ScaleAction`, all dimensions	`walker-walk-v0`	SAC	Figure 16	-
Sequential, one parallel switch, combine `OffsetAction`, all dimensions	`walker-walk-v0`	SAC	Figure 17	-
Sequential, continual adaptation, 10 tasks, dimension 0	`walker-walk-v0`	SAC	Figure 18	Table 9
Sequential, continual adaptation, 10 tasks, all dimensions	`walker-walk-v0`	SAC	Figure 19	Table 9
Sequential, continual adaptation, 10 tasks, dimension 0	`HalfCheetah-v4`	SAC	Figure 20	Table 10
Sequential, continual adaptation, 10 tasks, all dimensions	`HalfCheetah-v4`	SAC	Figure 21	Table 10
Sequential, continual adaptation, 500 tasks, dimension 0	`walker-walk-v0`	SAC	Figure 22	-
Sequential, continual adaptation, 500 tasks, all dimensions	`walker-walk-v0`	SAC	Figure 23	-
Multi-task, dimension 0	`walker-walk-v0`	SAC	Figure 24	Table 11
Multi-task, all dimensions	`walker-walk-v0`	SAC	Figure 25	Table 11
Multi-task, dimension 0	`HalfCheetah-v4`	SAC	Figure 26	Table 12
Multi-task, all dimensions	`HalfCheetah-v4`	SAC	Figure 27	Table 12

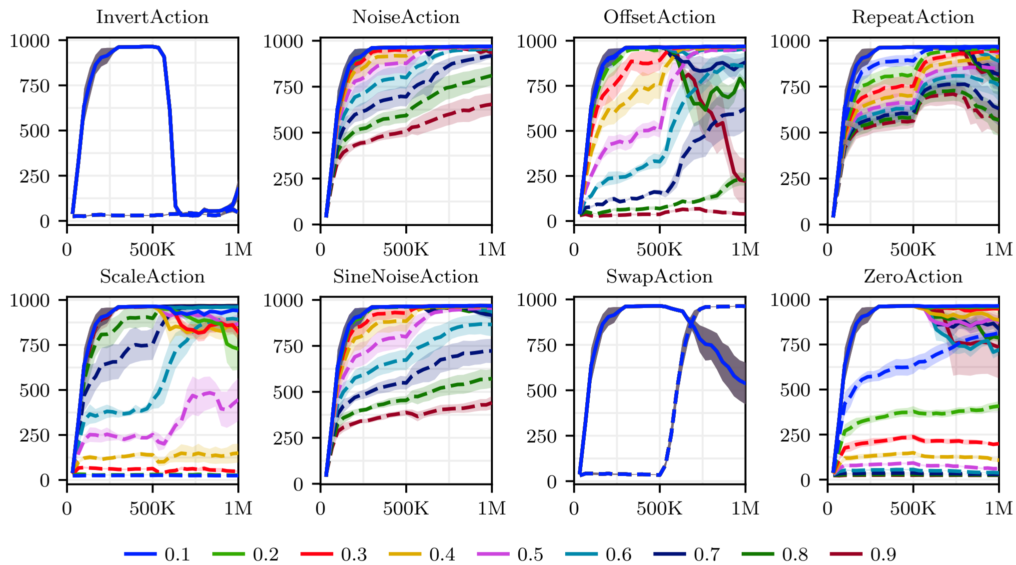


(a) Evaluation returns. Varied wrapper configuration value.

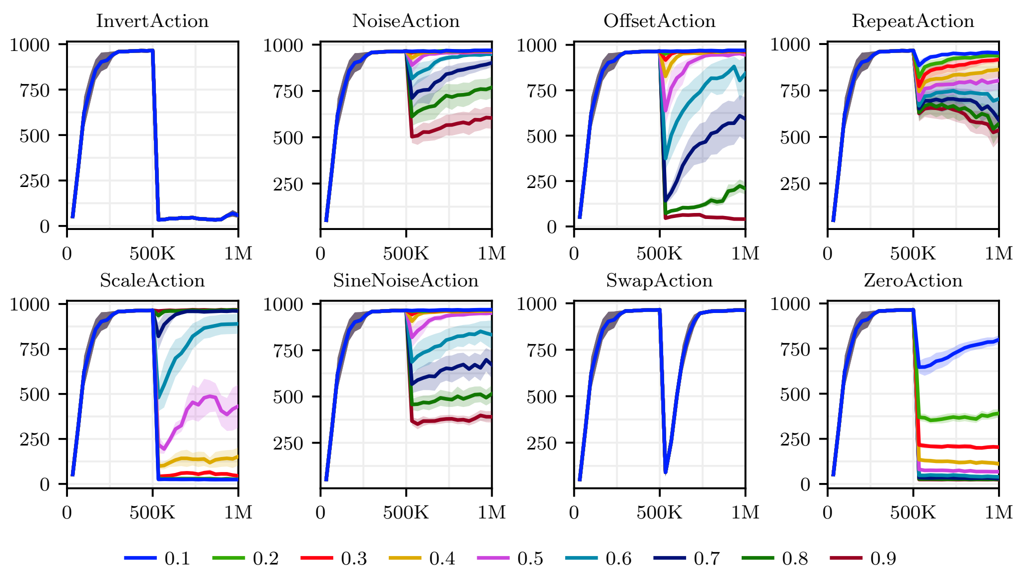


(b) Train returns. Varied wrapper configuration value.

Figure 6: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **action dimension 0**.

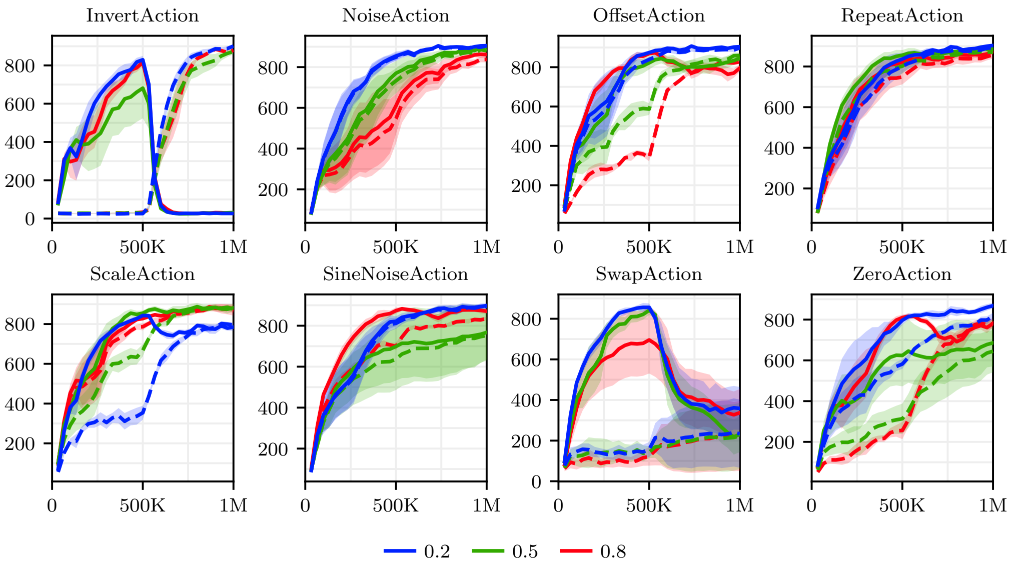


(a) Evaluation returns. Varied wrapper configuration value.

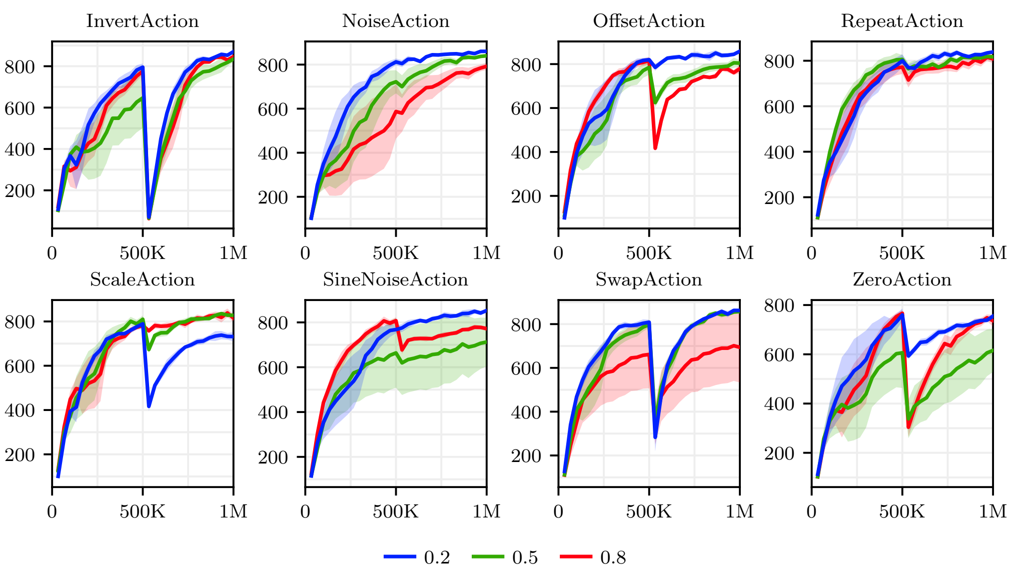


(b) Train returns. Varied wrapper configuration value.

Figure 7: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **all action dimensions**.

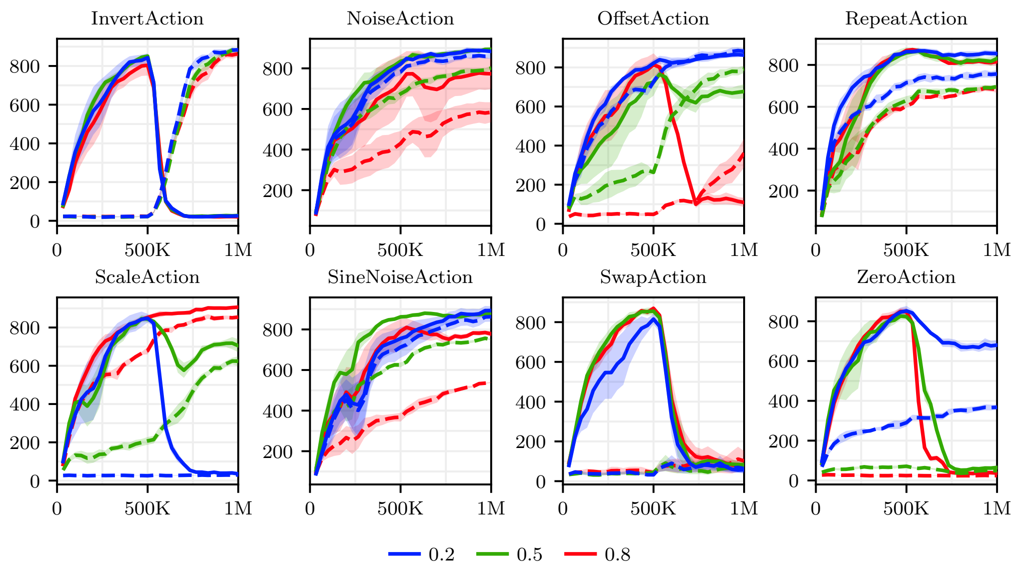


(a) Evaluation returns. Varied wrapper configuration value.

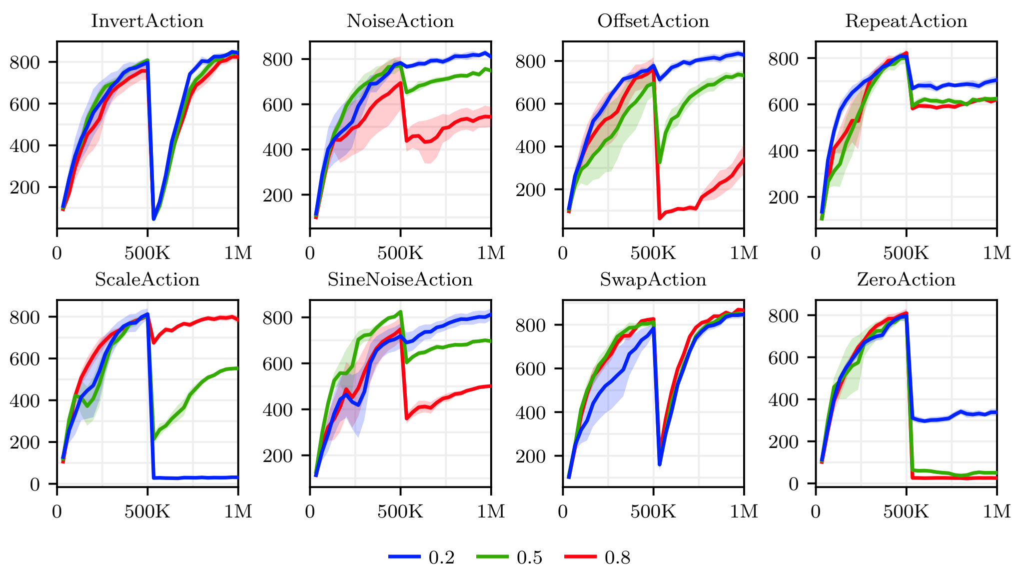


(b) Train returns. Varied wrapper configuration value.

