Fine-tuning Reinforcement Learning Models is Secretly a Forgetting Mitigation Problem

Maciej Wołczyk [*1] **Bartłomiej Cupiał** [*1 2] **Mateusz Ostaszewski** [3] **Michał Bortkiewicz** [3] **Michał Zając** [4] **Razvan Pascanu** [5] **Łukasz Kuciński** [1 2 6] **Piotr Miłoś** [1 2 6 7]

Abstract

Fine-tuning is a widespread technique that allows practitioners to transfer pre-trained capabilities, as recently showcased by the successful applications of foundation models. However, fine-tuning reinforcement learning (RL) models remains a challenge. This work conceptualizes one specific cause of poor transfer, accentuated in the RL setting by the interplay between actions and observations: *forgetting of pre-trained capabilities*. Namely, a model deteriorates on the state subspace of the downstream task not visited in the initial phase of fine-tuning, on which the model behaved well due to pre-training. This way, we lose the anticipated transfer benefits. We identify conditions when this problem occurs, showing that it is common and, in many cases, catastrophic. Through a detailed empirical analysis of the challenging NetHack and Montezuma's Revenge environments, we show that standard knowledge retention techniques mitigate the problem and thus allow us to take full advantage of the pre-trained capabilities. In particular, in NetHack, we achieve a new state-of-the-art for neural models, improving the previous best score from 5K to over 10K points in the Human Monk scenario.

1. Introduction

Fine-tuning neural networks is a widespread technique in deep learning for knowledge transfer between datasets (Yosinski et al., 2014; Girshick et al., 2014). Its effectiveness has recently been showcased by spectacular successes in the deployment of foundation models in downstream tasks, including natural language processing (Chung et al., 2022), computer vision (Sandler et al., 2022), automatic speech recognition (Zhang et al., 2022), and cheminformatics (Chithrananda et al., 2020). These successes are predominantly evident in supervised and self-supervised learning domains. However, achievements of comparable significance have not yet fully found their way to reinforcement learning (RL) (Wulfmeier et al., 2023).

In this study, we explore the challenges and solutions for effectively transferring knowledge from a pre-trained model to a downstream task in the context of RL fine-tuning. We find that the interplay between actions and observations in RL leads to a changing visitation of states during the fine-tuning process with catastrophic consequences. Intuitively, the agent may lose pre-trained abilities in parts of the downstream task not covered in early fine-tuning, diminishing the expected transfer benefits.

We refer to this issue as *forgetting of pre-trained capabilities* (FPC). We identify two important instances of FPC: *state coverage gap* and *imperfect cloning gap*, illustrated in Figure 1 and defined in Section 2. We show empirically that the problem is severe, as these instances are often encountered in practice, leading to poor transfer to downstream tasks. These findings are in contrast to the conventional wisdom that emerged from the supervised learning setting, where the data distribution is i.i.d. and forgetting is not a factor if one cares only about the performance on the downstream task; see (Wulfmeier et al., 2023, Sec 3.5) and (Radford et al., 2018; Devlin et al., 2019; Dosovitskiy et al., 2020).

Finally, we show that phrasing *state coverage gap* and *imperfect cloning gap* as instances of *forgetting* is meaningful as typical retention techniques (Kirkpatrick et al., 2017; Rebuffi et al., 2017; Wołczyk et al., 2021) can alleviate these problems. We demonstrate this effect on NetHack, Montezuma's Revenge, and tasks built out of Meta-World, an environment simulating tasks for robotic arms. Applying knowledge retention enhances the fine-tuning performance on all environments and leads to a 2x improvement in the state-of-art results for neural models on NetHack. Further analysis shows that forgetting of pre-trained capabilities is at the heart of the problem, as vanilla fine-tuning rapidly

[*]Equal contribution [1]IDEAS NCBR [2]University of Warsaw [3]Warsaw University of Technology [4]Jagiellonian University [5]Google DeepMind [6]Institute of Mathematics, Polish Academy of Sciences [7]deepsense.ai. Correspondence to: Maciej Wołczyk <maciej.wolczyk@gmail.com>.

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

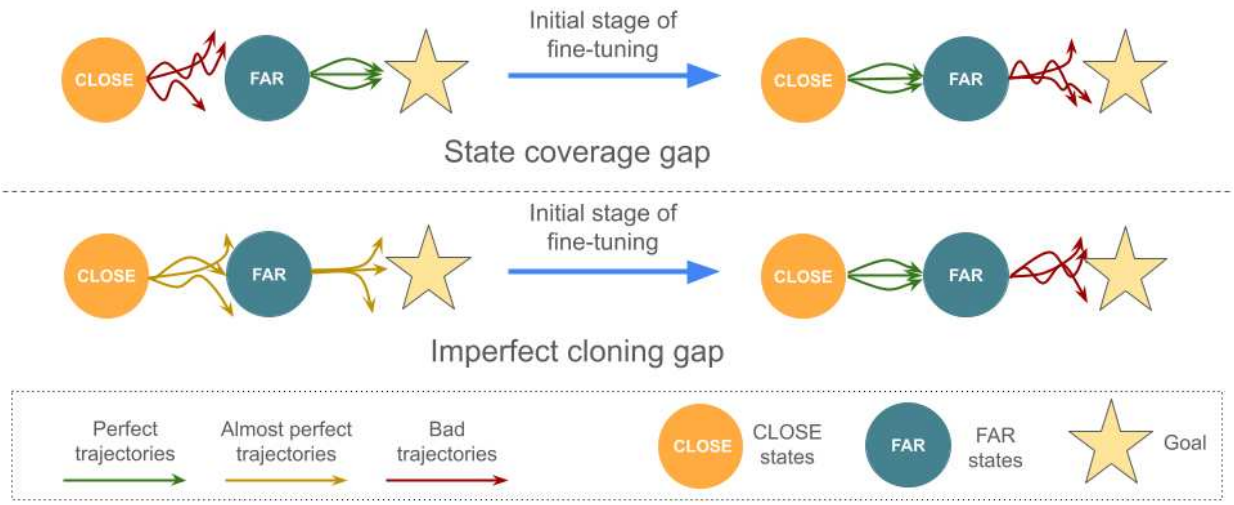


Figure 1: **Forgetting of pre-trained capabilities**. For illustration, we partition the states of the downstream task into CLOSE and FAR, depending on the distance from the starting state; the agent must master FAR to reach the goal. In the state coverage gap (top), the pre-trained policy performs perfectly on FAR but is suboptimal on CLOSE. During the initial stage of fine-tuning, while mastering CLOSE, the policy deteriorates, often catastrophically, on FAR. In imperfect cloning gap (bottom), the pre-trained policy is decent both on CLOSE and FAR; however, due to compounding errors in the initial stages of fine-tuning, the agent rarely visits FAR, and the policy deteriorates on this part. In both cases, the deteriorated policy on FAR is hard to recover and thus necessitates long training to solve the whole task.

forgets how to perform in parts of the state space not encountered immediately in the downstream task.

As such, the main recommendation of our work is that methods targeting catastrophic forgetting should be routinely used in transfer RL scenarios. In summary, our contributions are as follows:

- We pinpoint forgetting of pre-trained capabilities as a critical problem limiting transfer from pre-trained models in RL and provide a conceptualization of this phenomenon, along with its two common instances: state coverage gap and imperfect cloning gap.
- We propose knowledge retention techniques as a tool that mitigates FPC and allows us to transfer from the pre-trained model efficiently.
- We thoroughly examine our approach on Nethack, Montezuma's Revenge, and sequential robotic tasks, improving the state-of-the-art for neural models on NetHack by 2x.

2. Forgetting of pre-trained capabilities

To illustrate the forgetting of pre-trained capabilities, let us consider a Markov Decision Problem (MDP) where the state space can be approximately split into two sets: CLOSE and FAR, see Figure 1. The states in CLOSE are easily reachable from the starting state and the agent frequently visits them. The states in FAR are reachable only by going through CLOSE; hence, they are infrequently visited as they can be reached only once some learning on CLOSE happens. For example, an agent learning to play a video game might only see the first level of the game (CLOSE) at the start of the training before it learns how to get to the subsequent levels (FAR).

Forgetting of pre-trained capabilities happens when a model performing well on FAR loses this ability due to interference in the function approximator when training on CLOSE. We believe this problem has not yet been studied thoroughly and has a major significance for transfer RL since it is commonly present in standard RL settings and often leads to substantial performance deterioration. The subsequent experimental sections provide multiple examples of its occurrence, and in Appendix A, we show that it can be observed already in simple two-state MDPs as well as gridworlds. To facilitate further study of this problem, we highlight two specific scenarios where forgetting of pre-trained capabilities occurs: the state coverage gap and imperfect cloning gap.

In **state coverage gap**, we consider a pre-trained agent that is performing well mostly on FAR and does not know how to behave on CLOSE. However, when fine-tuned on CLOSE, its behavior on FAR will deteriorate considerably due to

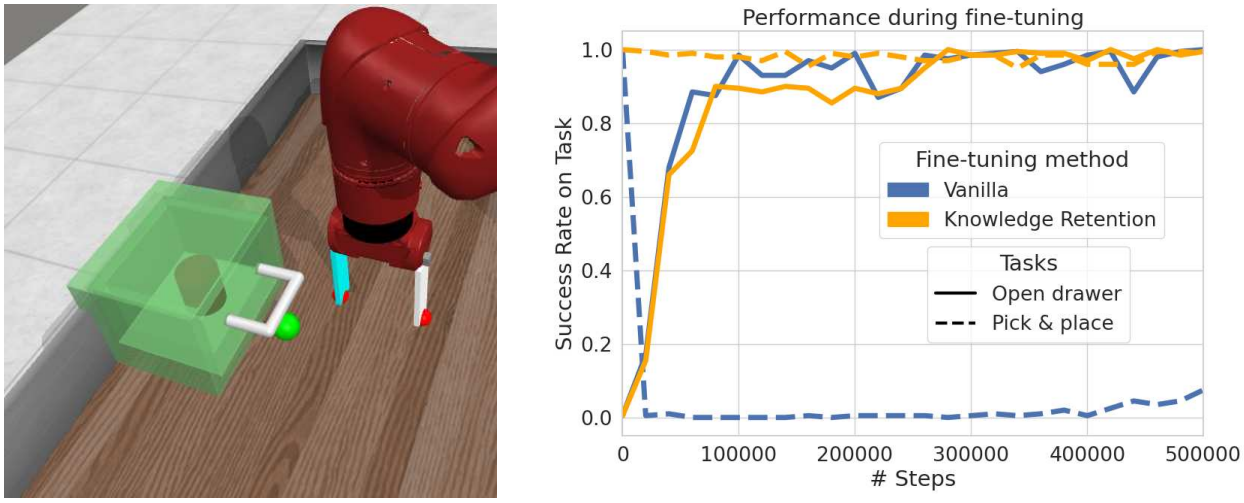


Figure 2: Example of **state coverage gap**. (Left) We assume that a pre-trained model is able to pick and place objects (e.g., the cylinder). However, it does not know how to open drawers. Consider a new task in which the agent needs first to open the drawer (CLOSE states) and then pick and place the object (FAR states). (Right) During fine-tuning, the model rapidly forgets how to manipulate objects before learning to open the drawer and struggles to reacquire this skill (dashed blue line). Knowledge retention techniques alleviate this issue (dashed orange line). At the same time, in both cases, the model learns how to open the drawer (solid lines).

forgetting[1] and will have to be re-acquired. This setting is representative of common transfer RL scenarios (Parisotto et al., 2015; Rusu et al., 2016; 2022), see also the top row of Figure 1 and Figure 2 for illustration.

The **imperfect cloning gap** occurs when the pre-trained agent is a perturbed version of an agent that is effective in the current environment. Even if the difference is small, this discrepancy can lead to a substantial imbalance with the agent visiting states in CLOSE much more often than FAR. While trying to correct the slightly suboptimal policy on CLOSE, the policy on FAR can get worse due to forgetting, see the depiction in Figure 1. Such scenarios frequently arise due to slight changes in the reward structure between pre-training and fine-tuning or approximation errors when cloning an expert policy, and, more generally, when using models pre-trained on offline static datasets (Nair et al., 2020; Baker et al., 2022; Zheng et al., 2023).

Knowledge retention In this paper, we argue that to benefit from fine-tuning pre-trained RL models, we need to mitigate FPC. To this end, we consider the following popular methods for knowledge retention: Elastic Weight Consolidation (EWC), replay by behavioral cloning (BC), kickstarting (KS), and episodic memory (EM). **EWC** is a regularization-based approach that applies a penalty on parameter changes by introducing an auxiliary loss: $\mathcal{L}_{aux}(\theta) = \sum_i F^i(\theta^i_* - \theta^i)^2$, where θ (resp θ_*) are the weights of the current (resp. pre-trained) model, and F is the diagonal of the Fisher matrix. We also use **behavioral cloning**, an efficient replay-based approach (Rebuffi et al., 2017; Wolczyk et al., 2022). We implement BC in the following way. Before the training, we gather a subset of states $\mathcal{S}_{BC}$ on which the pre-trained model π_* was trained, and we construct a buffer $\mathcal{B}_{BC} := \{(s, \pi_*(s)) : s \in \mathcal{S}_{BC}\}$. For the fine-tuning phase, we initialize the policy with θ_* and we apply an auxiliary loss of the form $\mathcal{L}_{BC}(\theta) = \mathbb{E}_{s\sim\mathcal{B}_{BC}}[D_{KL}(\pi_*(s) \parallel \pi_\theta(s))]$ alongside the RL objective. **Kickstarting** applies KL of a similar form, but the expectation is over data sampled by the current policy, i.e., $\mathcal{L}_{KS}(\theta) = \mathbb{E}_{s\sim\pi_\theta}[D_{KL}(\pi_*(s) \parallel \pi_\theta(s))]$. For **episodic memory**, we can easily use it with off-policy methods by simply keeping the examples from the pre-trained task in the replay buffer when training on the new task. Following previous best practices (Wolczyk et al., 2022), we do not apply knowledge retention to the parameters of the critic. See Appendix C for more details.

Relation to continual reinforcement learning The main focus of this paper is the efficient fine-tuning of a pre-trained RL agent. We consider forgetting only as far as it impacts the transfer and we are *solely* interested in the performance on the downstream task, disregarding the performance of the pre-trained tasks. This is in contrast to continual reinforcement learning (Khetarpal et al., 2022; Wołczyk et al., 2021; Kirkpatrick et al., 2017), where one of the goals is to retain the performance on the pre-trained tasks. Interestingly, we show that contrary to prior knowledge (Wulfmeier et al., 2023), forgetting might severely hinder the transfer

[1]For a more thorough discussion on the nature of interference in RL we refer the reader to Schaul et al. (2019)

capabilities in standard transfer RL settings with a stationary downstream task.

3. Experimental setup

We perform experiments on three environments: NetHack, Montezuma's Revenge, and `RoboticSequence`. Below, we describe them in detail and show concrete instances of concepts from Section 2 such as pre-trained policy π_* or FAR and CLOSE sets. In each environment, we run vanilla fine-tuning and training from scratch as baselines, and we test fine-tuning with different knowledge retention methods (e.g., Fine-tuning + BC).

NetHack Learning Environment (Küttler et al., 2020) is a complex game consisting of procedurally generated multi-level dungeons. Since their layouts are randomly generated in each run, the player has to learn a general strategy rather than memorize solutions. NetHack is stochastic and requires mastering diverse skills, such as maze navigation, searching for food, fighting, and casting spells. It has been a popular video game for decades that recently has become a challenging testbed at the forefront of RL research (Hambro et al., 2022a; Piterbarg et al., 2023; Klissarov et al., 2023). Due to computational constraints, we focus solely on a single setting in our experiments, i.e., Human Monk. The code is available at `https://github.com/BartekCupial/finetuning-RL-as-CL`.

We take the current state-of-the-art neural model (Tuyls et al., 2023) as our pre-trained policy π_*. It was trained using behavioral cloning on 115B environment transitions sampled from AutoAscend, a rule-based agent that is currently the best-performing bot. The policy π_* scores over 5K points.

Since the policy π_* rarely leaves the first level of the game (see Figure 4), we conceptualize CLOSE as the set of states corresponding to this initial level. Accordingly, FAR represents states from subsequent levels. During fine-tuning, we use asynchronous PPO (APPO) (Petrenko et al., 2020). More technical details, including the neural network architecture, can be found in Appendix B.1.

Montezuma's Revenge is a popular video game that requires the player to advance through a sequence of rooms filled with traps and enemies while collecting treasures and keys (Bellemare et al., 2013). The environment has sparse rewards and is a well-known exploration challenge in RL.

We pre-train a policy π_* on a part of the environment that includes only rooms from a certain room onward (see the layout of the game in Figure 12 in Appendix B.2). In particular, in the main text, we start pre-training from Room 7 and we verify other room choices in Appendix E. During fine-tuning, the agent has to solve the whole game, starting from the first room. As such, Room 7 and subsequent ones represent the FAR states, and the preceding rooms represent CLOSE states. We conduct experiments using PPO with Random Network Distillation (Burda et al., 2018) to boost exploration, which is essential in this sparse reward environment. More technical details, including the neural network architecture, can be found in Appendix B.2.

`RoboticSequence` is a multi-stage robotic task based on the Meta-World benchmark (Yu et al., 2020). The robot is successful only if during a single episode, it completes sequentially the following sub-tasks: use a hammer to hammer in a nail (`hammer`), push an object from one specific place to another (`push`), remove a bolt from a wall (`peg-unplug-side`), push an object around a wall (`push-wall`).

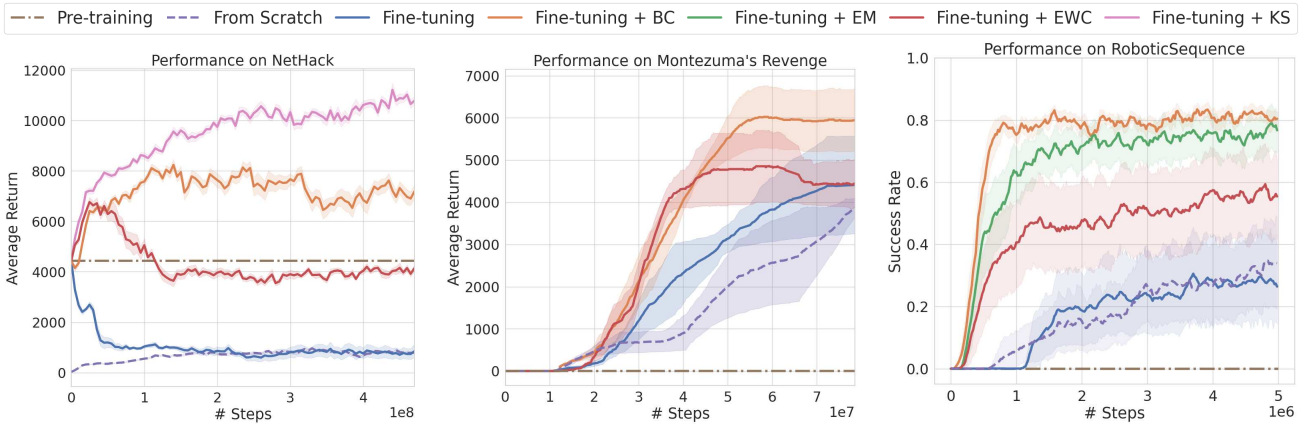
We use a pre-trained policy π_* that can solve the last two stages, `peg-unplug-side` and `push-wall` (FAR), but not the first two, `hammer` and `push` (CLOSE). See Figure 2 for an example of another, two-stage instantiation of `RoboticSequence`. We use Soft Actor-Critic (SAC) (Haarnoja et al., 2018a) in all robotic experiments. More technical details, including the neural network architecture, can be found in Appendix B.3.

4. Main result: knowledge retention mitigates forgetting of pre-trained capabilities

In this section, we present empirical results showing that across all environments (1) vanilla fine-tuning often fails to leverage pre-trained knowledge, and, importantly, (2) the knowledge retention methods fix this problem, unlocking the potential of the pre-trained model and leading to significant improvements. Here, we focus on performance and defer a detailed analysis of the forgetting of pre-trained capabilities phenomenon to Section 5.

NetHack We demonstrate that fine-tuning coupled with knowledge retention methods surpasses the current state-of-the-art (Tuyls et al., 2023) by 2x, achieving 10K points when compared to the previous 5K, see Figure 3a. Interestingly, vanilla fine-tuning alone proves insufficient, as the agent's performance deteriorates, losing pre-trained capabilities and failing to recover from this loss.

We discover that retaining the prior knowledge unlocks the possibility of improving the policy during fine-tuning, see Figure 3a. However, choosing an effective method for knowledge retention is nuanced, as discussed in the commentary at the end of this section. In the context of NetHack, KS works best, followed by BC, both surpassing the state-of-the-art. Conversely, EWC shows poor performance, deteriorating after some training. Importantly, implementing knowledge retention within existing frameworks is straightforward, distinguishing our method from the more intricate



(a) NetHack (b) Montezuma's Revenge (c) `RoboticSequence`

Figure 3: Performance on (a) NetHack, (b) Montezuma's Revenge, and (c) `RoboticSequence`. For NetHack, the FPC is driven by imperfect cloning gap, while for the remaining two by state coverage gap. In all cases, knowledge retention techniques improve the performance of fine-tuning. We omit KS in Montezuma's Revenge and `RoboticSequence` as it underperforms.

approaches used for NetHack (Piterbarg et al., 2023; Klissarov et al., 2023), which utilize large language models or hierarchical reinforcement learning. We note that our best agent performs well not only in terms of the overall score but other metrics that are relevant in NetHack, such as the number of visited levels or amount of gold gathered, see Appendix D.

Montezuma's Revenge We show that adding a knowledge retention technique in the form of BC improves not only the speed of learning but also the performance when compared to vanilla fine-tuning or training from scratch, see Figure 3b. EWC also outperforms training from scratch and converges faster than vanilla fine-tuning, although it saturates on the lower average return. The performance of the BC version starts to diverge from vanilla fine-tuning at around 20M steps when the agent starts to enter Room 7, which is the first room observed in pre-training. This is where the beneficial effects of state coverage gap mitigation come into play.

`RoboticSequence` We show that the knowledge retention methods mitigate the state coverage gap problem and allow the model to benefit from pre-trained capabilities, see Figure 3c. In terms of performance and speed of learning, BC is the most effective, followed by EM and EWC, respectively. Notably, BC successfully solves all four stages of `RoboticSequence` 80% of the time, a strong result considering the challenges posed by compounding failure probabilities; see Figure 7 for success rates of individual stages. Importantly, vanilla fine-tuning or training from scratch are virtually indistinguishable, and both significantly fall behind BC, EM, and EWC.

Discussion of knowledge retention methods Although knowledge retention methods improve the performance of fine-tuning, the choice of the right approach is crucial. We observe that the choice between KS and BC depends on the nature of the problem and, when in doubt, it might be prudent to test both. For NetHack and imperfect cloning gap case, where the agent is initialized to closely mimic the expert, it might be sufficient to prevent forgetting on states visited online by the fine-tuned policy, hence use KS. On the other hand, we found that BC is successful in mitigating state coverage gap, a phenomenon appearing in Montezuma's Revenge and `RoboticSequence`, as it allows the fine-tuned policy to learn on CLOSE and prevents it from forgetting on FAR. KS fails completely in this setting, as it tries to match the pre-trained model's outputs also on CLOSE states, which were not present in pre-training. As such, we do not report metrics for KS in these environments.

Episodic memory (EM) performs well on `RoboticSequence`, where we use SAC. However, it can be only applied with algorithms that employ an off-policy replay buffer. Since NetHack and Montezuma's Revenge leverage, respectively, APPO and PPO, it cannot be trivially applied in their case. Finally, although EWC exceeds vanilla fine-tuning in all settings, it is consistently outperformed by the other approaches.

5. Analysis: forgetting of pre-trained capabilities hinders RL fine-tuning

In this section, we investigate forgetting of pre-trained capabilities in detail, shedding additional light on the reasons for the poor performance of vanilla fine-tuning demonstrated in Section 4. One of the findings is that the results on the FAR

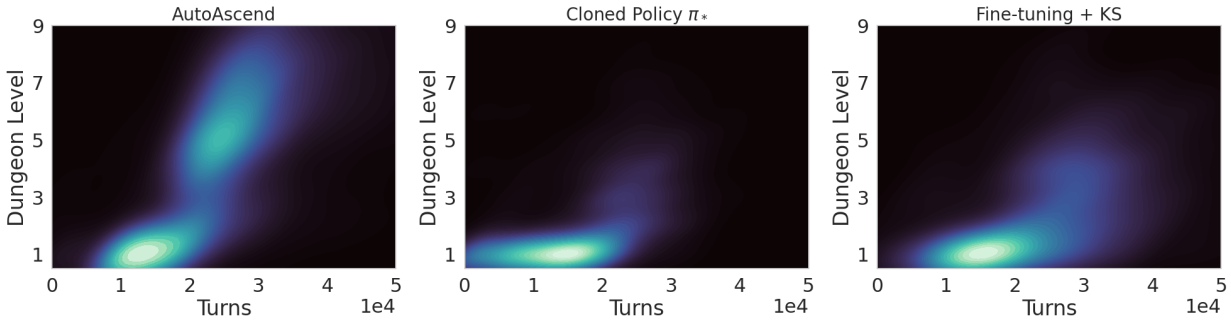


Figure 4: Density plots showing maximum dungeon level achieved compared to the total number of turns (units of in-game time) for expert AutoAscend (left), pre-trained policy π_* (center), and fine-tuning + KS (right) Brighter colors indicate higher visitation density. Level visitation of π_* differs significantly from the level visitation of the AutoAscend expert. This is an example of imperfect cloning gap as the agent will not see further levels at the start of fine-tuning. The knowledge retention-based method manages to perform well and explore different parts of the state space.