Figure 8: Returns for the **sequential** training setup. A **DrQ** agent is trained on the `walker-walk-v0(pixel)` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **action dimension 0**.

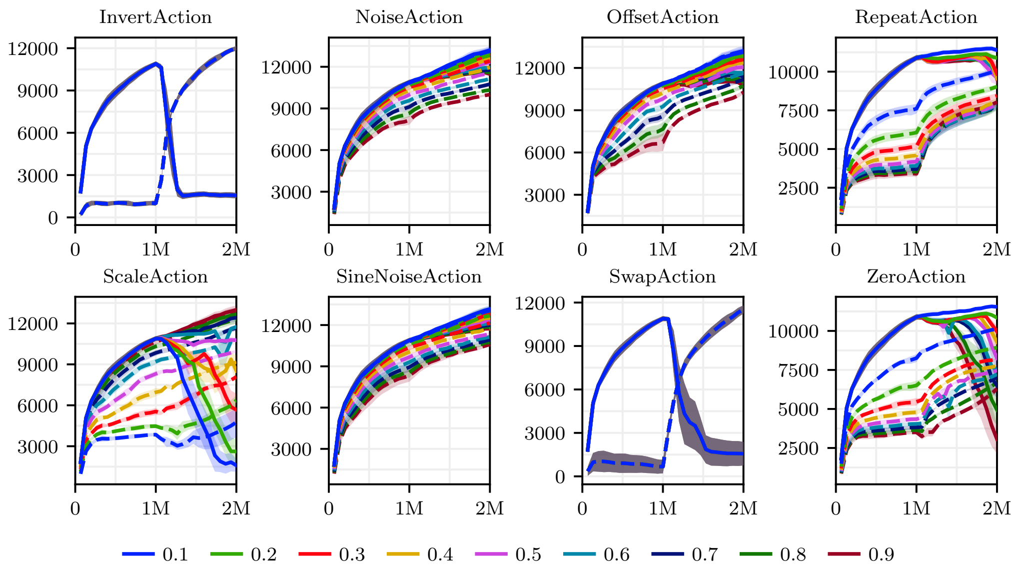


(a) Evaluation returns. Varied wrapper configuration value.

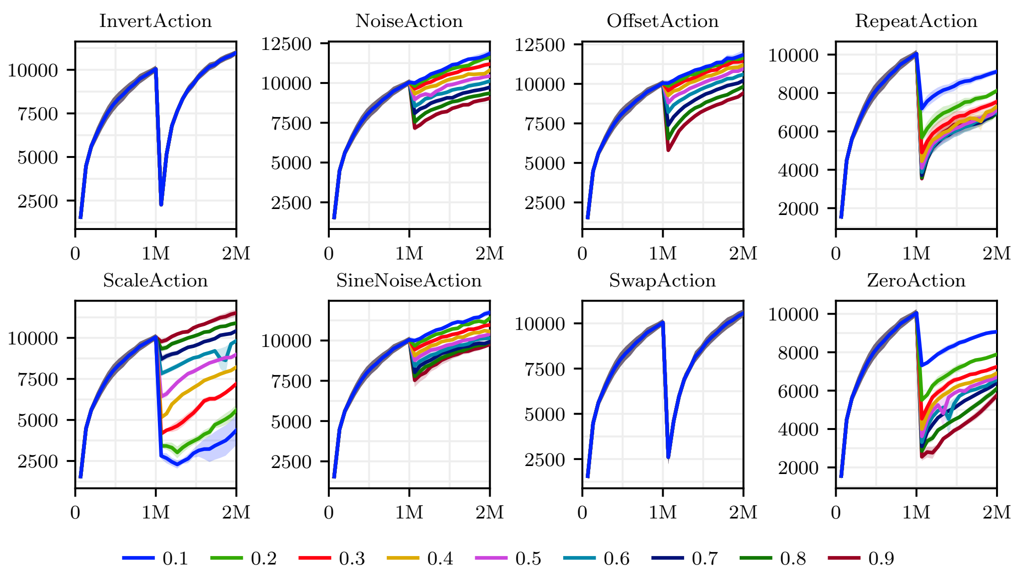


(b) Train returns. Varied wrapper configuration value.

Figure 9: Returns for the **sequential** training setup. A **DrQ** agent is trained on the `walker-walk-v0(pixel)` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **all action dimensions**.

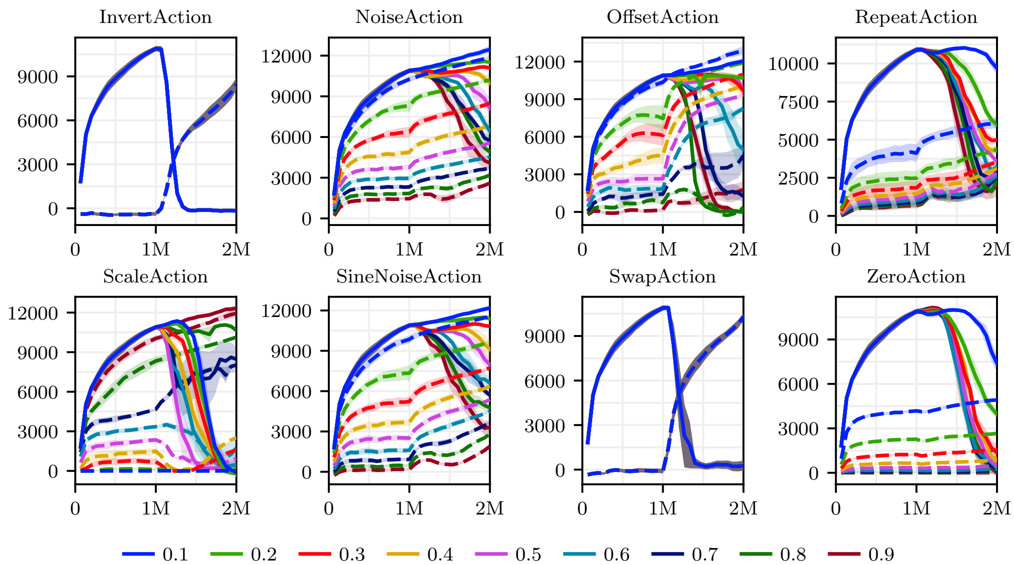


(a) Evaluation returns. Varied wrapper configuration value.

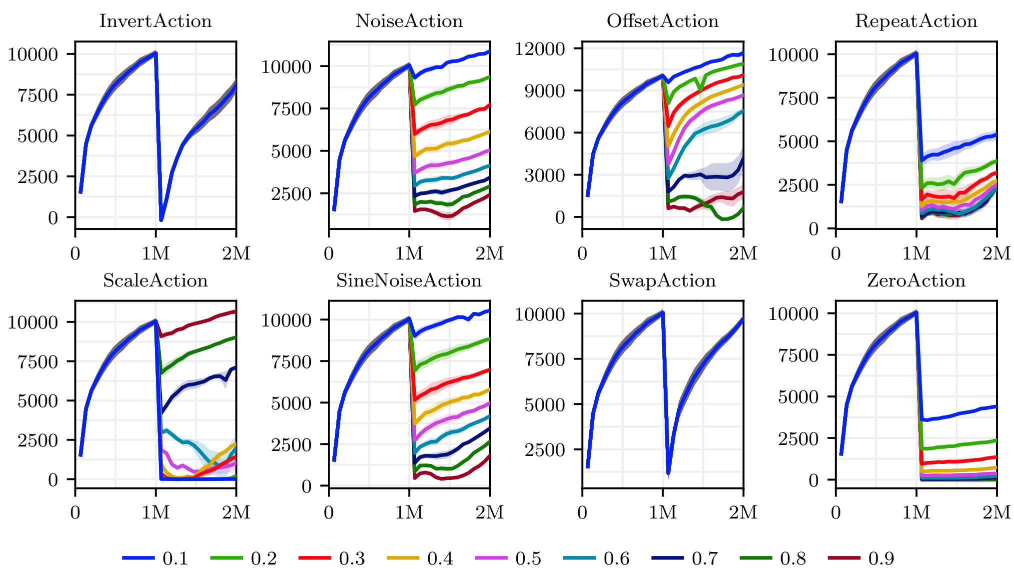


(b) Train returns. Varied wrapper configuration value.

Figure 10: Returns for the **sequential** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **action dimension 0**.



(a) Evaluation returns. Varied wrapper configuration value.



(b) Train returns. Varied wrapper configuration value.

Figure 11: Returns for the **sequential** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **all action dimensions**.

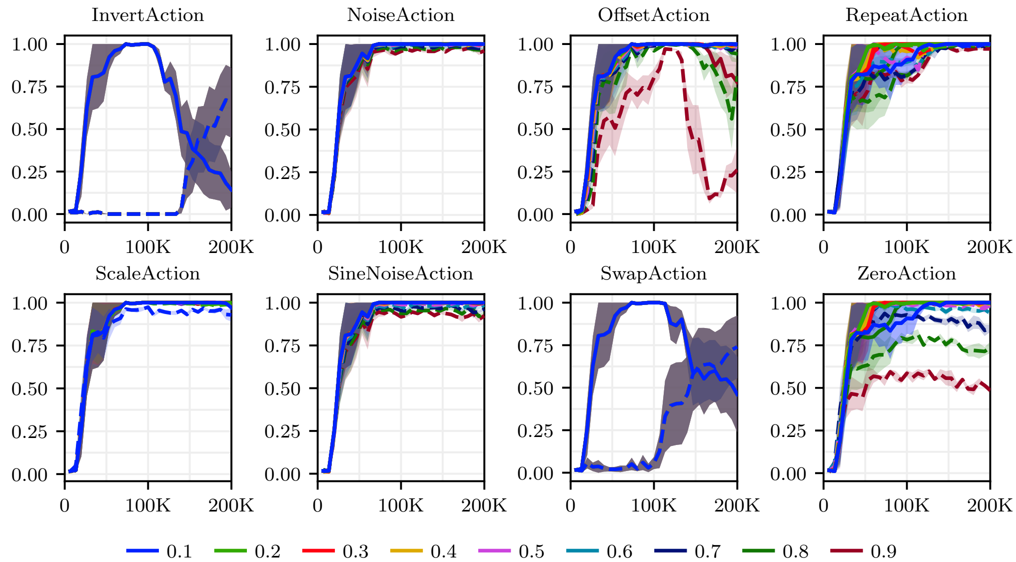


Figure 12: Evaluation success rates for the **sequential** training setup. A **SAC** agent is trained with **HER** on the goal-conditioned `FetchReach-v2` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **action dimension 0**.