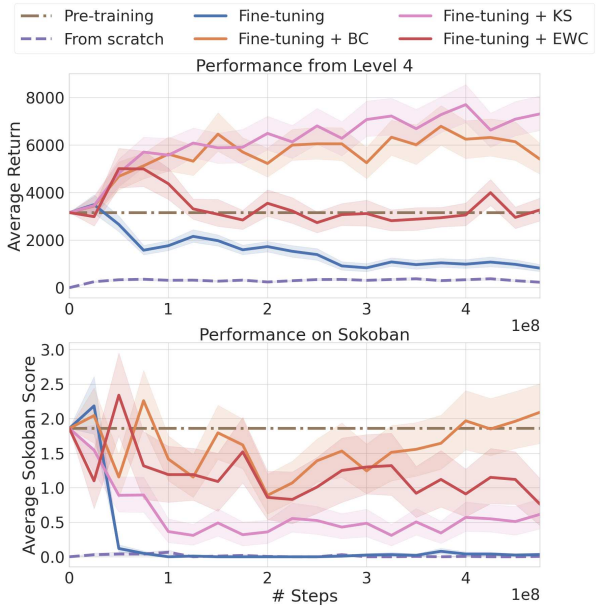


Figure 5: The average return throughout the fine-tuning process on two NetHack tasks: level 4 (top), and Sokoban level (bottom). The result is averaged over 200 episodes, each starting from where the expert (AutoAscend) ended up upon first entering level.

states rapidly decline as we start fine-tuning. Even after re-learning, the final policy is significantly different than the pre-trained one, suggesting that the agent learned a new solution instead of benefiting from the previous one. On the other hand, fine-tuning with knowledge retention techniques is robust to these issues.

NetHack Although π_* is a relatively big model pre-trained on a large amount of data, it fails to capture some of AutoAscend's complex behaviors and long-term strategy, a vivid example of imperfect cloning gap. Indeed, in Figure 4 we can see *a distribution shift between the expert and the pre-trained model* hindering fine-tuning efficiency. We also show that fine-tuning with knowledge retention (KS in this case) manages to overcome this problem and explores the dungeon in a manner more comparable to AutoAscend.

We study the extent to which knowledge retention techniques mitigate the negative effects of imperfect cloning gap on two levels representing FAR states: level 4 and Sokoban level[2], see Figure 5. The performance of fine-tuning on level 4 can be temporarily enhanced by EWC and consistently improved by KS or BC, which is in line with the results presented in Figure 3a. Solving the Sokoban level does not yield immediate rewards so the vanilla fine-tuning agent pursues other strategies that are more beneficial in the short term. As such, it is not surprising that this particular behavior is forgotten. However, forgetting this skill will be hurtful in the long term, since completing the Sokoban levels unlocks a variety of equipment that is crucial for high performance during the rest of the game. Differentiating between behaviors that should be forgotten and those that should be kept is an important future direction for knowledge retention methods.

The Sokoban results allow us to get some insights into the qualitative differences between the KS and BC. Namely, KS struggles with sustaining the performance on Sokoban, as uses trajectories *gathered by the online policy*. These do not contain any Sokoban levels at the start of the fine-tuning, as Sokoban is only encountered in the deeper parts of the dungeon. Conversely, BC uses data *gathered by the expert* and, as a result, constantly rehearses the correct way of solving this puzzle. As such, we note that both BC and KS have

[2]In NetHack, the Sokoban level is a branch of the dungeon modeled and named after an NP-hard game where the goal is to push boxes on target locations, see NetHack wiki and Appendix B.1.

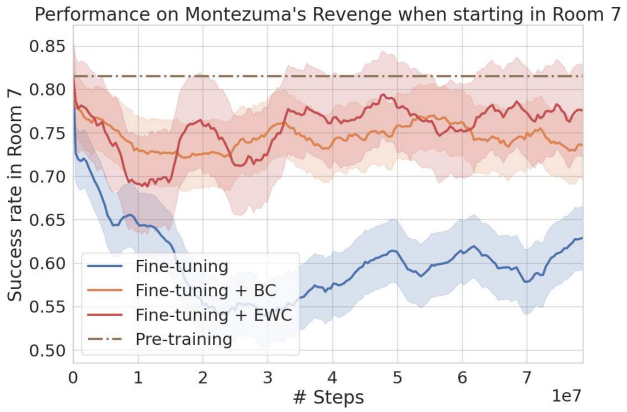


Figure 6: Montezuma's Revenge, success rate in Room 7 which represents a part of the FAR states.

their specific advantages. We identify designing methods that combine these improvements as important future work. See Appendix D for additional NetHack metrics.

Montezuma's Revenge We assess the scope of the state coverage gap problem by evaluating agents in Room 7, throughout the learning process, see Figure 6. This is the first room present in pre-training and as such marks the transition between CLOSE and FAR states. Verifying the agent's performance here allows us to measure how much knowledge was lost. The vanilla fine-tuning success rate[3] drops considerably as the training progresses. While it starts improving when the agent revisits Room 7, i.e., after 20M environment steps, it does not reach the performance of π_*. In contrast to this behavior, both BC and EWC maintain a stable success rate, closely resembling the performance of the expert policy π_* that was pre-trained to solve the game restricted only to the levels following Room 7.

In Appendix E we confirm these findings with different definitions of FAR and CLOSE sets. Additionally, we study how forgetting impacts exploration, showing that with knowledge retention the agent manages to visit a larger number of rooms than with vanilla fine-tuning.

`RoboticSequence` Figure 7 shows that the vanilla fine-tuned agent forgets on FAR states (stages `peg-unplug-side` and `push-wall`), again showcasing state coverage gap. While we observed in Section 4 that the knowledge retention methods mitigate this problem, here we can see the behavior broken down into individual stages. In particular, it is apparent that while learning on `hammer` or `push` (CLOSE), the agent initially forgets how to perform on FAR. Moreover, this deterioration is severe, i.e., when the training finally reaches these stages,

[3]We use this metric since the reward signal in Montezuma's Revenge is too sparse to provide reliable measurements.

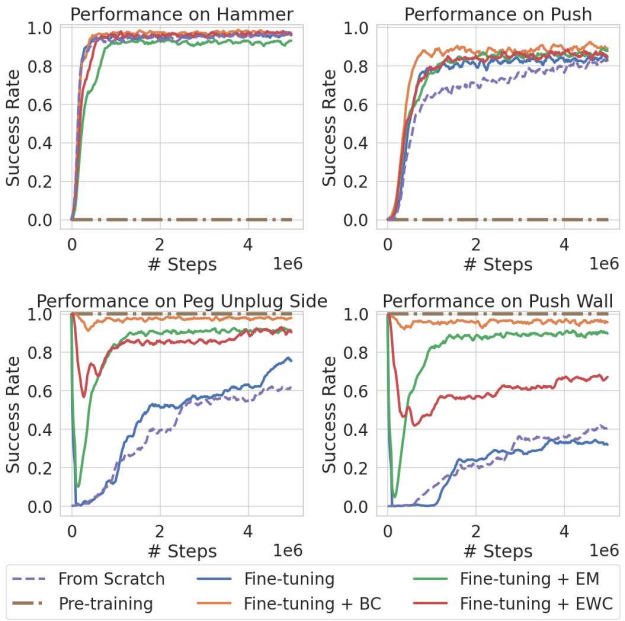


Figure 7: Success rate for each stage of `RoboticSequence`. The fine-tuning experiments start from a pre-trained policy π_* that performs well on `peg-unplug-side` and `push-wall`.

the performance grows slowly. However, BC, EM, and EWC are able to maintain or to a certain degree regain performance (exact results vary by the method). This pinpoints that *the standard fine-tuning does not exhibit positive transfer* of the knowledge of the last two stages.

We supplement this analysis by studying the log-likelihoods assigned by the fine-tuned policy to trajectories collected using the expert policy, i.e., the state-action pairs (s, a^*), where $a^* \sim \pi_*(s)$. This is visualized on Figure 8 where we show how the policy deteriorates in certain parts of the state space (projected to 2D using PCA) in the `push-wall` environment. After $100K$ steps, the model assigns high probability to some of the correct actions on the part of the state space, but its overall success rate has already collapsed to 0. As such, even partial forgetting in the initial stages significantly impacts performance. After the $500K$ steps, the likelihood values collapse on all expert trajectories. The situation changes when the agent relearns how to behave on `push-wall` but the log-likelihoods do not reach the original values, showing that the fine-tuned agent learned a different policy.

We expand this analysis in Appendix F, showing that the hidden representation of the policy network is irreversibly changed in the early stages of fine-tuning and that forgetting of pre-trained capabilities persists irrespective of the network size and aggravates as the size of CLOSE set increases.

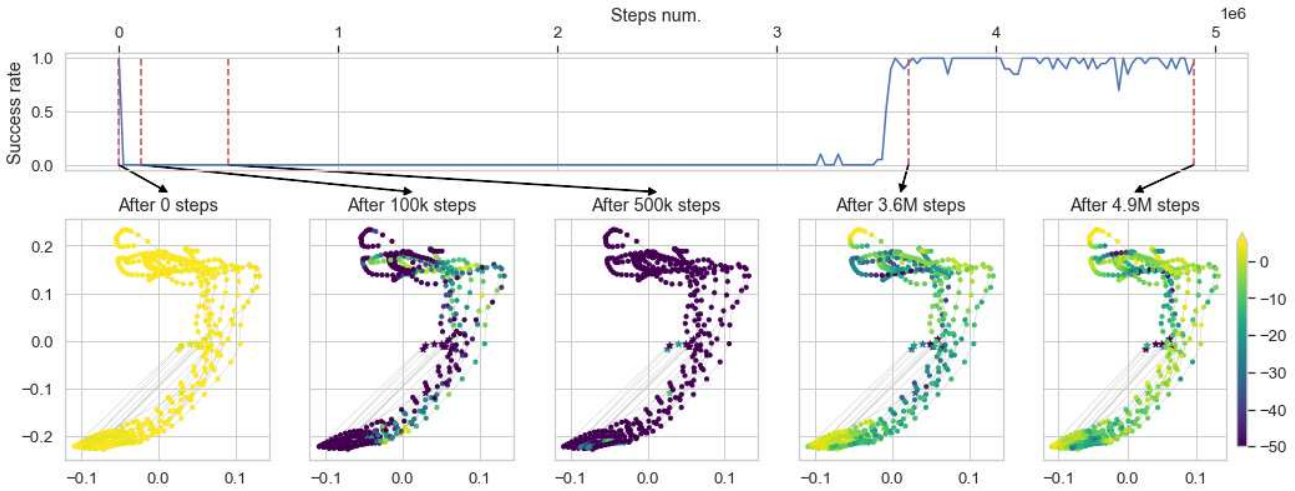


Figure 8: Log-likelihood under the fine-tuned policy of trajectories collected using π_* on `push-wall`, i.e., state-action pairs $(s, a^*), a^* \sim \pi_*(s)$. The top row contains success rates, while the bottom row visualizes 2D PCA projections, color-coded according to the log-likelihood. As fine-tuning progresses the model forgets the initial solution and is unable to recover it.

Other considerations Additionally, we note that choosing the most appropriate knowledge retention method for the problem at hand may depend on other constraints. In particular, if no prior data can be utilized during the fine-tuning, then BC and EM cannot be used, and one should instead leverage EWC, or apply Kickstarting that distills the knowledge on the online data. On the other hand, if one wishes to minimize computational complexity, EWC might be the best choice, as the other methods require processing more examples per training step. Finally, if there are restrictions on memory, one should weigh the cost of keeping the data (needed for BC, EM) against the cost of keeping the parameters of the pre-trained model (needed for EWC, KS).

6. Related Work

Transfer in RL Due to high sample complexity and computation costs, training reinforcement learning algorithms from scratch is expensive (Ceron & Castro, 2021; Vinyals et al., 2019; Machado et al., 2018a). As such, transfer learning and reusing prior knowledge as much as possible (Agarwal et al., 2022) are becoming more attractive. However, the fine-tuning strategy massively popular in supervised learning (Bommasani et al., 2021; Yosinski et al., 2014; Girshick et al., 2014) is relatively less common in reinforcement learning. Approaches that are often used instead include kickstarting without transferring the parameters (Schmitt et al., 2018; Lee et al., 2022a), and reusing offline data (Lee et al., 2022b; Kostrikov et al., 2021), skills (Pertsch et al., 2021) or the feature representations (Schwarzer et al., 2021; Stooke et al., 2021), see Wulfmeier et al. (2023) for a thorough discussion.

Fine-tuning in RL is often accompanied by knowledge retention mechanisms, even though they are sometimes not described as such. In particular, Baker et al. (2022) includes a regularization term to limit forgetting, Kumar et al. (2022) mixes new data with the old data, and Seo et al. (2022) introduces modularity to the model. Here, we focus on the characterization and the experimental analysis of this issue in fine-tuning RL models, and pinpointing some specific settings when forgetting might occur, such as imperfect cloning gap.