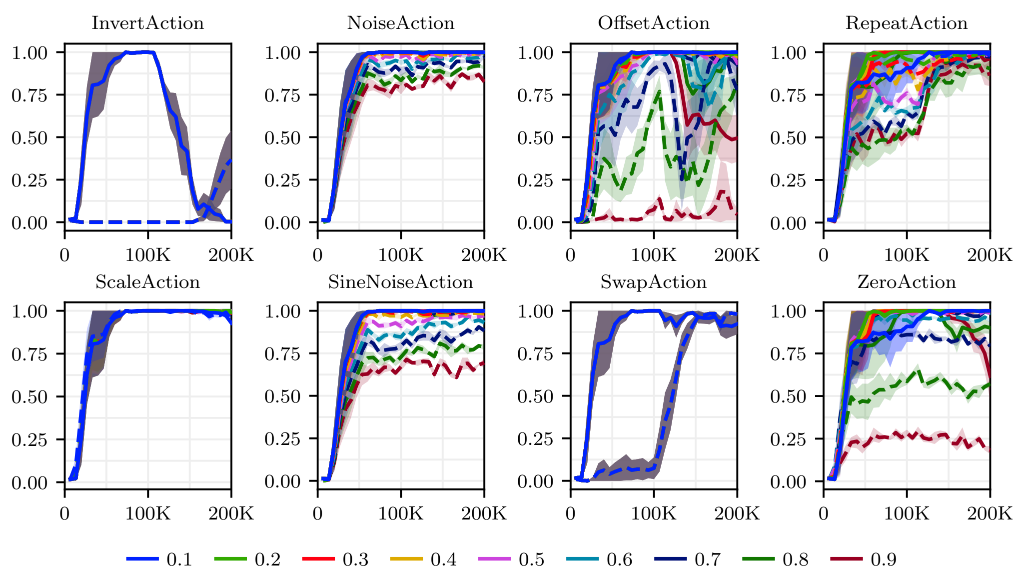


Figure 13: Evaluation success rates for the **sequential** training setup. A **SAC** agent is trained with **HER** on the goal-conditioned `FetchReach-v2` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **all action dimensions**.

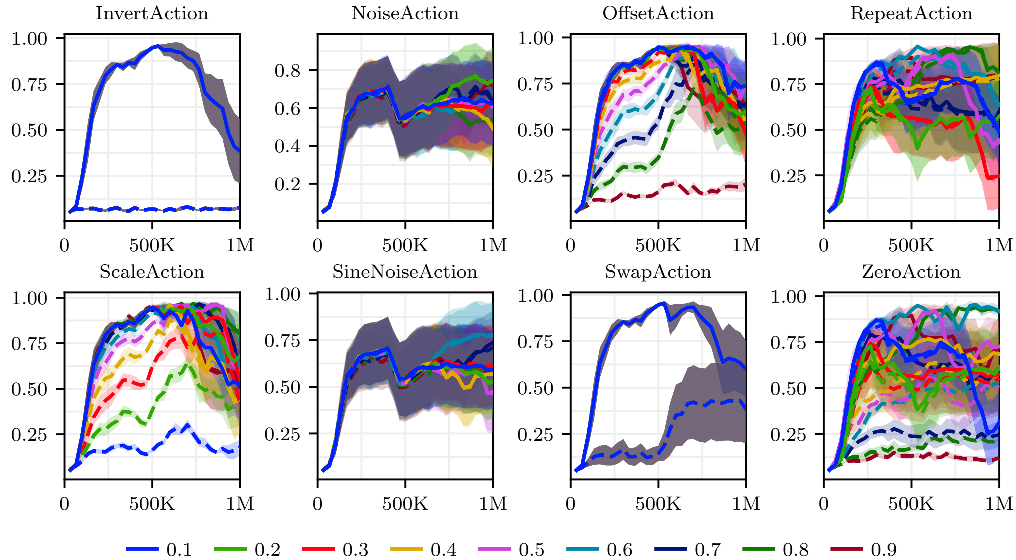


Figure 14: Evaluation success rates for the **sequential** training setup. A **SAC** agent is trained with **HER** on the goal-conditioned `FetchPush-v2` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **action dimension 0**.

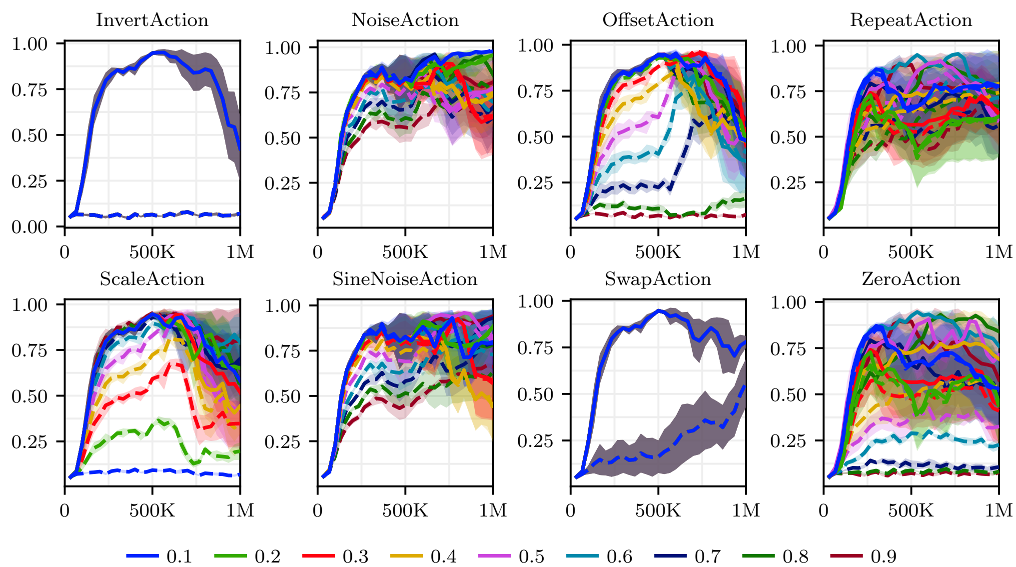
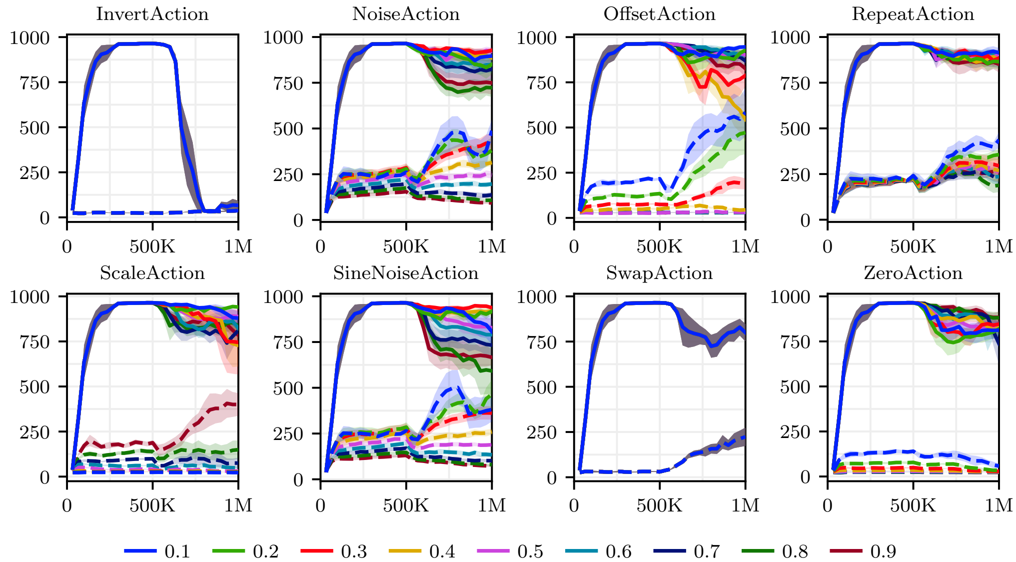
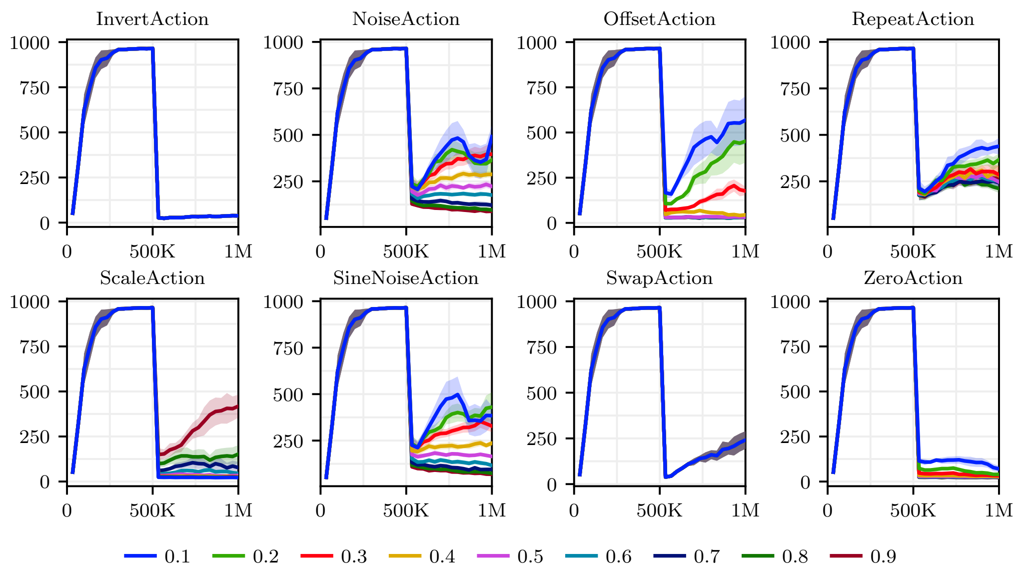


Figure 15: Evaluation success rates for the **sequential** training setup. A **SAC** agent is trained with **HER** on the goal-conditioned `FetchPush-v2` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **all action dimensions**.

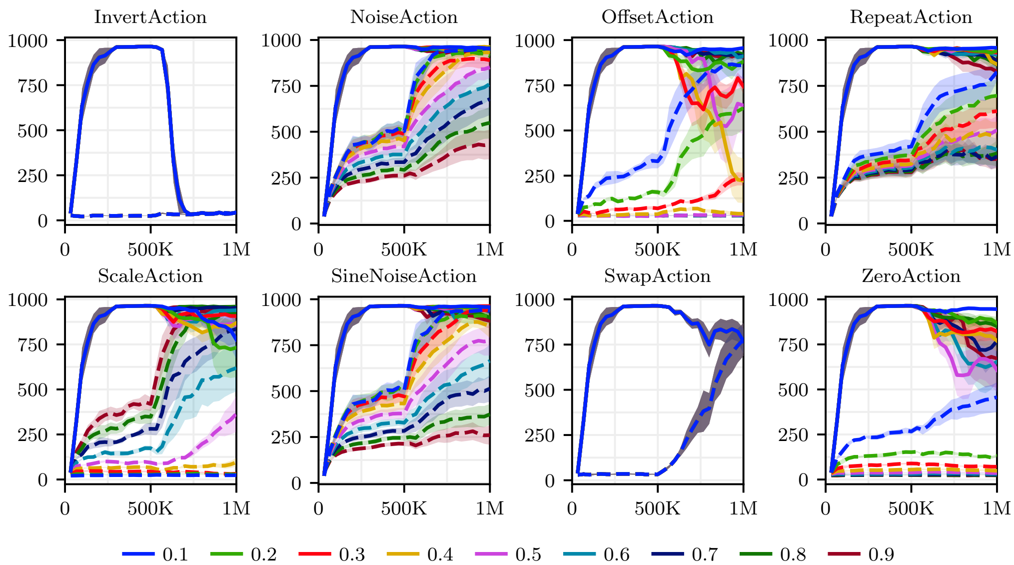


(a) Evaluation returns. Varied wrapper configuration value.

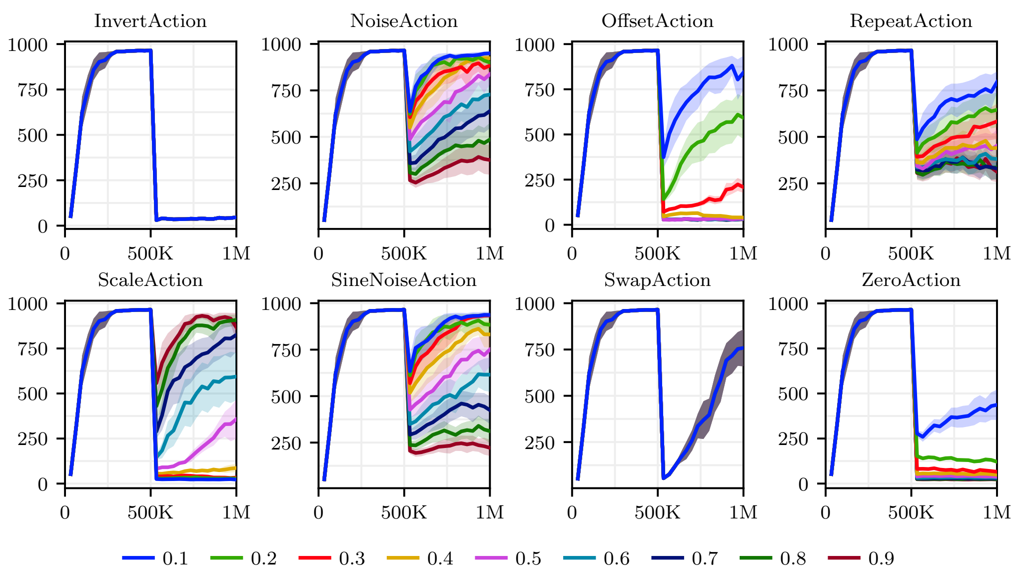


(b) Train returns. Varied wrapper configuration value.

Figure 16: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **One parallel dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. Two parallel modifications are applied. One modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. On top of that, the `ScaleAction` wrapper is applied with a scaling value of 0.5 representing a combination of changes. The modifications are done on **all action dimensions**.

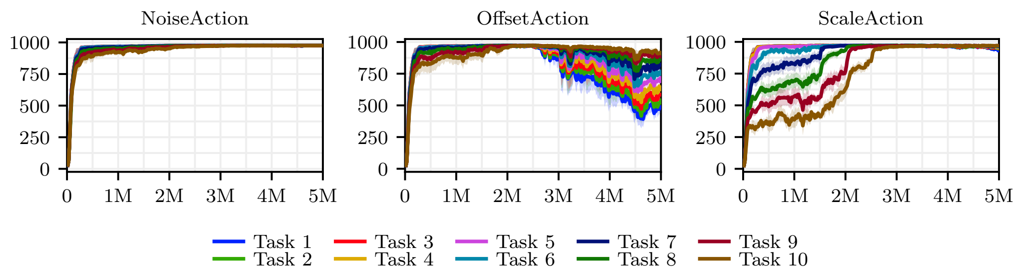


(a) Evaluation returns. Varied wrapper configuration value.

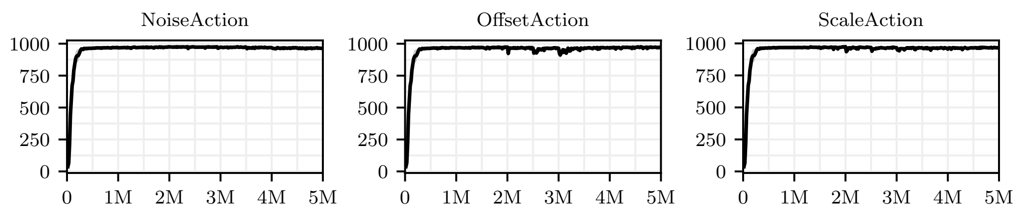


(b) Train returns. Varied wrapper configuration value.

Figure 17: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **One parallel dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. Two parallel modifications are applied. One modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. On top of that, the `OffsetAction` wrapper is applied with an offset value of 0.5 representing a combination of changes. The modifications are done on **all action dimensions**.

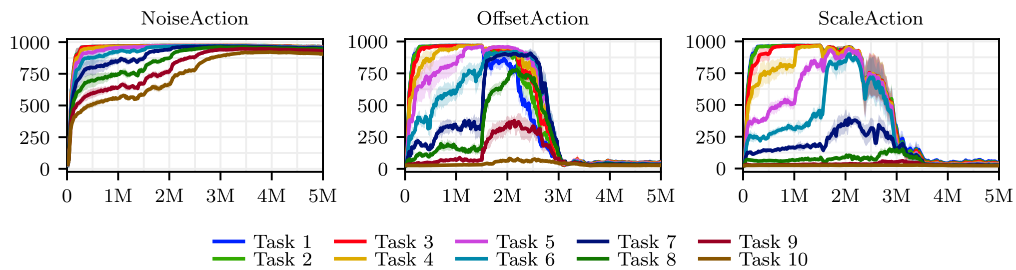


(a) Evaluation returns. Different tasks based on varied wrapper configuration values.

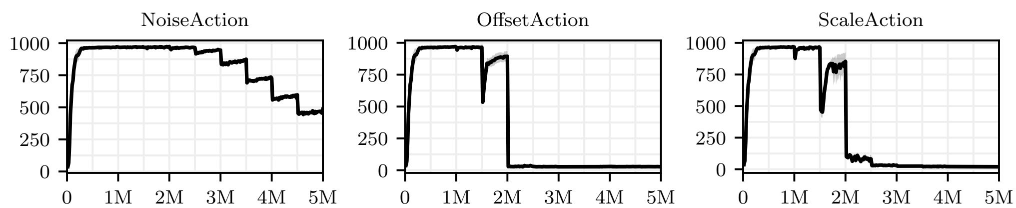


(b) Train returns.

Figure 18: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **Continual adaptation**: The agent trains for $T/10$ steps on the unmodified action dynamic, task $\mathcal{T}_1$. Then nine times for $T/10$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value, tasks in $\{\mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_{10}\}$. For `NoiseAction` tasks are determined by an action wrapper value in $\{0.1, 0.2, \ldots, 0.9\}$, for `OffsetAction` in $\{-0.1, -0.2, \ldots, -0.9\}$, for `ScaleAction` in $\{0.9, 0.8, \ldots, 0.1\}$. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the different modifications based on a varied wrapper value. The modification is done on **action dimension 0**.

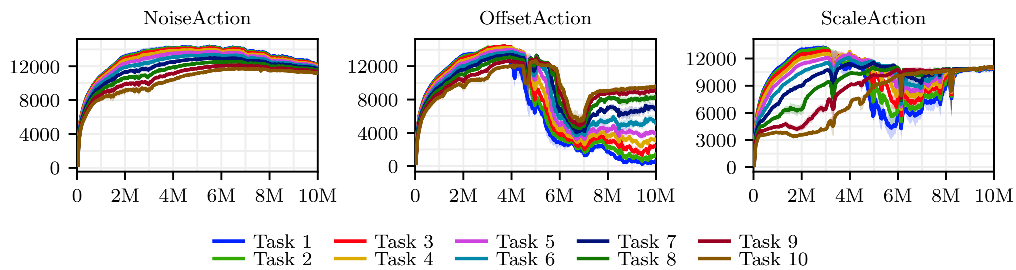


(a) Evaluation returns. Different tasks based on varied wrapper configuration values.

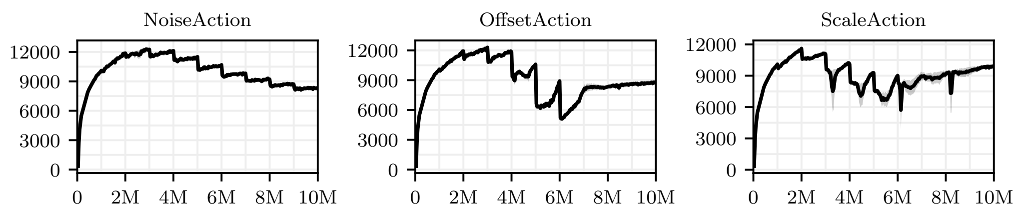


(b) Train returns.

Figure 19: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **Continual adaptation**: The agent trains for $T/10$ steps on the unmodified action dynamic, task $\mathcal{T}_1$. Then nine times for $T/10$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value, tasks in $\{\mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_{10}\}$. For `NoiseAction` tasks are determined by an action wrapper value in $\{0.1, 0.2, \ldots, 0.9\}$, for `OffsetAction` in $\{-0.1, -0.2, \ldots, -0.9\}$, for `ScaleAction` in $\{0.9, 0.8, \ldots, 0.1\}$. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the different modifications based on a varied wrapper value. The modification is done on **all action dimensions**.

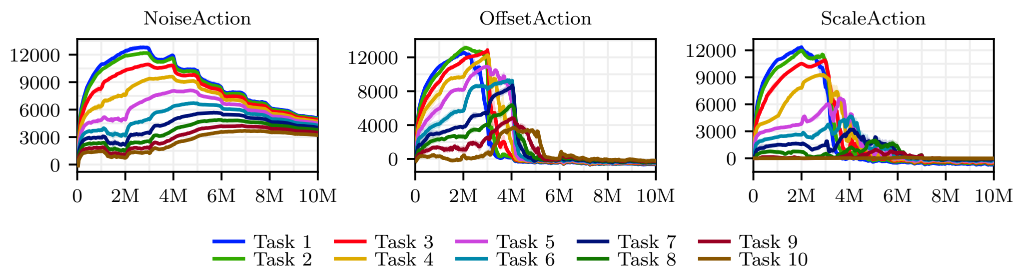


(a) Evaluation returns. Different tasks based on varied wrapper configuration values.

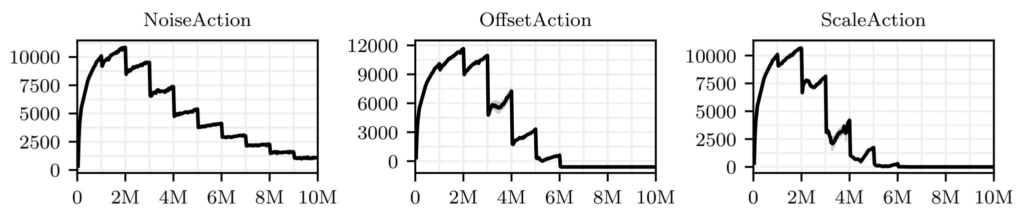


(b) Train returns.

Figure 20: Returns for the **sequential** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. **Continual adaptation**: The agent trains for $T/10$ steps on the unmodified action dynamic, task $\mathcal{T}_1$. Then nine times for $T/10$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value, tasks in $\{\mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_{10}\}$. For `NoiseAction` tasks are determined by an action wrapper value in $\{0.1, 0.2, \ldots, 0.9\}$, for `OffsetAction` in $\{-0.1, -0.2, \ldots, -0.9\}$, for `ScaleAction` in $\{0.9, 0.8, \ldots, 0.1\}$. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the different modifications based on a varied wrapper value. The modification is done on **action dimension 0**.

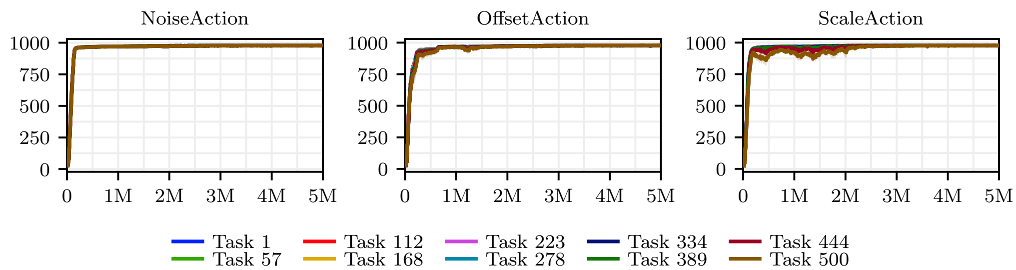


(a) Evaluation returns. Different tasks based on varied wrapper configuration values.

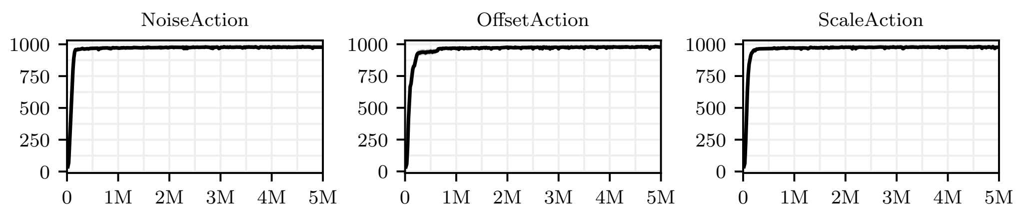


(b) Train returns.

Figure 21: Returns for the **sequential** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. **Continual adaptation**: The agent trains for $T/10$ steps on the unmodified action dynamic, task $\mathcal{T}_1$. Then nine times for $T/10$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value, tasks in $\{\mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_{10}\}$. For `NoiseAction` tasks are determined by an action wrapper value in $\{0.1, 0.2, \ldots, 0.9\}$, for `OffsetAction` in $\{-0.1, -0.2, \ldots, -0.9\}$, for `ScaleAction` in $\{0.9, 0.8, \ldots, 0.1\}$. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the different modifications based on a varied wrapper value. The modification is done on **all action dimensions**.