Offline to Online Reinforcement Learning Recent work explored techniques for efficiently transitioning from offline to online reinforcement learning. Ball et al. (2023) use symmetric sampling of offline and online data and combine it with layer normalization and ensembles in an off-policy setting. Lee et al. (2022c) propose using a network for measuring "online-ness" of data and prioritizing samples in a replay buffer according to that measure. Nakamoto et al. (2024) modify Conservative Q-Learning to train on a mixture of the offline data and the new online data, weighted in some proportion during fine-tuning. We highlight that mixing new data with old data can be viewed as a knowledge retention technique similar to Episodic Memory. Although these approaches are relevant to our study and we see testing them as important future work, we use behavioral cloning in pre-training for simplicity, especially as it has been shown to outperform offline RL methods in the NetHack domain (Hambro et al., 2022b).

Impact of interdependence between FAR and CLOSE The relation between FAR and CLOSE states has an impor-

tant impact on the degree of forgetting, which might be understood through the lens of CL literature on task similarity. For example, Lee et al. (2021) find that intermediate task similarity levels lead to the highest degrees of forgetting. Evron et al. (2022) reach a similar conclusion in the linear regression setting when a given task is seen only once, but also find that high similarity causes most forgetting when repeatedly revisiting tasks. Furthermore, Evron et al. (2024) suggest that this behavior might be explained by heavy overparameterization, since in non-overparameterized cases forgetting grows monotonically as the task difference increases.

Generalization to multi-task setting While our work focuses on single-task fine-tuning, prior research has explored fine-tuning on multiple unseen tasks. Yang et al. (2023) compared offline RL methods with imitation learning in a 2D goal-reaching aiming to test generalization to unseen goals. Mandi et al. (2022) showed that multi-task pre-training with fine-tuning often outperformed meta-reinforcement learning approaches in adaptation tasks with high task diversity and strictly unseen test tasks. At the same time, we believe studying single-task fine-tuning in NetHack provides valuable preliminary insights into this problem, as the game's procedural generation on each run requires flexibly applying learned skills to adapt to new contexts.

Continual reinforcement learning Continual RL deals with learning over a changing stream of tasks represented as MDPs (Khetarpal et al., 2022; Wołczyk et al., 2021; Nekoei et al., 2021; Powers et al., 2022; Huang et al., 2021; Kessler et al., 2022a). Several works propose methods for continual reinforcement learning based on replay and distillation (Rolnick et al., 2019; Traoré et al., 2019), or modularity (Mendez et al., 2022; Gaya et al., 2022). Although relevant to our study, these works usually investigate changes in the dynamics of non-stationary environments. In this paper, we switch the perspective and focus on the data shifts occurring during fine-tuning in a stationary environment. In fact, some of the standard techniques in RL, such as using the replay buffer, can be seen as a way to tame the non-stationarity inherent to RL (Lin, 1992; Mnih et al., 2013). For a further discussion about how our setup differs from continual reinforcement learning, see Section 2.

7. Limitations & Conclusions

This study shows that forgetting of pre-trained capabilities is a crucial consideration for fine-tuning RL models. Namely, we verify in multiple scenarios, ranging from toy MDPs to the challenging NetHack domain, that fine-tuning a model on a task where the states from pre-training are not available at the beginning of the training might lead to a rapid deterioration of the prior knowledge. We highlight two specific cases: state coverage gap and imperfect cloning gap.

Although we aim to comprehensively describe forgetting of pre-trained capabilities, our study is limited in several ways. In our experiments, we used fairly simple knowledge retention methods to illustrate the forgetting problem. We believe that CL offers numerous more sophisticated methods that should achieve great results on this problem (Mallya & Lazebnik, 2018; Ben-Iwhiwhu et al., 2022; Mendez et al., 2022; Khetarpal et al., 2022). Additionally, we note that knowledge retention methods can be harmful if the pre-trained policy is suboptimal since they will stop the fine-tuned policy from improving. In some environments, it might not be easy to identify the part of the state space where the policy should be preserved. Furthermore, we focus on two specific transfer scenarios, while in the real world, there are many more settings exhibiting unique problems. Finally, we do not study very large models (i.e. over $1B$ parameters) and efficient approaches to fine-tuning that tune only selected parameters (Xu et al., 2023; Hu et al., 2021). We see all these topics as important directions for future work.

While our study focuses on the RL setting, some of its findings might have a broader scope. Non-stationary dynamics might also emerge in the supervised learning i.i.d. setting when the model sequentially acquires increasingly sophisticated skills (e.g., LLMs first learn simple grammar and understand advanced skills only much later) (Evanson et al., 2023; Luo et al., 2023). This suggests that the principles of knowledge retention and forgetting we explored could be relevant beyond the specific RL scenarios we tested, potentially impacting a wide range of learning systems that evolve over time. A comprehensive examination of these dynamics across different learning models and environments remains a crucial area for future research.

Impact statement

Our main focus is improving the transfer capabilities of reinforcement learning models. We do not foresee any major societal impact of this study that we feel should be highlighted here.

Acknowledgements

The work of MO and MB was funded by National Science Center Poland under the grant agreement 2020/39/B/ST6/01511 and by Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme. PM was supported by National Science Center Poland under the grant agreement 2019/35/O/ST6/03464. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Centers: ACK Cyfronet AGH) for providing com-

puter facilities and support within computational grant no. PLG/2023/016286

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. *arXiv preprint arXiv:2206.01626*, 2022.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 139–154, 2018.

Bain, M. and Sammut, C. A framework for behavioural cloning. In *Machine Intelligence 15*, pp. 103–129, 1995.

Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *arXiv preprint arXiv:2206.11795*, 2022.

Ball, P. J., Smith, L., Kostrikov, I., and Levine, S. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pp. 1577–1594. PMLR, 2023.

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

Ben-Iwhiwhu, E., Nath, S., Pilly, P. K., Kolouri, S., and Soltoggio, A. Lifelong reinforcement learning with modulating masks. *arXiv preprint arXiv:2212.11110*, 2022.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Bornschein, J., Galashov, A., Hemsley, R., Rannen-Triki, A., Chen, Y., Chaudhry, A., He, X. O., Douillard, A., Caccia, M., Feng, Q., et al. Nevis'22: A stream of 100 tasks sampled from 30 years of computer vision research. *arXiv preprint arXiv:2211.11747*, 2022.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *International Conference On Learning Representations*, 2018.

Buzzega, P., Boschini, M., Porrello, A., and Calderara, S. Rethinking experience replay: a bag of tricks for continual learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 2180–2187. IEEE, 2021.

Ceron, J. S. O. and Castro, P. S. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning*, pp. 1373–1383. PMLR, 2021.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Chithrananda, S., Grand, G., and Ramsundar, B. Chemberta: Large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885*, 2020.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Evanson, L., Lakretz, Y., and King, J.-R. Language acquisition: do children and language models follow similar learning stages? *arXiv preprint arXiv:2306.03586*, 2023.

Evron, I., Moroshko, E., Ward, R., Srebro, N., and Soudry, D. How catastrophic can catastrophic forgetting be in linear regression? In *Conference on Learning Theory*, pp. 4028–4079. PMLR, 2022.

Evron, I., Goldfarb, D., Weinberger, N., Soudry, D., and Hand, P. The joint effect of task similarity and overparameterization on catastrophic forgetting–an analytical model. *arXiv preprint arXiv:2401.12617*, 2024.

Gaya, J.-B., Doan, T., Caccia, L., Soulier, L., Denoyer, L., and Raileanu, R. Building a subspace of policies for scalable continual learning. *arXiv preprint arXiv:2211.10445*, 2022.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on*

computer vision and pattern recognition, pp. 580–587, 2014.

Gretton, A., Bousquet, O., Smola, A., and Schölkopf, B. Measuring statistical dependence with hilbert-schmidt norms. In *Algorithmic Learning Theory: 16th International Conference, ALT 2005, Singapore, October 8-11, 2005. Proceedings 16*, pp. 63–77. Springer, 2005.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Hambro, E., Mohanty, S., Babaev, D., Byeon, M., Chakraborty, D., Grefenstette, E., Jiang, M., Daejin, J., Kanervisto, A., Kim, J., et al. Insights from the neurips 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 41–52. PMLR, 2022a.

Hambro, E., Raileanu, R., Rothermel, D., Mella, V., Rocktäschel, T., Küttler, H., and Murray, N. Dungeons and data: A large-scale nethack dataset. *Advances in Neural Information Processing Systems*, 35:24864–24878, 2022b.

Hambro, E., Raileanu, R., Rothermel, D., Mella, V., Rocktäschel, T., Kuttler, H., and Murray, N. Dungeons and data: A large-scale nethack dataset. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022c.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Huang, Y., Xie, K., Bharadhwaj, H., and Shkurti, F. Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 799–805. IEEE, 2021.

Kemker, R., McClure, M., Abitino, A., Hayes, T., and Kanan, C. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Kessler, S., Miłoś, P., Parker-Holder, J., and Roberts, S. J. The surprising effectiveness of latent world models for continual reinforcement learning. *arXiv preprint arXiv:2211.15944*, 2022a.

Kessler, S., Parker-Holder, J., Ball, P., Zohren, S., and Roberts, S. J. Same state, different task: Continual reinforcement learning without interference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7143–7151, 2022b.

Kessler, S., Ostaszewski, M., Bortkiewicz, M., Żarski, M., Wołczyk, M., Parker-Holder, J., Roberts, S. J., and Miłoś, P. The effectiveness of world models for continual reinforcement learning, 2023.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Klissarov, M., D'Oro, P., Sodhani, S., Raileanu, R., Bacon, P.-L., Vincent, P., Zhang, A., and Henaff, M. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pp. 3519–3529. PMLR, 2019.

Kornblith, S., Chen, T., Lee, H., and Norouzi, M. Why do better loss functions lead to less transferable features? *Advances in Neural Information Processing Systems*, 34: 28648–28662, 2021.

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Kumar, A., Singh, A., Ebert, F., Yang, Y., Finn, C., and Levine, S. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *arXiv preprint arXiv:2210.05178*, 2022.

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Lee, A. X., Devin, C., Springenberg, J. T., Zhou, Y., Lampe, T., Abdolmaleki, A., and Bousmalis, K. How to spend your robot time: Bridging kickstarting and offline reinforcement learning for vision-based robotic manipulation.

In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2468–2475. IEEE, 2022a.

Lee, K.-H., Nachum, O., Yang, M., Lee, L., Freeman, D., Xu, W., Guadarrama, S., Fischer, I., Jang, E., Michalewski, H., et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022b.

Lee, S., Goldt, S., and Saxe, A. Continual learning in the teacher-student setup: Impact of task similarity. In *International Conference on Machine Learning*, pp. 6109–6119. PMLR, 2021.

Lee, S., Seo, Y., Lee, K., Abbeel, P., and Shin, J. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pp. 1702–1712. PMLR, 2022c.

Lesort, T., Ostapenko, O., Misra, D., Arefin, M. R., Rodríguez, P., Charlin, L., and Rish, I. Scaling the number of tasks in continual learning. *arXiv preprint arXiv:2207.04543*, 2022.

Lin, L.-J. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J., and Zhang, Y. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018a.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018b.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Mandi, Z., Abbeel, P., and James, S. On the effectiveness of fine-tuning versus meta-reinforcement learning. *arXiv preprint arXiv:2206.03271*, 2022.

Mendez, J. A., van Seijen, H., and Eaton, E. Modular lifelong reinforcement learning via neural composition. *arXiv preprint arXiv:2207.00429*, 2022.

Mirzadeh, S. I., Chaudhry, A., Yin, D., Nguyen, T., Pascanu, R., Gorur, D., and Farajtabar, M. Architecture matters in continual learning. *arXiv preprint arXiv:2202.00275*, 2022.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mu, J., Zhong, V., Raileanu, R., Jiang, M., Goodman, N., Rocktäschel, T., and Grefenstette, E. Improving intrinsic exploration with language abstractions. *Advances in Neural Information Processing Systems*, 35:33947–33960, 2022.

Nair, A., Gupta, A., Dalal, M., and Levine, S. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Nakamoto, M., Zhai, S., Singh, A., Sobol Mark, M., Ma, Y., Finn, C., Kumar, A., and Levine, S. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

Nekoei, H., Badrinaaraayanan, A., Courville, A., and Chandar, S. Continuous coordination as a realistic scenario for lifelong learning. In *International Conference on Machine Learning*, pp. 8016–8024. PMLR, 2021.

NetHack DevTeam. NetHack Home Page. https://nethackwiki.com/wiki/DevTeam, 1987. Accessed: 2023-05-04.

Neyshabur, B., Sedghi, H., and Zhang, C. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020.

Ostapenko, O., Rodriguez, P., Caccia, M., and Charlin, L. Continual learning via local module composition. *Advances in Neural Information Processing Systems*, 34: 30298–30312, 2021.