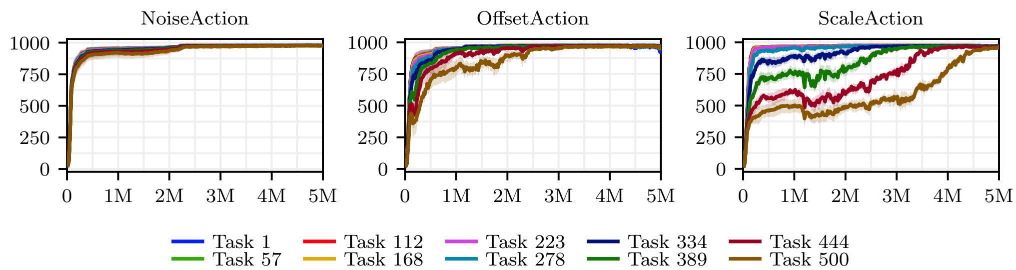


(a) Evaluation returns. Different tasks based on varied wrapper configuration values.

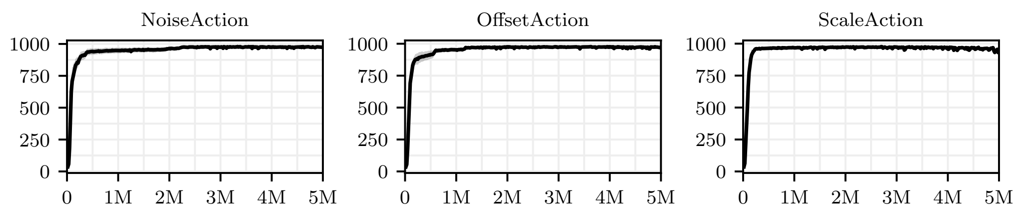


(b) Train returns.

Figure 22: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **Continual slowly adaptation**: The agent trains for $T/500$ steps on the unmodified action dynamic, task $\mathcal{T}_1$. Then 499 times for $T/500$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value, tasks in $\{\mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_{500}\}$. For `NoiseAction` tasks are determined by an action wrapper value in $\{0.0, \ldots, 0.9\}$, for `OffsetAction` in $\{0.0, \ldots, -0.9\}$, for `ScaleAction` in $\{1.0, \ldots, 0.1\}$. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the different modifications based on a varied wrapper value. Only ten tasks are shown for better clarity. The modification is done on **action dimension 0**.

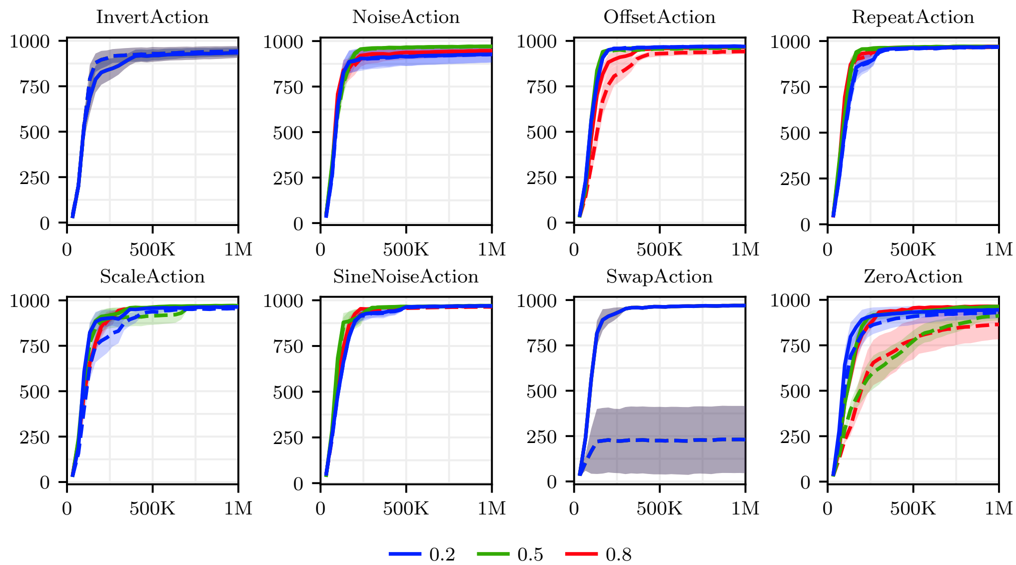


(a) Evaluation returns. Different tasks based on varied wrapper configuration values.

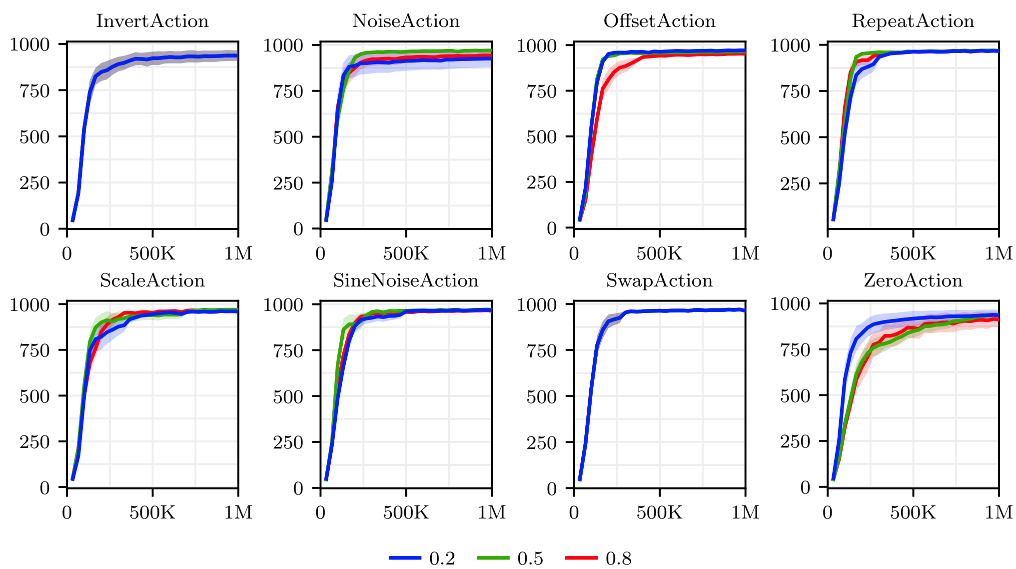


(b) Train returns.

Figure 23: Returns for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **Continual slowly adaptation**: The agent trains for $T/500$ steps on the unmodified action dynamic, task $\mathcal{T}_1$. Then 499 times for $T/500$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value, tasks in $\{\mathcal{T}_2, \mathcal{T}_3, \dots, \mathcal{T}_{500}\}$. For `NoiseAction` tasks are determined by an action wrapper value in $\{0.0, \dots, 0.9\}$, for `OffsetAction` in $\{0.0, \dots, -0.9\}$, for `ScaleAction` in $\{1.0, \dots, 0.1\}$. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the different modifications based on a varied wrapper value. Only ten tasks are shown for better clarity. The modification is done on **all action dimensions**.

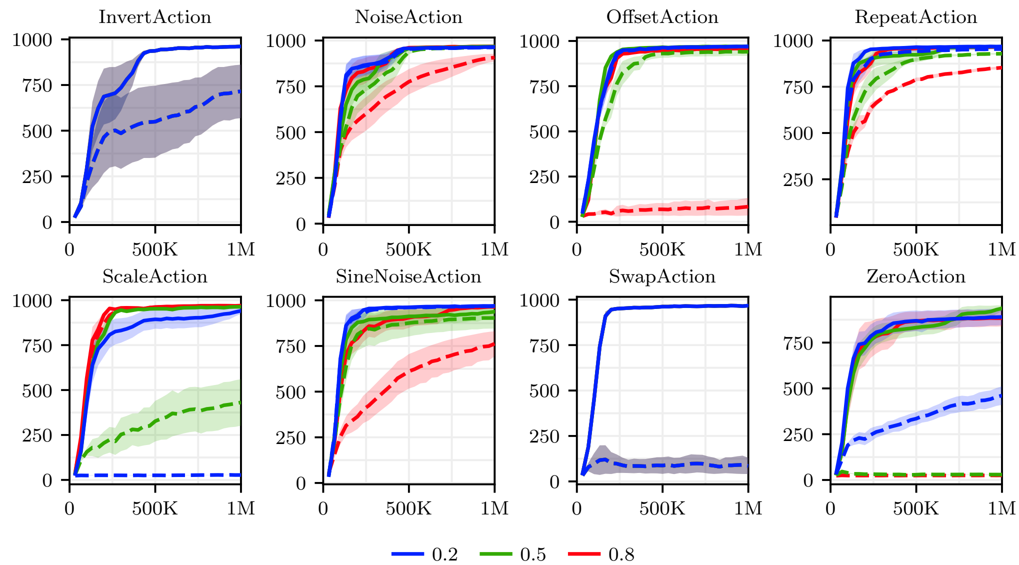


(a) Evaluation returns. Varied wrapper configuration value.

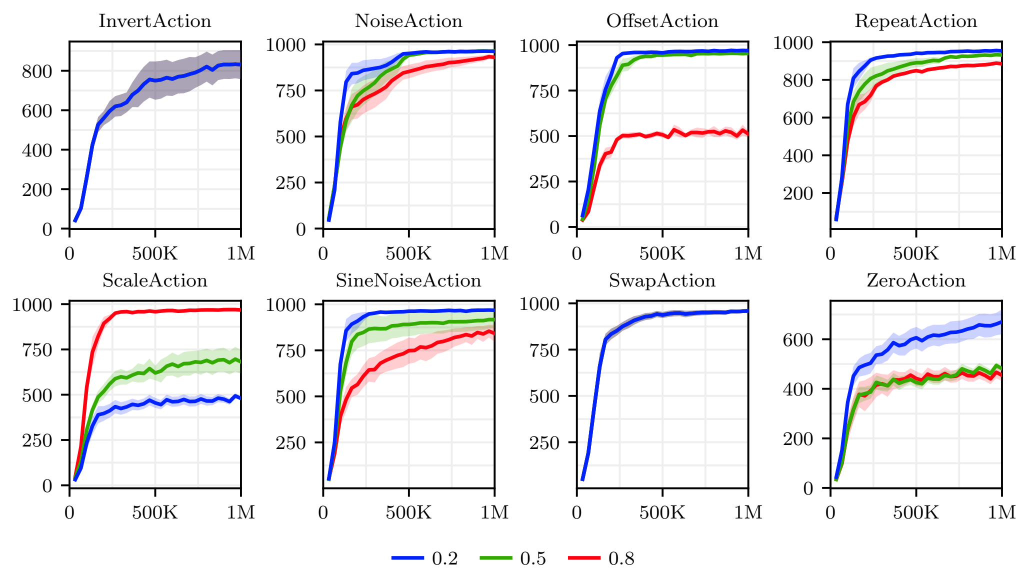


(b) Train returns. Varied wrapper configuration value.

Figure 24: Returns for the **multi-task** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. The agent trains on two tasks resulting from the unmodified action dynamic and one modification of such. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **action dimension 0**.

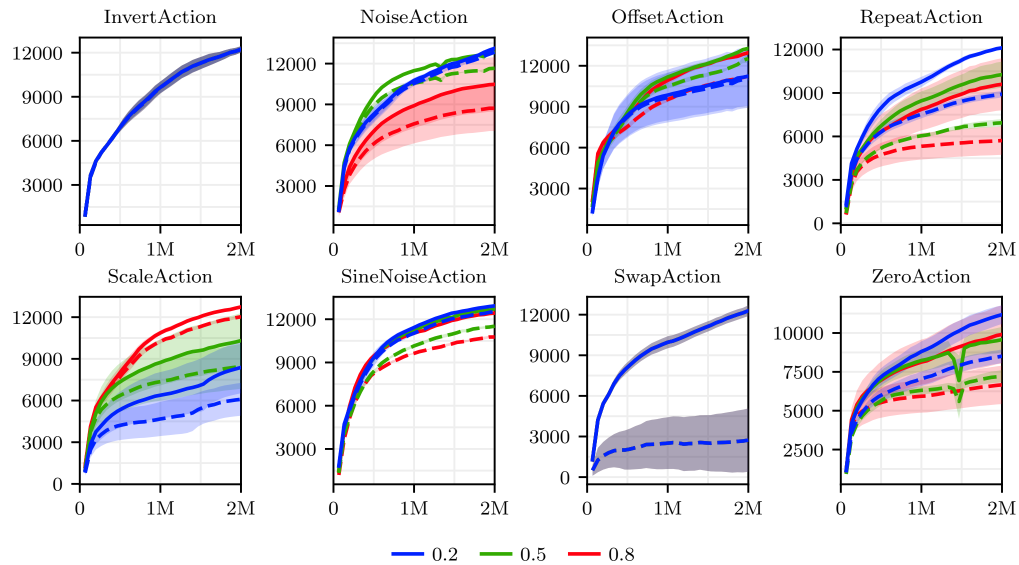


(a) Evaluation returns. Varied wrapper configuration value.