Pardo, F., Tavakoli, A., Levdik, V., and Kormushev, P. Time limits in reinforcement learning. In *International Conference on Machine Learning*, 2017.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Pertsch, K., Lee, Y., Wu, Y., and Lim, J. J. Guided reinforcement learning with learned skills. *arXiv preprint arXiv:2107.10253*, 2021.

Petrenko, A., Huang, Z., Kumar, T., Sukhatme, G. S., and Koltun, V. Sample factory: Egocentric 3d control from

pixels at 100000 fps with asynchronous reinforcement learning. *ArXiv*, abs/2006.11751, 2020.

Piterbarg, U., Pinto, L., and Fergus, R. Nethack is hard to hack. *arXiv preprint arXiv:2305.19240*, 2023.

Powers, S., Xing, E., Kolve, E., Mottaghi, R., and Gupta, A. Cora: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. In *Conference on Lifelong Learning Agents*, pp. 705–743. PMLR, 2022.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Ramasesh, V. V., Dyer, E., and Raghu, M. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*, 2020.

Ramasesh, V. V., Lewkowycz, A., and Dyer, E. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations*, 2022.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., and Wayne, G. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Ross, S. and Bagnell, D. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, 2010.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Rusu, A. A., Flennerhag, S., Rao, D., Pascanu, R., and Hadsell, R. Probing transfer in deep reinforcement learning without task engineering. In *Conference on Lifelong Learning Agents*, pp. 1231–1254. PMLR, 2022.

Sandler, M., Zhmoginov, A., Vladymyrov, M., and Jackson, A. Fine-tuning image transformers using learnable memory. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12155–12164, 2022.

Schaul, T., Borsa, D., Modayil, J., and Pascanu, R. Ray interference: a source of plateaus in deep reinforcement learning, 2019.

Schmitt, S., Hudson, J. J., Zidek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Kuttler, H., Zisserman, A., Simonyan, K., et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Schwarzer, M., Rajkumar, N., Noukhovitch, M., Anand, A., Charlin, L., Hjelm, R. D., Bachman, P., and Courville, A. C. Pretraining representations for data-efficient reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12686–12699, 2021.

Seo, Y., Lee, K., James, S. L., and Abbeel, P. Reinforcement learning with action-free pre-training from videos. In *International Conference on Machine Learning*, pp. 19561–19579. PMLR, 2022.

Stooke, A., Lee, K., Abbeel, P., and Laskin, M. Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pp. 9870–9879. PMLR, 2021.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Traoré, R., Caselles-Dupré, H., Lesort, T., Sun, T., Cai, G., Díaz-Rodríguez, N., and Filliat, D. Discorl: Continual reinforcement learning via policy distillation. *arXiv preprint arXiv:1907.05855*, 2019.

Tuyls, J., Madeka, D., Torkkola, K., Foster, D., Narasimhan, K., and Kakade, S. Scaling laws for imitation learning in nethack. *arXiv preprint arXiv:2307.09423*, 2023.

Veniat, T., Denoyer, L., and Ranzato, M. Efficient continual learning with modular networks and task-driven priors. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.

Wołczyk, M., Zając, M., Pascanu, R., Kuciński, Ł., and Miłoś, P. Continual world: A robotic benchmark for

continual reinforcement learning. *Advances in Neural Information Processing Systems*, 34:28496–28510, 2021.

Wolczyk, M., Zając, M., Pascanu, R., Kuciński, Ł., and Miłoś, P. Disentangling transfer in continual reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022.

Wulfmeier, M., Byravan, A., Bechtle, S., Hausman, K., and Heess, N. Foundations for transfer in reinforcement learning: A taxonomy of knowledge modalities, 2023.

Xu, L., Xie, H., Qin, S.-Z. J., Tao, X., and Wang, F. L. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*, 2023.

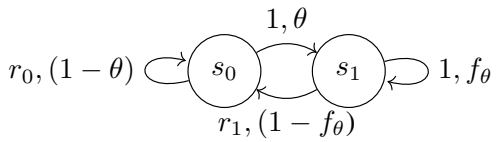
Yang, R., Yong, L., Ma, X., Hu, H., Zhang, C., and Zhang, T. What is essential for unseen goal generalization of offline goal-conditioned rl? In *International Conference on Machine Learning*, pp. 39543–39571. PMLR, 2023.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

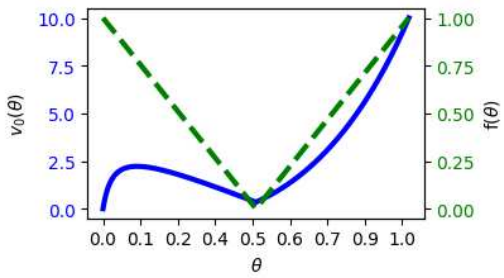
Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.

Zhang, Y., Park, D. S., Han, W., Qin, J., Gulati, A., Shor, J., Jansen, A., Xu, Y., Huang, Y., Wang, S., et al. Bigssl: Exploring the frontier of large-scale semi-supervised learning for automatic speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1519–1532, 2022.

Zheng, H., Luo, X., Wei, P., Song, X., Li, D., and Jiang, J. Adaptive policy learning for offline-to-online reinforcement learning. *arXiv preprint arXiv:2303.07693*, 2023.

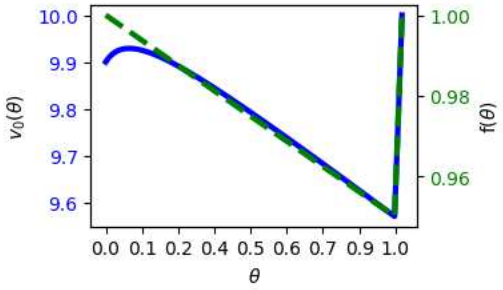


(a) MDP with two states with $\gamma = 0.9$.



z

(b) Value of s_0 (blue) and f_θ (green) for $r_0 = 0$, $r_1 = -1$, and $f_\theta = 2|\theta - 0.5|$.



(c) Value of s_0 (blue) and f_θ (green) for $r_0 = 0.99$, $r_1 = 0$, $\epsilon = 0.05$, $f_\theta = 1 - \frac{\epsilon\theta}{1-\epsilon/2}$ for $\theta \leq 1 - \epsilon/2$ and $f_\theta = 2\theta - 1$ for $\theta > 1 - \epsilon/2$.

Figure 9: (a) A toy two-state MDP. Each arrow depicts a transition between states, and the annotation encodes the reward and the probability of transition from the policy. (b,c) A policy with its corresponding value function $v_0(\theta)$, for two variants of parameterization and reward functions.

A. Toy Examples – MDP and AppleRetrieval

In the main text, we showed empirically that forgetting of pre-trained capabilities appears in standard RL scenarios. Here, we additionally provide two toy environments: two-state MDPs and a simple grid-world called AppleRetrieval. We find these environments to be helpful for understanding the core of the problem and for building intuition.

A.1. Two-state MDPs

In this subsection, we show that the two scenarios of forgetting of pre-trained capabilities, state coverage gap and imperfect cloning gap can happen even in a very simple 2-state MDP. This observation fits well into the RL tradition of showing counterexamples on small MDPs (Sutton & Barto, 2018). The MDP, shown in Figure 9(a), consists of two states, labeled as s_0 and s_1. The transition between states is stochastic and is indicated by an arrow annotated by a reward and transition probability. For example, a transition from s_1 to s_0 happens with probability $1 - f_\theta$ and grants a reward r_1. The value of state s_0, visualized as a blue line in Figure 9(b) and 9(c), equals

$$v_0(\theta) = \frac{1}{1-\gamma} \frac{\theta + r_0(1-\theta)(1-\gamma f_\theta) + \gamma\theta r_1(1-f_\theta)}{1 - \gamma f_\theta + \gamma\theta}.$$

In each case, we treat fine-tuning as the process of adjusting θ towards the gradient direction of $v_0(\theta)$ until a local extremum is encountered. We now consider two parameterizations of this MDP that represent state coverage gap and imperfect cloning gap.

State coverage gap In Figure 9(b), we present a state coverage gap scenario, where we fine-tune a policy that was pre-trained on a subset of downstream states and we show that it can lead to divergence. We parameterize the policy as:

$$f_\theta = \left(\frac{-\epsilon}{1-\epsilon/2}\theta + 1\right) \mathbf{1}_{\theta \leq 1-\epsilon/2} + (2\theta - 1)\, \mathbf{1}_{\theta > 1-\epsilon/2}.$$

Here, we have an MDP where the initial policy $\theta = 0$ was trained only on state s_1. Since $f_0 = 1$, such a policy stays in s_1 once it starts from s_1.

If we now try to fine-tune this policy where the starting state is s_0, the agent will forget the behavior in s_1 due to the interference caused by the parametrization of the policy. This in turn will lead the system to converge to a suboptimal policy $\theta = 0.11$ with a value of 2.22. In this case, the environment has changed by introducing new states that need to be traversed to reach states on which we know how to behave. Learning on these new states that are visited early on will lead to forgetting of the pre-trained behavior.

Imperfect cloning gap Subsequently, in Figure 9(c), we provide an example of imperfect cloning gap. The policy is parametrized as

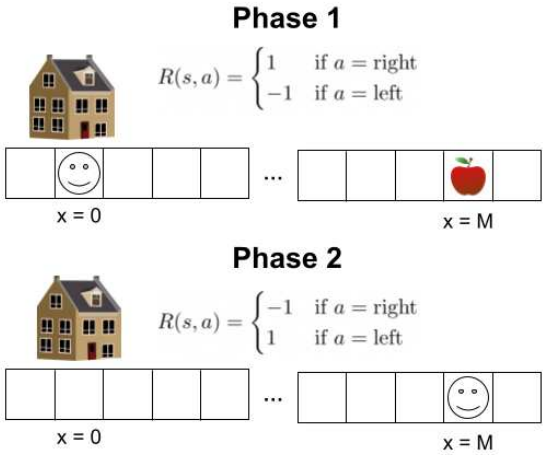
$$f_\theta = 2|\theta - 0.5|.$$

In this scenario, $\theta = 1$ (with $f_1 = 1$) represents the optimal behavior of staying in s_1 and achieving maximum total discounted returns equal to 10. However, for a given parametrization of f_θ, this maximum can be unstable, and adding a small noise ϵ to θ before fine-tuning will lead to divergence towards a local maximum at $\theta = 0.08$ with the corresponding value 9.93. Perturbing θ by ϵ will make the system visit s_0 more often, and learning on s_0 with further push θ away from 1, forgetting the skill of *moving to and staying in* s_1.

A.2. Synthetic example: APPLERETRIEVAL

Additionally, we introduce a synthetic example of an environment exhibiting state coverage gap, dubbed APPLERETRIEVAL. We will show that even a vanilla RL algorithm with linear function approximators shows forgetting of pre-trained capabilities.

APPLERETRIEVAL is a 1D gridworld, consisting of two phases. In Phase 1, starting at home: $x = 0$, the agent has to go to $x = M$ and retrieve an apple, $M \in \mathbb{N}$. In Phase 2, the agent has to go back to $x = 0$. In each phase, the reward is 1 for going in the correct direction and -1 otherwise. The observation is $o = [-c]$ in Phase 1 and $o = [c]$ in Phase 2, for some $c \in \mathbb{R}$; i.e. it encodes the information about the current phase. Given this observation, it is now trivial to encode the optimal policy: go right in Phase 1 and go left in Phase 2. Episodes are terminated if the solution is reached or after 100 timesteps. Since we can only get to Phase 2 by completing Phase 1, this corresponds to dividing the states to sets CLOSE and FAR, as described in Section 2.



APPLERETRIEVAL environment.

We run experiments in APPLERETRIEVAL using the REINFORCE algorithm (Williams, 1992) and assume a simple model in which the probability to move right is given by: $\pi_{w,b}(o) = \sigma(w \cdot o + b), w, b \in \mathbb{R}$. Importantly, we initialize w, b with the weights trained in Phase 2.

We show experimentally, see Figure 10, that for high enough distance M, the forgetting of pre-trained capabilities problem appears. Intuitively, the probability of concluding Phase 1 becomes small enough that the pre-trained Phase 2 policy is forgotten, leading to overall poor performance. In this simple case, we can mechanically analyze this process of forgetting.

Since the linear model in APPLERETRIEVAL has only two parameters (weight w, bias b) we can analyze and understand what parameter sets lead to forgetting. If the pre-trained policy mostly relies on weight (i.e. $|w| \gg |b|$) then the interference will be limited. However, if the model relies on bias (i.e. $|b| \gg |w|$) then interference will occur as bias will impact the output in the same way in both phases. We can guide the model towards focusing on one or the other by setting the c parameter since the linear model trained with gradient descent will tend towards a solution with a low weight norm. The results presented in Figure 11 confirm our hypothesis, as lower values of c encourage models to rely more on b which leads to forgetting. Such a low-level analysis is infeasible for deep neural networks, but experimental results confirm that interference occurs in practice (Kirkpatrick et al., 2017; Kemker et al., 2018; Ramasesh et al., 2022).