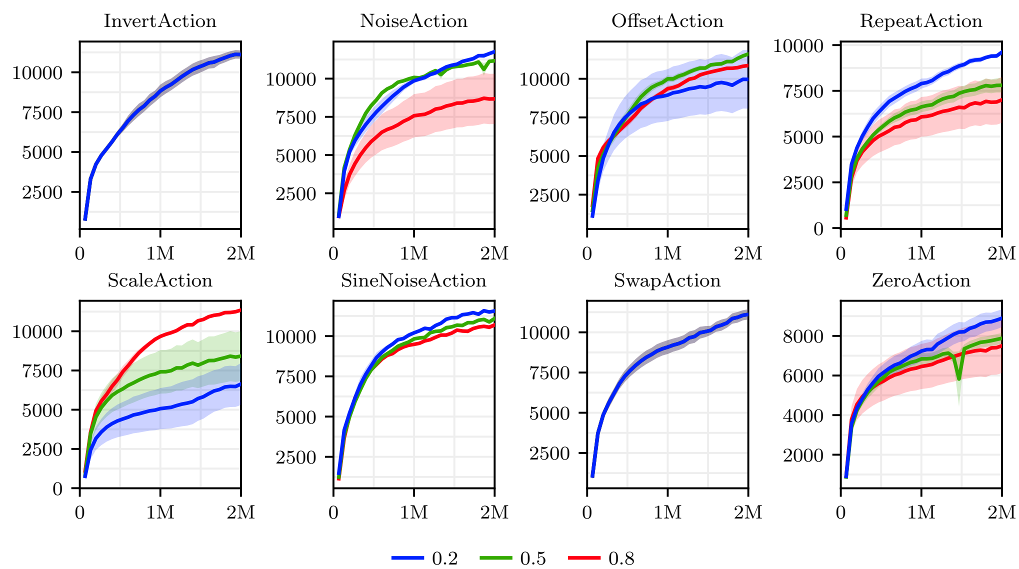


(b) Train returns. Varied wrapper configuration value.

Figure 25: Returns for the **multi-task** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. The agent trains on two tasks resulting from the unmodified action dynamic and one modification of such. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **all action dimensions**.

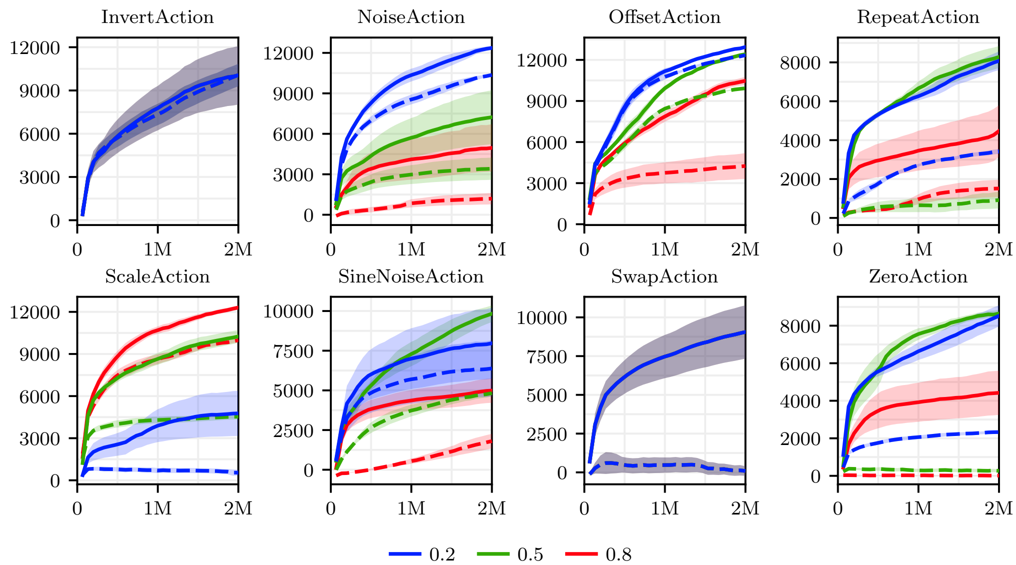


(a) Evaluation returns. Varied wrapper configuration value.

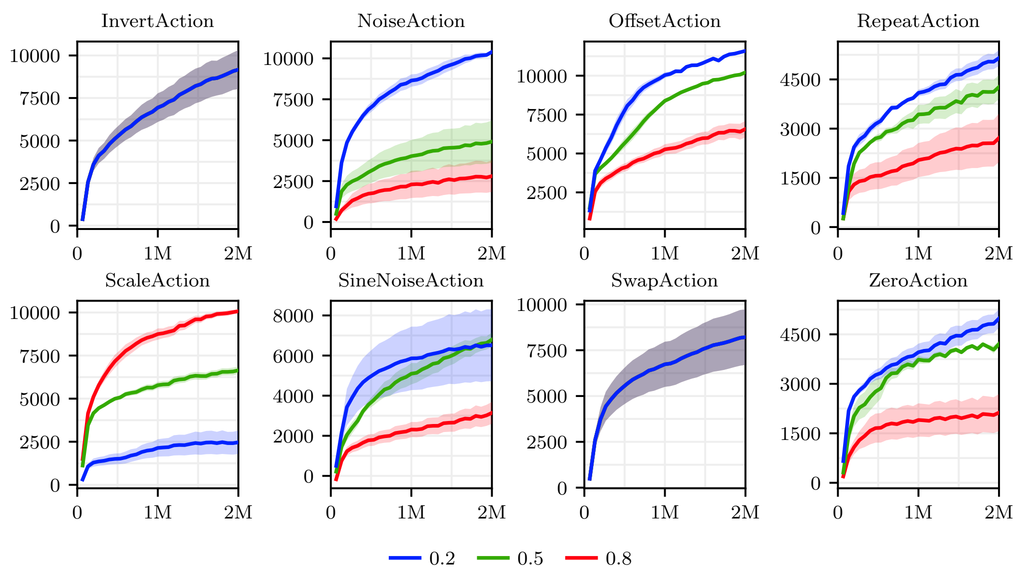


(b) Train returns. Varied wrapper configuration value.

Figure 26: Returns for the **multi-task** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. The agent trains on two tasks resulting from the unmodified action dynamic and one modification of such. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **action dimension 0**.



(a) Evaluation returns. Varied wrapper configuration value.



(b) Train returns. Varied wrapper configuration value.

Figure 27: Returns for the **multi-task** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. The agent trains on two tasks resulting from the unmodified action dynamic and one modification of such. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1. Colors represent the varied wrapper value. Solid lines represent the unmodified action dynamic, dashed lines the modified one. The modification is done on **all action dimensions**.

Table 4: Metrics for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	**Dim.**	**Value**	**Avg. perf.**	**Fwd. trans.**	**Bwd. trans.**	**Forgetting**
InvertAction	0	-	0.54 (0.01)	-0.01 (0.01)	0.0 (0.0)	0.92 (0.02)
	all	-	0.17 (0.06)	0.0 (0.0)	0.0 (0.0)	0.7 (0.12)
NoiseAction	0	0.2	1.0 (0.0)	0.96 (0.0)	0.01 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.96 (0.01)	0.01 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	0.92 (0.03)	0.01 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	0.96 (0.01)	0.01 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.85 (0.07)	0.01 (0.0)	0.0 (0.0)
		0.8	0.92 (0.02)	0.57 (0.04)	0.0 (0.0)	0.01 (0.01)
OffsetAction	0	0.2	1.0 (0.0)	0.96 (0.0)	0.01 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.95 (0.01)	0.01 (0.0)	0.0 (0.0)
		0.8	0.99 (0.0)	0.45 (0.08)	0.0 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	0.96 (0.01)	0.01 (0.0)	0.0 (0.0)
		0.5	0.98 (0.01)	0.48 (0.05)	0.0 (0.0)	0.0 (0.0)
		0.8	0.45 (0.04)	0.05 (0.02)	0.0 (0.0)	0.32 (0.07)
RepeatAction	0	0.2	1.0 (0.0)	0.94 (0.02)	0.01 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.91 (0.04)	0.01 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	0.92 (0.03)	0.01 (0.0)	0.0 (0.0)
	all	0.2	0.99 (0.0)	0.8 (0.06)	0.01 (0.0)	0.0 (0.0)
		0.5	0.95 (0.02)	0.67 (0.04)	0.0 (0.0)	0.0 (0.0)
		0.8	0.74 (0.08)	0.58 (0.04)	0.0 (0.0)	0.12 (0.06)
ScaleAction	0	0.2	1.0 (0.0)	0.51 (0.05)	0.0 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.9 (0.05)	0.01 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	0.96 (0.0)	0.01 (0.0)	0.0 (0.0)
	all	0.2	0.37 (0.05)	0.01 (0.0)	0.0 (0.0)	0.29 (0.1)
		0.5	0.72 (0.06)	0.26 (0.05)	0.0 (0.0)	0.08 (0.04)
		0.8	1.0 (0.0)	0.88 (0.08)	0.01 (0.0)	0.0 (0.0)
SineNoiseAction	0	0.2	1.0 (0.0)	0.96 (0.01)	0.01 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.95 (0.01)	0.01 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	0.87 (0.05)	0.01 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	0.95 (0.01)	0.01 (0.0)	0.0 (0.0)
		0.5	0.99 (0.0)	0.77 (0.07)	0.01 (0.0)	0.0 (0.0)
		0.8	0.81 (0.03)	0.44 (0.04)	0.0 (0.0)	0.0 (0.0)
SwapAction	0	-	0.88 (0.04)	0.06 (0.04)	0.0 (0.0)	0.24 (0.08)
	all	-	0.77 (0.06)	0.0 (0.0)	0.0 (0.0)	0.46 (0.12)
ZeroAction	0	0.2	1.0 (0.0)	0.78 (0.06)	0.01 (0.0)	0.0 (0.0)
		0.5	0.99 (0.0)	0.52 (0.05)	0.01 (0.0)	0.0 (0.0)
		0.8	0.96 (0.03)	0.34 (0.05)	0.0 (0.0)	0.01 (0.01)
	all	0.2	0.71 (0.01)	0.36 (0.03)	0.0 (0.0)	0.0 (0.0)
		0.5	0.5 (0.03)	0.07 (0.01)	0.0 (0.0)	0.06 (0.04)
		0.8	0.44 (0.02)	0.01 (0.0)	0.0 (0.0)	0.14 (0.03)

Table 5: Metrics for the **sequential** training setup. A **DrQ** agent is trained on the `walker-walk-v0(pixel)` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	**Dim.**	**Value**	**Avg. perf.**	**Fwd. trans.**	**Bwd. trans.**	**Forgetting**
InvertAction	0	-	0.52 (0.01)	-0.0 (0.0)	0.0 (0.0)	0.73 (0.18)
	all	-	0.51 (0.01)	-0.0 (0.0)	0.0 (0.0)	0.92 (0.02)
NoiseAction	0	0.2	1.0 (0.01)	0.9 (0.02)	0.09 (0.04)	0.01 (0.01)
		0.5	0.99 (0.01)	0.78 (0.07)	0.13 (0.07)	0.0 (0.0)
		0.8	0.98 (0.01)	0.67 (0.13)	0.24 (0.13)	0.01 (0.01)
	all	0.2	0.98 (0.02)	0.9 (0.02)	0.06 (0.02)	0.01 (0.01)
		0.5	0.94 (0.02)	0.75 (0.03)	0.04 (0.02)	0.0 (0.0)
		0.8	0.88 (0.08)	0.52 (0.06)	0.06 (0.06)	0.05 (0.02)
OffsetAction	0	0.2	0.99 (0.02)	0.89 (0.02)	0.05 (0.03)	0.02 (0.01)
		0.5	0.98 (0.03)	0.67 (0.03)	0.04 (0.02)	0.05 (0.03)
		0.8	0.95 (0.02)	0.38 (0.02)	0.0 (0.0)	0.08 (0.03)
	all	0.2	0.99 (0.01)	0.8 (0.05)	0.04 (0.02)	0.0 (0.0)
		0.5	0.91 (0.03)	0.24 (0.05)	0.06 (0.04)	0.11 (0.06)
		0.8	0.29 (0.06)	0.03 (0.01)	0.0 (0.0)	0.88 (0.05)
RepeatAction	0	0.2	0.99 (0.01)	0.87 (0.04)	0.08 (0.02)	0.0 (0.0)
		0.5	0.98 (0.01)	0.89 (0.03)	0.06 (0.02)	0.0 (0.0)
		0.8	1.0 (0.02)	0.86 (0.05)	0.05 (0.03)	0.01 (0.01)
	all	0.2	0.94 (0.02)	0.77 (0.02)	0.02 (0.01)	0.0 (0.0)
		0.5	0.86 (0.02)	0.71 (0.03)	0.0 (0.0)	0.07 (0.02)
		0.8	0.85 (0.03)	0.67 (0.02)	0.0 (0.0)	0.09 (0.03)
ScaleAction	0	0.2	0.94 (0.03)	0.43 (0.06)	0.02 (0.01)	0.08 (0.04)
		0.5	0.99 (0.01)	0.77 (0.02)	0.03 (0.02)	0.03 (0.02)
		0.8	0.99 (0.03)	0.87 (0.02)	0.06 (0.02)	0.03 (0.03)
	all	0.2	0.04 (0.0)	0.0 (0.0)	0.0 (0.0)	0.96 (0.02)
		0.5	0.77 (0.03)	0.2 (0.03)	0.0 (0.0)	0.2 (0.04)
		0.8	0.98 (0.02)	0.74 (0.04)	0.07 (0.04)	0.02 (0.02)
SineNoiseAction	0	0.2	0.99 (0.02)	0.86 (0.03)	0.07 (0.02)	0.0 (0.0)
		0.5	1.0 (0.17)	0.82 (0.21)	0.08 (0.04)	0.0 (0.0)
		0.8	0.98 (0.01)	0.72 (0.03)	0.03 (0.02)	0.01 (0.01)
	all	0.2	1.0 (0.03)	0.8 (0.07)	0.14 (0.07)	0.0 (0.0)
		0.5	0.93 (0.01)	0.62 (0.03)	0.03 (0.02)	0.0 (0.0)
		0.8	0.81 (0.02)	0.41 (0.03)	0.02 (0.02)	0.06 (0.02)
SwapAction	0	-	0.26 (0.13)	0.14 (0.02)	0.0 (0.0)	0.74 (0.08)
	all	-	0.1 (0.03)	0.03 (0.02)	0.0 (0.0)	0.89 (0.05)
ZeroAction	0	0.2	0.97 (0.02)	0.62 (0.04)	0.08 (0.03)	0.0 (0.0)
		0.5	0.96 (0.13)	0.4 (0.1)	0.14 (0.09)	0.05 (0.03)
		0.8	0.96 (0.02)	0.27 (0.06)	0.01 (0.0)	0.04 (0.02)
	all	0.2	0.63 (0.02)	0.3 (0.04)	0.0 (0.0)	0.18 (0.04)
		0.5	0.07 (0.01)	0.06 (0.01)	0.0 (0.0)	0.92 (0.03)
		0.8	0.04 (0.0)	0.0 (0.0)	0.0 (0.0)	0.96 (0.03)

Table 6: Metrics for the **sequential** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	Dim.	Value	Avg. perf.	Fwd. trans.	Bwd. transfer	Forgetting
InvertAction	0	-	0.56 (0.01)	0.09 (0.01)	0.0 (0.0)	0.79 (0.02)
	all	-	0.4 (0.02)	0.01 (0.02)	0.0 (0.0)	1.02 (0.01)
NoiseAction	0	0.2	0.99 (0.02)	0.84 (0.02)	0.16 (0.02)	0.0 (0.0)
		0.5	0.98 (0.01)	0.83 (0.03)	0.09 (0.01)	0.0 (0.0)
		0.8	0.95 (0.01)	0.75 (0.02)	0.06 (0.02)	0.01 (0.01)
	all	0.2	0.95 (0.01)	0.72 (0.04)	0.03 (0.02)	0.0 (0.0)
		0.5	0.61 (0.01)	0.34 (0.03)	0.0 (0.0)	0.29 (0.03)
		0.8	0.39 (0.01)	0.2 (0.01)	0.0 (0.0)	0.51 (0.02)
OffsetAction	0	0.2	1.0 (0.03)	0.85 (0.02)	0.14 (0.02)	0.0 (0.0)
		0.5	1.0 (0.03)	0.81 (0.02)	0.09 (0.03)	0.0 (0.0)
		0.8	0.98 (0.01)	0.69 (0.04)	0.01 (0.01)	0.02 (0.01)
	all	0.2	0.94 (0.02)	0.62 (0.09)	0.0 (0.0)	0.04 (0.01)
		0.5	0.82 (0.04)	0.26 (0.04)	0.0 (0.0)	0.22 (0.06)
		0.8	0.01 (0.01)	0.09 (0.02)	0.0 (0.0)	0.97 (0.02)
RepeatAction	0	0.2	0.9 (0.02)	0.56 (0.03)	0.0 (0.0)	0.02 (0.0)
		0.5	0.8 (0.02)	0.39 (0.02)	0.0 (0.0)	0.12 (0.03)
		0.8	0.77 (0.02)	0.33 (0.02)	0.0 (0.0)	0.16 (0.02)
	all	0.2	0.51 (0.02)	0.25 (0.03)	0.0 (0.0)	0.39 (0.03)
		0.5	0.32 (0.01)	0.12 (0.03)	0.0 (0.0)	0.63 (0.02)
		0.8	0.25 (0.02)	0.08 (0.02)	0.0 (0.0)	0.75 (0.08)
ScaleAction	0	0.2	0.47 (0.06)	0.4 (0.01)	0.0 (0.0)	0.66 (0.11)
		0.5	0.94 (0.02)	0.74 (0.02)	0.02 (0.01)	0.04 (0.02)
		0.8	0.98 (0.02)	0.83 (0.02)	0.12 (0.02)	0.0 (0.0)
	all	0.2	0.0 (0.0)	0.01 (0.01)	0.0 (0.0)	1.03 (0.02)
		0.5	0.01 (0.03)	0.21 (0.01)	0.0 (0.0)	1.01 (0.02)
		0.8	0.95 (0.01)	0.76 (0.02)	0.01 (0.01)	0.02 (0.01)
SineNoiseAction	0	0.2	0.99 (0.02)	0.84 (0.02)	0.15 (0.02)	0.0 (0.0)
		0.5	0.95 (0.02)	0.8 (0.02)	0.12 (0.03)	0.0 (0.0)
		0.8	0.94 (0.02)	0.76 (0.04)	0.06 (0.02)	0.0 (0.0)
	all	0.2	0.94 (0.01)	0.65 (0.06)	0.01 (0.0)	0.0 (0.0)
		0.5	0.62 (0.03)	0.24 (0.02)	0.0 (0.0)	0.26 (0.07)
		0.8	0.37 (0.01)	0.08 (0.01)	0.0 (0.0)	0.52 (0.02)
SwapAction	0	-	0.57 (0.03)	0.07 (0.03)	0.0 (0.0)	0.81 (0.08)
	all	-	0.5 (0.02)	0.02 (0.02)	0.0 (0.0)	0.98 (0.03)
ZeroAction	0	0.2	0.88 (0.01)	0.59 (0.02)	0.01 (0.01)	0.03 (0.02)
		0.5	0.72 (0.03)	0.39 (0.01)	0.0 (0.0)	0.24 (0.06)
		0.8	0.55 (0.06)	0.32 (0.01)	0.0 (0.0)	0.51 (0.13)
	all	0.2	0.33 (0.02)	0.21 (0.01)	0.0 (0.0)	0.58 (0.03)
		0.5	0.05 (0.01)	0.03 (0.0)	0.0 (0.0)	0.94 (0.03)
		0.8	0.0 (0.01)	0.0 (0.0)	0.0 (0.0)	1.0 (0.02)

Table 7: Metrics for the **sequential** training setup. A **SAC** agent is trained with **HER** on the goal-conditioned `FetchReach-v2` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	Dim.	Value	Avg. perf.	Fwd. trans.	Bwd. trans.	Forgetting
InvertAction	0	-	0.38 (0.07)	-0.04 (0.02)	0.0 (0.0)	0.88 (0.1)
	all	-	0.21 (0.08)	-0.02 (0.02)	0.0 (0.0)	1.0 (0.0)
NoiseAction	0	0.2	1.0 (0.0)	0.94 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.98 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.8	0.98 (0.01)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	1.0 (0.0)	0.0 (0.0)	0.0 (0.0)
		0.5	0.99 (0.01)	0.92 (0.04)	0.0 (0.0)	0.0 (0.0)
		0.8	0.94 (0.03)	0.84 (0.04)	0.0 (0.0)	0.0 (0.0)
OffsetAction	0	0.2	1.0 (0.0)	1.0 (0.0)	0.0 (0.0)	0.0 (0.0)
		0.5	0.99 (0.01)	1.0 (0.0)	0.0 (0.0)	0.0 (0.0)
		0.8	0.94 (0.04)	0.98 (0.02)	0.0 (0.0)	0.02 (0.02)
	all	0.2	0.99 (0.01)	1.0 (0.0)	0.0 (0.0)	0.02 (0.02)
		0.5	0.94 (0.05)	1.0 (0.0)	0.0 (0.0)	0.12 (0.1)
		0.8	0.97 (0.02)	0.76 (0.08)	0.0 (0.0)	0.0 (0.0)
RepeatAction	0	0.2	1.0 (0.0)	0.94 (0.04)	0.0 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.92 (0.06)	0.0 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	0.8 (0.06)	0.0 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	0.92 (0.06)	0.0 (0.0)	0.0 (0.0)
		0.5	0.99 (0.01)	0.62 (0.12)	0.0 (0.0)	0.0 (0.0)
		0.8	0.95 (0.04)	0.52 (0.02)	0.0 (0.0)	0.02 (0.02)
ScaleAction	0	0.2	0.98 (0.02)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	0.98 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
SineNoiseAction	0	0.2	1.0 (0.0)	0.94 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.5	0.99 (0.01)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.8	0.99 (0.01)	0.78 (0.07)	0.0 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	1.0 (0.0)	0.0 (0.0)	0.0 (0.0)
		0.5	0.98 (0.01)	0.9 (0.04)	0.0 (0.0)	0.0 (0.0)
		0.8	0.87 (0.01)	0.68 (0.11)	0.0 (0.0)	0.0 (0.0)
SwapAction	0	-	0.61 (0.17)	0.0 (0.04)	0.0 (0.0)	0.54 (0.22)
	all	-	0.87 (0.13)	0.0 (0.03)	0.0 (0.0)	0.2 (0.2)
ZeroAction	0	0.2	0.98 (0.02)	0.96 (0.02)	0.0 (0.0)	0.02 (0.02)
		0.5	1.0 (0.0)	0.94 (0.04)	0.0 (0.0)	0.0 (0.0)
		0.8	0.88 (0.03)	0.7 (0.08)	0.0 (0.0)	0.0 (0.0)
	all	0.2	1.0 (0.0)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.5	1.0 (0.0)	0.96 (0.02)	0.0 (0.0)	0.0 (0.0)
		0.8	0.82 (0.02)	0.58 (0.02)	0.0 (0.0)	0.04 (0.02)