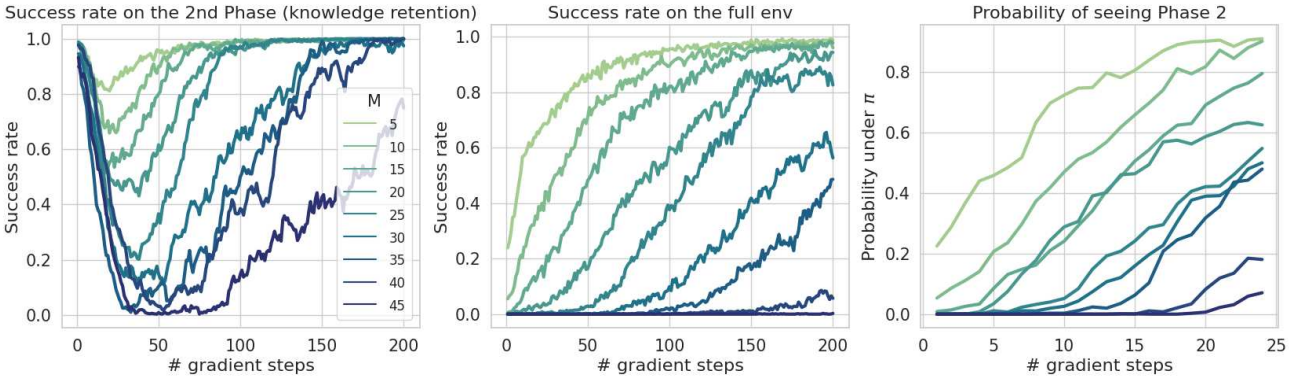


Figure 10: Forgetting of pre-trained capabilities in APPLERETRIEVAL. (Left) Forgetting becomes more problematic as M (the distance from the house to the apple) increases and (center) hinders the overall performance. (Right, note x-scale change) This happens since the probability of reaching Phase 2 in early training decreases.

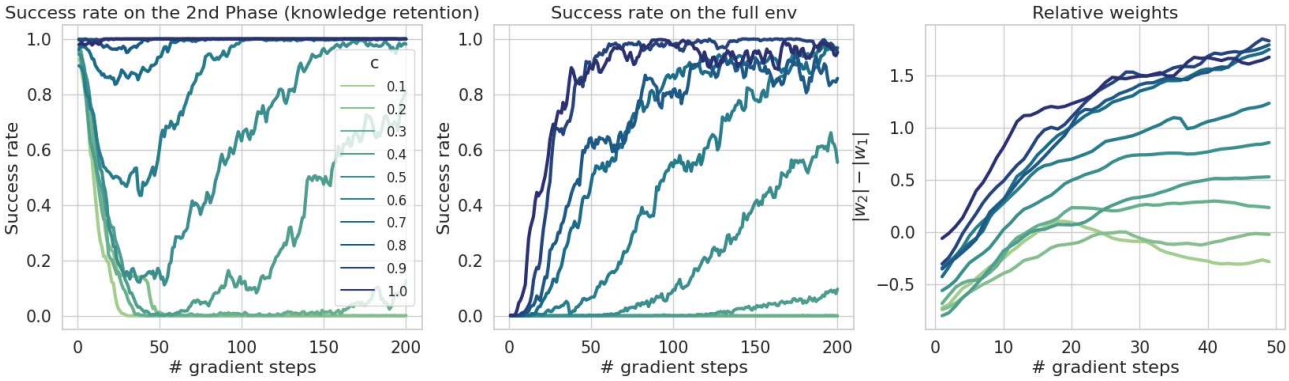


Figure 11: Impact of c on the results for set $M = 30$. For smaller c forgetting (left) is greater and the overall success rate is smaller (center) since it encourages the pre-trained model to find solutions with a high $\frac{|b|}{|w|}$ ratio, as confirmed by looking at weight difference early in fine-tuning (right).

B. Technical details

B.1. NetHack

Environment NetHack (NetHack DevTeam, 1987) is a classic and highly complex terminal roguelike game that immerses players in a procedurally generated dungeon crawling experience, navigating through a labyrinth in a world filled with monsters, treasures, and challenges. The NetHack Learning Environment (NLE) introduced in (Küttler et al., 2020) is a scalable, procedurally generated, stochastic, rich, and challenging environment aimed to drive long-term research on problems such as exploration, planning, skill acquisition, and language-conditioned RL.

The NLE is characterized by a state space that includes a 2D grid representing the game map and additional information like the player's inventory, health, and other statistics. Thus, the NLE is multimodal and consists of an image, the main map screen, and text. The action space in NLE consists of a set of 120 discrete actions. At the same time, the NLE presents a challenge for RL agents due to its action-chaining behavior. For instance, the player must press three distinct keys in a specific sequence to throw an item, which creates additional complexity to the RL problem. The environmental reward in `score` task, used in this paper, is based on the increase in the in-game score between two-time steps. A complex calculation determines the in-game score. However, during the game's early stages, the score is primarily influenced by factors such as killing monsters and the number of dungeon levels the agent explores. The in-game score is a sensible proxy for incremental progress on NLE. Still, training agents to maximize it is likely not perfectly aligned with solving the game, as expert human players can solve NetHack while keeping the score low. In each run, the dungeon is generated anew, so the agent does not ever see a specific level twice. Consequently, we can't expect the agent to remember solutions to specific levels, but rather, we aim for it to recall general behavioral patterns for different levels.

It is important to note that during training, the agent may not follow levels in a linear sequence due to NetHack's allowance for backtracking or branching to different dungeon parts (as described in `https://nethackwiki.com/wiki/Branch`). This highlights the issue of forgetting, even in the absence of strictly defined linear tasks or stages, contrary to the continual learning literature.

Architecture We fine-tune the model pre-trained by Tuyls et al. (2023), which scales up (from 6M to 33M parameters) and modifies the solution proposed by the 'Chaotic Dwarven GPT-5' team, which is based on Sample Factory (Petrenko et al., 2020) that was also used in (Hambro et al., 2022c). This model utilizes an LSTM architecture that incorporates representations from three encoders, which take observations as inputs. The LSTM network's output is then fed into two separate heads: a policy head and a baseline head. The model architecture used both in online and offline settings consists of a joint backbone for both actor and critic. It takes as an input three components: main observation of the dungeon screen, `blstats`, and `message`. `blstats` refers to the player's status information, such as health and hunger, and `message` refers to the textual information displayed to the player, such as notifications and warnings. `blstats` and `message` are processed using two layer MLP. The main observation of the dungeon screen is processed by embedding each character and color in an embedding lookup table which is later put into a grid processed by ResNet. For more details about Main screen encoder refer to (Tuyls et al., 2023). The components are encoded, and are merged before passing to LSTM. This baseline allows for fast training but struggles with learning complex behaviours required for certain roles in the game. More details about the architecture can be found in (Tuyls et al., 2023; Petrenko et al., 2020). The model hyperparameters are shown in Table 1 – analogical to Table 6 from (Petrenko et al., 2020).

Dataset The knowledge retention methods presented in this paper use a subset of the NetHack Learning Dataset (NLD) collected by (Hambro et al., 2022c) called NLD-AA. It contains over 3 billion state-action-score transitions and metadata from 100,000 games collected from the winning bot of the NetHack Challenge (Hambro et al., 2022a). In particular, we use about 8000 games of Human Monk. This character was chosen because it was extensively evaluated in the previous work (Hambro et al., 2022c) and because the game setup for the Human Monk is relatively straightforward, as it does not require the agent to manage the inventory. The bot is based on the 'AutoAscend' team solution, a symbolic agent that leverages human knowledge and hand-crafted heuristics to progress in the game. Its decision-making module is based on a behavior tree model.

The checkpoint we use as the pre-trained policy π_* was trained by Tuyls et al. (2023) on a larger set of trajectories from the AutoAscend agent, containing over $115B$ transitions.

Pre-training As for the offline pre-training phase, we used a model trained with Behavioral Cloning (BC) (Bain & Sammut, 1995; Ross & Bagnell, 2010) by (Tuyls et al., 2023), an imitation learning approach that utilizes a supervised learning objective to train the policy to mimic the actions present in the dataset. To be more specific, it utilizes a cross-entropy loss function between the policy action distribution and the actions from the NLD-AA dataset. For more details on hyperparameters, please refer to the original article (Tuyls et al., 2023). It should be noted that BC does not include a critic. To improve stability during the beginning of the fine-tuning we additionally pre-train the baseline head by freezing the rest of the model for 500M environment steps.

Fine-tuning In the online training phase, we employed a highly parallelizable architecture called Asynchronous Proximal Policy Optimization (APPO) (Schulman et al., 2017; Petrenko et al., 2020). In this setup, we can run over 500 million environment steps under 24 hours of training on A100 Nvidia GPU. Within the main manuscript, we examined vanilla fine-tuning and fine-tuning with a behavioral cloning loss, kickstarting and EWC, explained in more detail in Appendix C.

In Fine-tuning + KS we compute the auxiliary loss on data generated by the online policy. We scaled the loss by a factor of 0.5 and used exponential decay 0.99998, where the coefficient was decayed every train step. In Fine-tuning + BC we compute the auxiliary loss by utilizing the trajectories generated by the expert (i.e. the AutoAscend algorithm), note that no decay was used here. We scaled the auxiliary loss by a factor of 2.0. To improve the stability of the models we froze the encoders during the course of the training. Additionally, we turn off entropy when employing knowledge retention methods in similar fashion to (Baker et al., 2022). For EWC we use a regularization coefficient of $2 \cdot 10^6$.

Evaluation During the evaluation phase, we provide the in-game score achieved and the number of filled pits for Sokoban levels at specific checkpoints during training. Models were evaluated every 25 million environment steps for Figure 5. To perform the per-level evaluation in Figure 5, we employ the AutoAscend expert, used for behavioral cloning in pre-training. We use AutoAscend to play the game and save the state when it reaches the desired level. We generate 200 game saves for each level and evaluate our agents on each save by loading the game, running our agent where the expert finished, and reporting the score our agent achieved on top of the expert's score.

Table 1: Hyperparameters of the model used in NLE. For the most part, we use hyperparameters values from (Hambro et al., 2022c).

Hyperparameter Name	Value
activation_function	relu
adam_beta1	0.9
adam_beta2	0.999
adam_eps	0.0000001
adam_learning_rate	0.0001
weight_decay	0.0001
appo_clip_policy	0.1
appo_clip_baseline	1.0
baseline_cost	1
discounting	0.999999
entropy_cost	0.001
grad_norm_clipping	4
hidden_dim	1738
batch_size	128
penalty_step	0.0
penalty_time	0.0
reward_clip	10
reward_scale	1
unroll_length	32

B.2. Montezuma's Revenge

Environment In this section, we provide further details on our experiments with Montezuma's Revenge from Atari Learning Environment (ALE) (Machado et al., 2018b). Montezuma's Revenge, released in 1984, presents a challenging

platformer scenario where players control the adventurer Panama Joe as he navigates a labyrinthine Aztec temple, solving puzzles and avoiding a variety of deadly obstacles and enemies. What makes Montezuma's Revenge particularly interesting for research purposes is its extreme sparsity of rewards, where meaningful positive feedback is rare and often delayed, posing a significant challenge.

We enumerate rooms according to the progression shown in Figure 12, starting from Room 1, where the player begins gameplay. As a successful completion of the room in Figure 6, we consider achieving at least one of the following options: either earn a coin as a reward, acquire a new item, or exit the room through a different passage than the one we entered through.

Architecture In our experiments, we use a PPO agent with a Random Network Distillation (RND) mechanism (Burda et al., 2018) for exploration boost. It achieves this by employing two neural networks: a randomly initialized target network and a prediction network. Both networks receive observation as an input and return a vector with size 512. The prediction network is trained to predict the random outputs generated by the target network. During interaction with the environment, the prediction network assesses the novelty of states, prioritizing exploration in less predictable regions. States for which the prediction network's predictions deviate significantly from the random targets are considered novel and are prioritized for exploration. Detailed hyperparameter values can be found in Table 2.

Dataset For behavioral cloning purposes, we collected more than 500 trajectories sampled from a pre-trained PPO agent with RND that achieved an episode cumulative reward of around 7000. In Figure 13 we show the impact of different values of the Kullback–Leibler weight coefficient on agent performance.