Table 8: Metrics for the **sequential** training setup. A **SAC** agent is trained with **HER** on the goal-conditioned `FetchPush-v2` environment. **One dynamics switch**: The agent trains for $T/2$ steps on the unmodified action dynamic, and then for $T/2$ steps on the modified action dynamic. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	Dim.	Value	Avg. perf.	Fwd. trans.	Bwd. trans.	Forgetting
InvertAction	0	-	0.24 (0.12)	0.1 (0.05)	0.0 (0.0)	0.59 (0.2)
	all	-	0.26 (0.1)	0.08 (0.04)	0.0 (0.0)	0.55 (0.19)
NoiseAction	0	0.2	1.0 (0.24)	0.67 (0.25)	0.36 (0.23)	0.0 (0.0)
		0.5	0.98 (0.3)	0.88 (0.34)	0.25 (0.15)	0.03 (0.03)
		0.8	0.96 (0.33)	0.79 (0.32)	0.21 (0.21)	0.11 (0.11)
	all	0.2	1.0 (0.01)	0.78 (0.15)	0.2 (0.16)	0.0 (0.0)
		0.5	0.98 (0.24)	0.85 (0.18)	0.25 (0.19)	0.28 (0.24)
		0.8	0.92 (0.16)	0.71 (0.18)	0.22 (0.16)	0.17 (0.17)
OffsetAction	0	0.2	0.65 (0.22)	0.92 (0.05)	0.0 (0.0)	0.33 (0.2)
		0.5	0.78 (0.14)	0.84 (0.04)	0.0 (0.0)	0.29 (0.14)
		0.8	0.6 (0.16)	0.31 (0.06)	0.0 (0.0)	0.33 (0.18)
	all	0.2	0.43 (0.21)	0.92 (0.06)	0.0 (0.0)	0.53 (0.22)
		0.5	0.54 (0.2)	0.74 (0.06)	0.0 (0.0)	0.47 (0.18)
		0.8	0.41 (0.16)	0.18 (0.07)	0.0 (0.0)	0.47 (0.21)
RepeatAction	0	0.2	0.98 (0.31)	0.66 (0.31)	0.38 (0.26)	0.09 (0.06)
		0.5	0.54 (0.21)	0.95 (0.1)	0.02 (0.02)	0.54 (0.24)
		0.8	0.99 (0.22)	0.88 (0.12)	0.12 (0.08)	0.1 (0.07)
	all	0.2	0.98 (0.3)	0.67 (0.35)	0.33 (0.26)	0.07 (0.04)
		0.5	0.77 (0.2)	0.88 (0.05)	0.02 (0.02)	0.24 (0.17)
		0.8	0.94 (0.16)	0.64 (0.09)	0.08 (0.05)	0.08 (0.05)
ScaleAction	0	0.2	0.57 (0.16)	0.31 (0.06)	0.0 (0.0)	0.31 (0.19)
		0.5	0.62 (0.16)	0.9 (0.04)	0.0 (0.0)	0.37 (0.17)
		0.8	0.82 (0.14)	0.94 (0.04)	0.0 (0.0)	0.22 (0.18)
	all	0.2	0.44 (0.14)	0.35 (0.08)	0.0 (0.0)	0.33 (0.2)
		0.5	0.44 (0.23)	0.82 (0.05)	0.0 (0.0)	0.55 (0.22)
		0.8	0.46 (0.18)	0.92 (0.03)	0.0 (0.0)	0.55 (0.17)
SineNoiseAction	0	0.2	0.89 (0.34)	0.96 (0.36)	0.07 (0.07)	0.11 (0.07)
		0.5	0.88 (0.41)	0.96 (0.41)	0.08 (0.08)	0.23 (0.14)
		0.8	0.98 (0.31)	0.72 (0.3)	0.25 (0.18)	0.06 (0.06)
	all	0.2	0.98 (0.08)	0.83 (0.13)	0.19 (0.19)	0.06 (0.06)
		0.5	0.95 (0.18)	0.79 (0.22)	0.26 (0.21)	0.19 (0.19)
		0.8	0.92 (0.17)	0.44 (0.15)	0.07 (0.05)	0.02 (0.02)
SwapAction	0	-	0.51 (0.16)	0.2 (0.08)	0.0 (0.0)	0.43 (0.15)
	all	-	0.67 (0.1)	0.24 (0.12)	0.0 (0.0)	0.24 (0.06)
ZeroAction	0	0.2	0.98 (0.31)	0.61 (0.32)	0.33 (0.23)	0.03 (0.03)
		0.5	0.85 (0.16)	0.58 (0.1)	0.1 (0.07)	0.12 (0.08)
		0.8	0.62 (0.02)	0.17 (0.04)	0.17 (0.1)	0.02 (0.02)
	all	0.2	0.98 (0.45)	0.87 (0.46)	0.44 (0.34)	0.44 (0.38)
		0.5	0.68 (0.14)	0.49 (0.13)	0.1 (0.07)	0.22 (0.16)
		0.8	0.56 (0.03)	0.13 (0.04)	0.19 (0.1)	0.02 (0.02)

Table 9: Metrics for the **sequential** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. **Continual adaptation**: The agent trains for $T/10$ steps on the unmodified action dynamic, and then nine times for $T/10$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value (increasing for `NoiseAction` and `OffsetAction`, decreasing for `ScaleAction`). The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	Dim.	Avg. perf.	Fwd. trans.	Bwd. trans.	Forgetting
NoiseAction	0	1.0 (0.0)	0.19 (0.0)	0.0 (0.0)	0.0 (0.0)
	all	0.97 (0.0)	0.19 (0.0)	0.0 (0.0)	0.0 (0.0)
OffsetAction	0	0.71 (0.05)	0.19 (0.0)	0.04 (0.02)	0.09 (0.01)
	all	0.04 (0.0)	0.11 (0.0)	0.0 (0.0)	0.11 (0.0)
ScaleAction	0	0.98 (0.0)	0.19 (0.0)	0.0 (0.0)	0.01 (0.0)
	all	0.03 (0.0)	0.1 (0.0)	0.0 (0.0)	0.11 (0.0)

Table 10: Metrics for the **sequential** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. **Continual adaptation**: The agent trains for $T/10$ steps on the unmodified action dynamic, and then nine times for $T/10$ steps on modified action dynamics with increasing difficulty, determined by the wrapper value (increasing for `NoiseAction` and `OffsetAction`, decreasing for `ScaleAction`). The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	Dim.	Avg. perf.	Fwd. trans.	Bwd. trans.	Forgetting
NoiseAction	0	0.83 (0.01)	0.18 (0.0)	0.01 (0.0)	0.02 (0.0)
	all	0.34 (0.0)	0.12 (0.0)	0.0 (0.0)	0.06 (0.0)
OffsetAction	0	0.36 (0.02)	0.15 (0.0)	0.04 (0.01)	0.11 (0.01)
	all	-0.03 (0.0)	0.08 (0.0)	0.01 (0.0)	0.08 (0.0)
ScaleAction	0	0.81 (0.03)	0.17 (0.0)	0.04 (0.0)	0.06 (0.01)
	all	-0.02 (0.0)	0.06 (0.0)	0.01 (0.0)	0.08 (0.0)

Table 11: Metrics for the **multi-task** training setup. A **SAC** agent is trained on the `walker-walk-v0` environment. The agent trains on two tasks resulting from the unmodified action dynamic and one modification of such. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	Dim.	Value	Avg. perf.	Parallel trans.
InvertAction	0	-	0.99 (0.03)	-0.02 (0.03)
	all	-	0.86 (0.08)	-0.14 (0.08)
NoiseAction	0	0.2	1.0 (0.05)	-0.04 (0.05)
		0.5	1.0 (0.0)	0.01 (0.01)
		0.8	1.0 (0.02)	-0.0 (0.0)
	all	0.2	1.0 (0.01)	-0.0 (0.01)
		0.5	1.0 (0.0)	-0.0 (0.0)
		0.8	0.97 (0.01)	-0.02 (0.01)
OffsetAction	0	0.2	0.98 (0.01)	-0.0 (0.02)
		0.5	1.0 (0.0)	0.02 (0.01)
		0.8	0.99 (0.01)	-0.0 (0.01)
	all	0.2	1.0 (0.0)	-0.0 (0.0)
		0.5	0.98 (0.01)	-0.0 (0.01)
		0.8	0.55 (0.03)	-0.28 (0.08)
RepeatAction	0	0.2	1.0 (0.0)	-0.0 (0.0)
		0.5	0.99 (0.02)	-0.02 (0.02)
		0.8	1.0 (0.0)	0.0 (0.0)
	all	0.2	0.99 (0.0)	-0.0 (0.0)
		0.5	0.98 (0.0)	-0.0 (0.01)
		0.8	0.95 (0.0)	-0.03 (0.0)
ScaleAction	0	0.2	1.0 (0.0)	-0.0 (0.0)
		0.5	1.0 (0.0)	0.01 (0.01)
		0.8	1.0 (0.0)	-0.0 (0.0)
	all	0.2	0.51 (0.01)	-0.08 (0.02)
		0.5	0.71 (0.08)	-0.17 (0.07)
		0.8	1.0 (0.0)	0.01 (0.01)
SineNoiseAction	0	0.2	1.0 (0.0)	-0.0 (0.0)
		0.5	1.0 (0.0)	0.0 (0.0)
		0.8	1.0 (0.0)	-0.0 (0.0)
	all	0.2	1.0 (0.0)	0.0 (0.0)
		0.5	0.98 (0.05)	-0.05 (0.05)
		0.8	0.9 (0.04)	-0.08 (0.04)
SwapAction	0	-	0.62 (0.1)	-0.0 (0.01)
	all	-	0.55 (0.02)	0.02 (0.02)
ZeroAction	0	0.2	0.99 (0.03)	-0.03 (0.03)
		0.5	0.97 (0.01)	0.0 (0.03)
		0.8	0.95 (0.04)	-0.01 (0.06)
	all	0.2	0.76 (0.06)	-0.16 (0.06)
		0.5	0.52 (0.01)	-0.08 (0.01)
		0.8	0.52 (0.02)	-0.04 (0.02)

Table 12: Metrics for the **multi-task** training setup. A **SAC** agent is trained on the `HalfCheetah-v4` environment. The agent trains on two tasks resulting from the unmodified action dynamic and one modification of such. The modification is configured with the action wrapper, the action dimension and if applicable a wrapper specific value, see Section 4 and Table 1.

Wrapper	**Dim.**	**Value**	**Avg. perf.**	**Parallel trans.**
InvertAction	0	-	1.0 (0.02)	-0.06 (0.01)
	all	-	0.99 (0.12)	-0.16 (0.02)
NoiseAction	0	0.2	0.98 (0.02)	0.0 (0.03)
		0.5	0.95 (0.01)	-0.02 (0.01)
		0.8	0.92 (0.18)	-0.11 (0.18)
	all	0.2	0.92 (0.01)	0.02 (0.07)
		0.5	0.73 (0.19)	-0.34 (0.11)
		0.8	0.62 (0.22)	-0.42 (0.08)
OffsetAction	0	0.2	1.0 (0.19)	-0.13 (0.16)
		0.5	0.97 (0.02)	0.01 (0.04)
		0.8	0.93 (0.01)	-0.02 (0.02)
	all	0.2	0.98 (0.01)	-0.01 (0.0)
		0.5	0.9 (0.01)	-0.04 (0.02)
		0.8	0.7 (0.06)	-0.2 (0.05)
RepeatAction	0	0.2	0.87 (0.01)	-0.02 (0.06)
		0.5	0.84 (0.04)	-0.09 (0.1)
		0.8	0.8 (0.14)	-0.21 (0.1)
	all	0.2	0.71 (0.03)	-0.18 (0.03)
		0.5	0.56 (0.05)	-0.22 (0.03)
		0.8	0.67 (0.18)	-0.38 (0.07)
ScaleAction	0	0.2	0.86 (0.18)	-0.26 (0.08)
		0.5	0.91 (0.18)	-0.17 (0.13)
		0.8	0.97 (0.01)	-0.06 (0.04)
	all	0.2	0.56 (0.14)	-0.34 (0.06)
		0.5	0.72 (0.02)	-0.11 (0.02)
		0.8	0.91 (0.01)	-0.05 (0.02)
SineNoiseAction	0	0.2	0.98 (0.01)	-0.02 (0.02)
		0.5	0.96 (0.0)	0.01 (0.02)
		0.8	0.93 (0.01)	-0.03 (0.02)
	all	0.2	0.93 (0.26)	-0.36 (0.15)
		0.5	0.74 (0.03)	-0.18 (0.02)
		0.8	0.67 (0.12)	-0.39 (0.05)
SwapAction	0	-	0.61 (0.09)	-0.04 (0.02)
	all	-	0.51 (0.11)	-0.16 (0.07)
ZeroAction	0	0.2	0.88 (0.05)	-0.07 (0.06)
		0.5	0.88 (0.04)	-0.21 (0.02)
		0.8	0.85 (0.16)	-0.21 (0.1)
	all	0.2	0.64 (0.03)	-0.19 (0.02)
		0.5	0.51 (0.01)	-0.2 (0.01)
		0.8	0.5 (0.13)	-0.34 (0.04)