Table 2: Hyperparameters of the model used in Montezuma's Revenge. For the most part, we use hyperparameter values from (Burda et al., 2018). We used PyTorch implementation by *jcwleo* from https://github.com/jcwleo/random-network-distillation-pytorch

Hyperparameter Name	Value
MaxStepPerEpisode	4500
ExtCoef	2.0
LearningRate	1e-4
NumEnv	128
NumStep	128
Gamma	0.999
IntGamma	0.99
Lambda	0.95
StableEps	1e-8
StateStackSize	4
PreProcHeight	84
ProProcWidth	84
UseGAE	True
UseGPU	True
UseNorm	False
UseNoisyNet	False
ClipGradNorm	0.5
Entropy	0.001
Epoch	4
MiniBatch	4
PPOEps	0.1
IntCoef	1.0
StickyAction	True
ActionProb	0.25
UpdateProportion	0.25
LifeDone	False
ObsNormStep	50

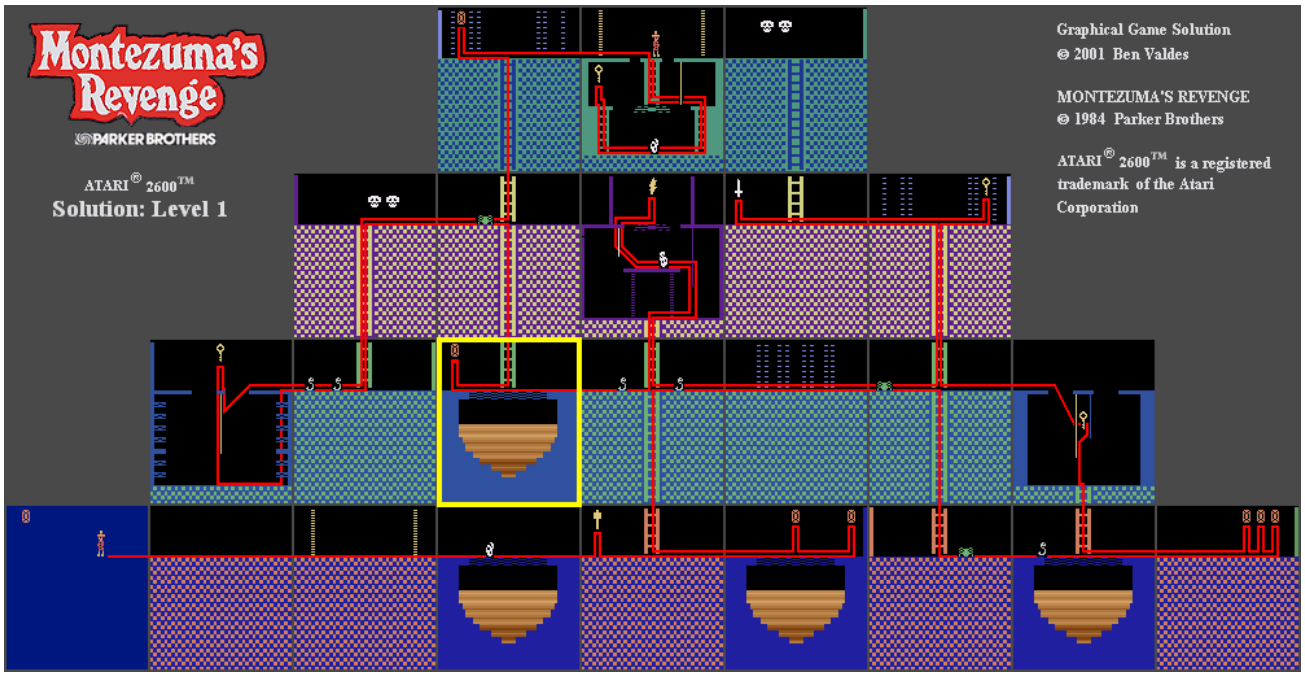


Figure 12: The order in which rooms are visited to complete the first level of Montezuma's Revenge is presented with the red line. We highlight Room 7, which we use for experiments in the mani text, with a yellow border. Source: https://pitfallharry.tripod.com/MapRoom/MontezumasRevengeLvl1.html

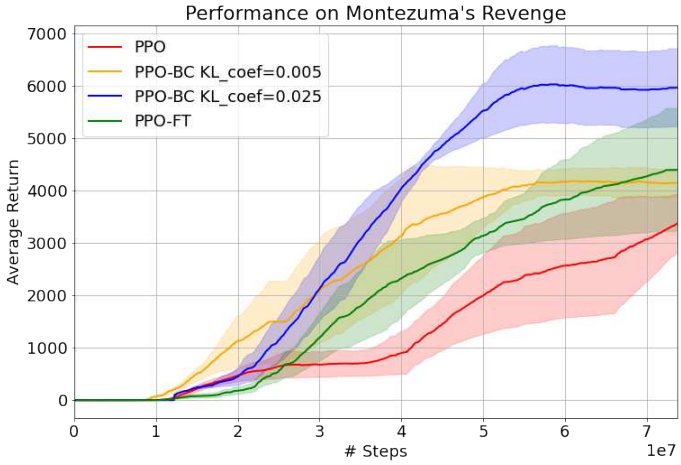


Figure 13: Average return in Montezuma's Revenge for PPO (trained from scratch), fine-tuned PPO and two different coefficients for fine-tuned PPO + BC.

B.3. Meta World

In this section, we describe the `RoboticSequence` setting, and we provide more details about its construction. The algorithm representing `RoboticSequence` construction is presented in Algorithm 1.

We use multi-layer perceptrons (4 hidden layers, 256 neurons each) as function approximators for the policy and Q-value function. For all experiments in this section, we use the Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018a). The observation space consists of information about the current robot configuration, see (Yu et al., 2020) for details, and the

stage ID encoded as a one-hot vector. In our experiments, we use a pre-trained model that we trained with SAC on the last two stages (`peg-unplug-side` and `push-wall`) until convergence (i.e. 100% success rate). All experiments on Meta-World are run with at least 20 seeds and we present the results with 90% confidence intervals. The codebase is available in the supplementary materials.

Algorithm 1 `RoboticSequence`

Input: list of N environments E_k, policy π, time limit T.
Returns: number of solved environments.
$i = 1; t = 1$ {Initialize env idx, timestep counter}
while $i \leq N$ **and** $t \leq T$ **do**
  Take a step in E_i using π
  if E_i is solved **then**
    $i = i + 1; t = 1$ {Move to the next env, reset timestep counter }
  end if
end while
Return $i - 1$

In order to make the problem more challenging, we randomly sample the start and goal conditions, similarly as in (Wołczyk et al., 2021). Additionally, we change the behavior of the terminal states. In the original paper and codebase, the environments are defined to run indefinitely, but during the training, finite trajectories are sampled (i.e. 200 steps). On the 200th step even though the trajectory ends, SAC receives information that the environment is still going. Effectively, it means that we still bootstrap our Q-value target as if this state was not terminal. This is a common approach for environments with infinite trajectories (Pardo et al., 2017).

However, this approach is unintuitive from the perspective of `RoboticSequence`. We would like to go from a given stage to the next one at the moment when the success signal appears, without waiting for an arbitrary number of steps. As such, we introduce a change to the environments and terminate the episode in two cases: when the agent succeeds or when the time limit is reached. In both cases, SAC receives a signal that the state was terminal, which means we do not apply bootstrapping in the target Q-value. In order for the MDP to be fully observable, we append the normalized timestep (i.e. the timestep divided by the maximal number of steps in the environment, $T = 200$ in our case) to the state vector. Additionally, when the episode ends with success, we provide the agent with the "remaining" reward it would get until the end of the episode. That is, if the last reward was originally r_t, the augmented reward is given by $r'_t = \beta r_t(T - t)$. $\beta = 1.5$ is a coefficient to encourage the agent to succeed. Without the augmented reward there is a risk that the policy would avoid succeeding and terminating the episode, in order to get rewards for a longer period of time.

SAC We use the Soft Actor-Critic (Haarnoja et al., 2018a) algorithm for all the experiments on Meta-World and by default use the same architecture as in the Continual World (Wołczyk et al., 2021) paper, which is a 4-layer MLP with 256 neurons each and Leaky-ReLU activations. We apply layer normalization after the first layer. The entropy coefficient is tuned automatically (Haarnoja et al., 2018b). We create a separate output head for each stage in the neural networks and then we use the stage ID information to choose the correct head. We found that this approach works better than adding the stage ID to the observation vector.

For the base SAC, we started with the hyperparameters listed in (Wołczyk et al., 2021) and then performed additional hyperparameter tuning. We set the learning rate to 10^{-3} and use the Adam (Kingma & Ba, 2014) optimizer. The batch size is 128 in all experiments. We use EWC, and BC as described in (Wołczyk et al., 2021; Wolczyk et al., 2022). For episodic memory, we sample 10k state-action-reward tuples from the pre-trained stages using the pre-trained policy and we keep them in SAC's replay buffer throughout the training on the downstream task. Since replay buffer is of size 100k, 10% of the buffer is filled with samples from the prior stages. For each method, we perform a hyperparameter search on method-specific coefficients. Following (Wołczyk et al., 2021; Wolczyk et al., 2022) we do not regularize the critic. The final hyperparameters are listed in Table 3.

CKA We use Central Kernel Alignment (Kornblith et al., 2019) to study similarity of representations. CKA is computed between a pair of matrices, $X \in \mathbb{R}^{n \times p_1}, Y \in \mathbb{R}^{n \times p_2}$, which record, respectively, activations for p_1 and p_2 neurons for the

Table 3: Hyperparameters of knowledge retention methods in Meta-World experiments.

Method	actor reg. coef.	critic reg. coef.	memory
EWC	100	0	-
BC	1	0	10000
EM	-	-	10000

same n examples. The formula is then given as follows:

$$\text{CKA}(K, L) = \frac{\text{HSIC}(K, L)}{\sqrt{\text{HSIC}(K, K)\text{HSIC}(L, L)}}, \tag{1}$$

where HSIC is the Hilbert-Schmidt Independence Criterion (Gretton et al., 2005), $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and $L_{ij} = l(\mathbf{y}_i, \mathbf{y}_j)$, and k and l are two kernels. In our experiments, we simply use a linear kernel in both cases.

Compute For the experiments based on Meta-World, we use CPU acceleration, as the observations and the networks are relatively small and the gains from GPUs are marginal (Wołczyk et al., 2021). For each experiment, we use 8 CPU cores and 30GB RAM. The average length of an experiment is 48 hours. During our research for this paper, we ran over 20,000 experiments on Contiual World.

C. Knowledge retention methods

In this section, we provide more details about the knowledge retention methods used in the experiments, and we briefly describe different types of possible approaches.

In this paper, we mostly focus on fine-tuning only on a single stationary task. However, in continual learning literature that often focuses on the problem of mitigating forgetting, the goal is to usually deal with a sequence of tasks (up to several hundred (Lesort et al., 2022)) and efficiently accumulate knowledge over the whole sequence. As such, although here we will describe knowledge retention methods with two tasks (corresponding to pre-training and fine-tuning), in practice dealing with a longer sequence of tasks might require more careful considerations.

C.1. Regularization-based methods

Regularization-based methods in CL aim to limit forgetting by penalizing changes in parameters that are relevant to the current task. In particular, a few regularization methods (Kirkpatrick et al., 2017; Aljundi et al., 2018) add an auxiliary loss of the following form:

$$\mathcal{L}_{aux}(\theta) = \sum_i F^i(\theta^i_{\text{pre}} - \theta^i)^2, \tag{2}$$

where θ are the weights of the current model, θ_{pre} are the weights of a prior model, and F^i are weighting coefficients. In **Elastic Weight Consolidation (EWC)** (Kirkpatrick et al., 2017) we use in our experiments, F is the diagonal of the Fisher Information Matrix, see (Wołczyk et al., 2021) for details about its implementation in Soft Actor-Critic.

C.2. Distillation-based methods

In this work, we use the behavioral cloning approach used previously in continual reinforcement learning setup (Wolczyk et al., 2022; Rolnick et al., 2019) This approach is based on minimizing the Kullback-Leibler of action distributions under particular states $D^s_{KL}(p \parallel q) = \mathbb{E}_{a\sim p(\cdot|s)}\left[\log(\frac{p(a|s)}{q(a|s)})\right]$. Assume that π_θ is the current policy parameterized by θ (student) and π_* is the pre-trained policy (teacher).

In behavioral cloning, we apply the following loss:

$$\mathcal{L}_{BC}(\theta) = \mathbb{E}_{s\sim\mathcal{B}}[D^s_{KL}(\pi_\theta \parallel \pi_*)], \tag{3}$$

where $\mathcal{B}$ is a buffer of data containing states from pre-training.

In **Kickstarting** (KS) (Schmitt et al., 2018), we use a very similar loss, but now we apply KL on the data gathered online by the student. More formally:

$$\mathcal{L}_{KS}(\theta) = \mathbb{E}_{s\sim\mathcal{B}_\theta}[D^s_{KL}(\pi_*(\cdot|s) \parallel \pi_\theta(\cdot|s))], \tag{4}$$

where $\mathcal{B}_\theta$ denotes a buffer of data gathered by the online policy π_θ.

C.3. Replay-based methods

A simple way to mitigate forgetting is to add the prior data to the training dataset for the current dataset (in supervised learning (Chaudhry et al., 2019; Buzzega et al., 2021)) or to the replay buffer (in off-policy RL (Rolnick et al., 2019; Kessler et al., 2022b)). By mixing the data from the previous and the current task, one approximates the perfectly mixed i.i.d. data distribution, thus going closer to stationary learning.

In our experiments, we use a simple episodic memory (EM) approach along with the off-policy SAC algorithm. At the start of the training, we gather a set of trajectories from the pre-trained environment and we use them to populate SAC's replay buffer. In our experiments, old samples take 10% of the whole buffer size. Then, throughout the training we protect that part of the buffer, i.e. we do not allow the data from the pre-trained task to be overridden.

Although episodic memory performs well in our experiments, it is difficult to use this strategy in settings with on-policy algorithms. In particular, we cannot trivially use it with PPO in Montezuma's Revenge and with APPO in NetHack as these methods do not use a replay buffer and might become unstable when trained with off-policy data. Additionally, we note that episodic memory seems to work poorly with SAC in traditional continual learning settings (Wołczyk et al., 2021; Wolczyk et al., 2022). As such, we focus on the distillation approaches instead.

C.4. Parameter-isolation methods

Standard taxonomies of continual learning (De Lange et al., 2021) also consider parameter isolation-based (or modularity-based) method. Such methods assign a subset of parameters to each task and preserve the performance by keeping these weights frozen. For example, Progressive Networks (Rusu et al., 2016) introduces a new set of parameters with each introduced task, and PackNet (Mallya & Lazebnik, 2018) freezes a subset of existing weights after each task. Recent works showed that by carefully combining the modules, one can achieve a significant knowledge transfer without any forgetting (Veniat et al., 2021; Ostapenko et al., 2021). However, in most cases, methods in this family require access to the task ID. Although we provide the stage ID in our controlled `RoboticSequence` environments, most realistic problems, such as NetHack, do not have clearly separable tasks and as such application of such methods to the general fine-tuning problem might be non-trivial.

C.5. Note on critic regularization

In actor-critic architectures popular in reinforcement learning, one can decide whether to apply knowledge retention methods only to the actor and only to the critic. If all we care about is the policy being able to correctly execute the policies for the previous tasks, then it is enough to force the actor to not forget. Since the critic is only used for training, forgetting in the critic will not directly impact the performance. On the other hand, in principle preserving knowledge in the critic might allow us to efficiently re-train on any of the prior tasks. In this paper, following (Wolczyk et al., 2022) we focus on regularizing only the actor, i.e. we do not apply any distillation loss on the critic in distillation-based methods and we do not minimize the distance on the L2 norm on the critic-specific parameters.

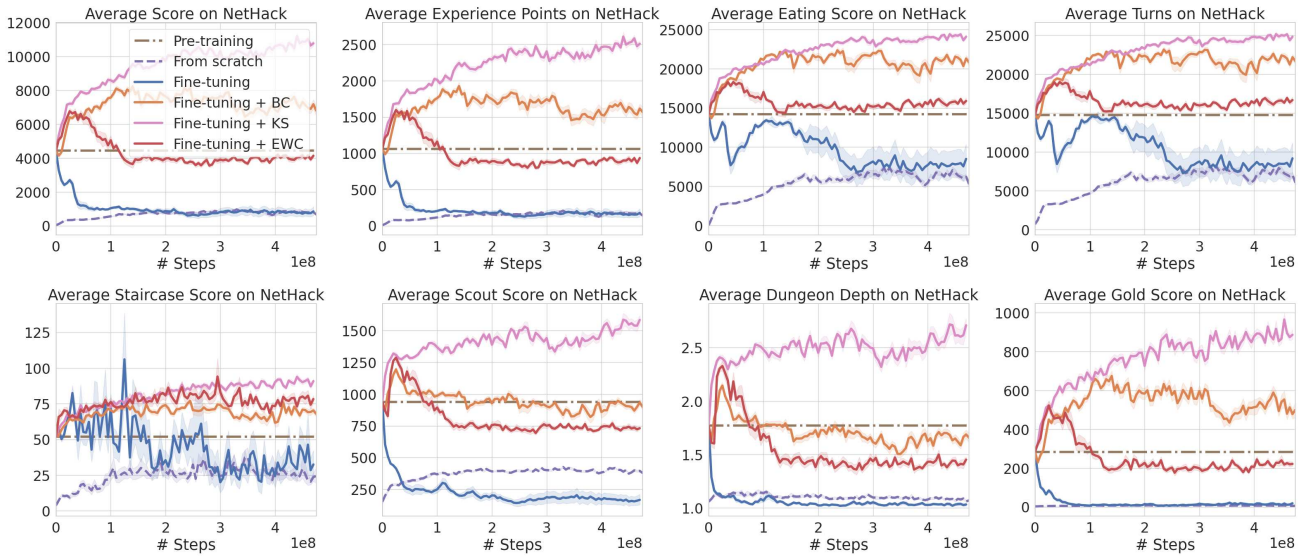


Figure 14: Performance on NetHack on additional metrics. Gold Score, Eating Score, Staircase Score and Scout Score are measured in the same way as additional tasks defined in NLE (Küttler et al., 2020). Score, Turns, Experience Points and Dungeon Depth are taken from blstats. All metrics are measured throughout the training.

D. Additional NetHack results

Additional metrics of NetHack performance In Figure 14, we visualize additional metrics. Some of them were originally introduced as tasks in NLE (Küttler et al., 2020) (Gold Score, Eating Score, Staircase Score, and Scout Score), while the others are displayed at the bottom of the screen as statistics (Score, Turns, Experience Points, and Dungeon Depth). These metrics were measured throughout the training, providing a detailed perspective on the behavior of agents. Indeed, it is evident that knowledge retention methods are crucial for making progress in the game, as fine-tuning + KS achieves the highest score while also being the best in all other metrics that measure progress in the actual game. This observation confirms the importance of score as a reliable proxy for game progress, as methods achieving higher scores almost always outperform others in all additional metrics.

While the previous results were gathered during the training process, in Table 4 we provide different metrics for the full evaluation. Following the community standards (Küttler et al., 2020), we take the last checkpoints of each run and we generate 1000 trajectories from it. The results again show that fine-tuning with knowledge retention methods helps and, in particular, fine-tuning + KS achieves state-of-the-art results throughout all metrics.

Additionally, in Table 5 we position our score results against the prior work.

Table 4: NetHack full evaluation results on last checkpoint of each run for 1000 episodes.

method	score	turns	steps	dlvl	xplvl	eating	gold	scout	sokoban	staircase
From scratch	776	6696	13539	1.06	4.07	5862.56	5.34	370.62	0.00	25.17
Fine-tuning	647	7756	13352	1.02	2.73	7161.20	9.26	149.70	0.00	19.94
Fine-tuning + EWC	3976	16725	35018	1.41	6.29	15896.45	217.12	719.70	0.00	81.74
Fine-tuning + BC	7610	22895	34560	1.7	7.30	21995.63	582.33	959.34	0.00	69.89
Fine-tuning + KS	10588	24436	38635	2.66	7.73	23705.56	857.20	1551.18	0.04	90.10

Table 5: Score comparison of methods from prior work and our best performing method (denoted as Fine-tuning + KS in the main text, here as "Scaled-BC + Fine-tuning + KS" to differentiate the pre-trained model).

Models	**Human Monk**
Offline only	
DQN-Offline (Hambro et al., 2022c)	0.0 ± 0.0
CQL (Hambro et al., 2022c)	366 ± 35
IQL (Hambro et al., 2022c)	267 ± 28
BC (CDGPT5) (Hambro et al., 2022c;a)	1059 ± 159
Scaled-BC (Tuyls et al., 2023)	5218 ± -
Offline + Online	
From Scratch + KS (Hambro et al., 2022c)	2090 ± 123
From Scratch + BC (Hambro et al., 2022c)	2809 ± 103
LDD* (Mu et al., 2022)	2100 ± -
Scaled-BC + Fine-tuning + KS (ours)	**10588 ± 672**

Return density In previous sections we looked at the mean return. Here, to better understand the behavior of the tested methods, we also look at the whole distribution of returns. This way, we can understand whether e.g., the score of a given method relies on a few lucky high-reward runs. The results presented in Figure 15 show that while from scratch and fine-tuning achieve consistently poor returns, the variance in scores is much higher for fine-tuning with knowledge retention. In particular, we observe that there are occurrences of fine-tuning + KS achieving returns as high as 50000. At the same time, there is a significant time of unlucky runs that end with a return of 1000. We can attribute this variance to the high stochasticity of the game, e.g., if the first level happens to contain many monsters that are difficult to defeat, that episode may end earlier than expected.

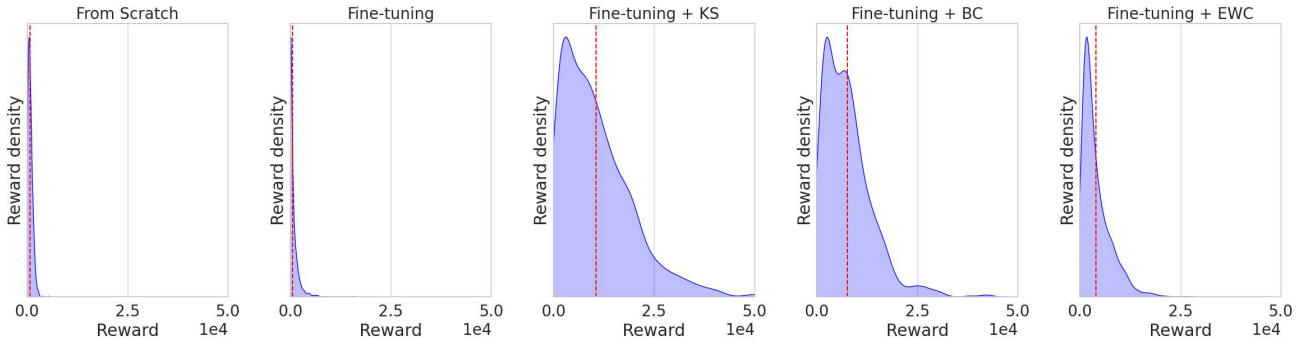


Figure 15: Return distribution for each of the tested methods. The mean return is denoted by the dashed red line.

Level visitation density In Figure 16 we show the level density plots from Figure 4 for all methods. In particular, we observe that fine-tuning and training from scratch almost never manage to leave the first level, confirming their poor performance with respect to score.

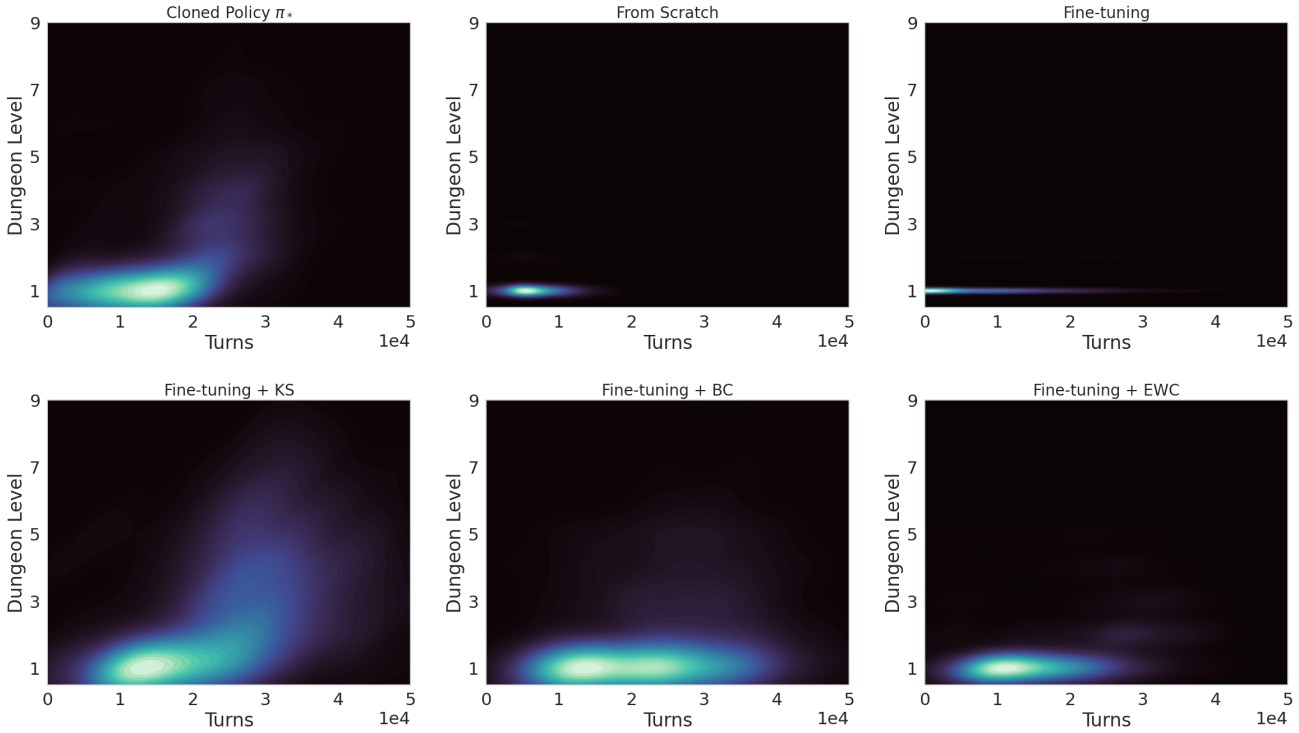
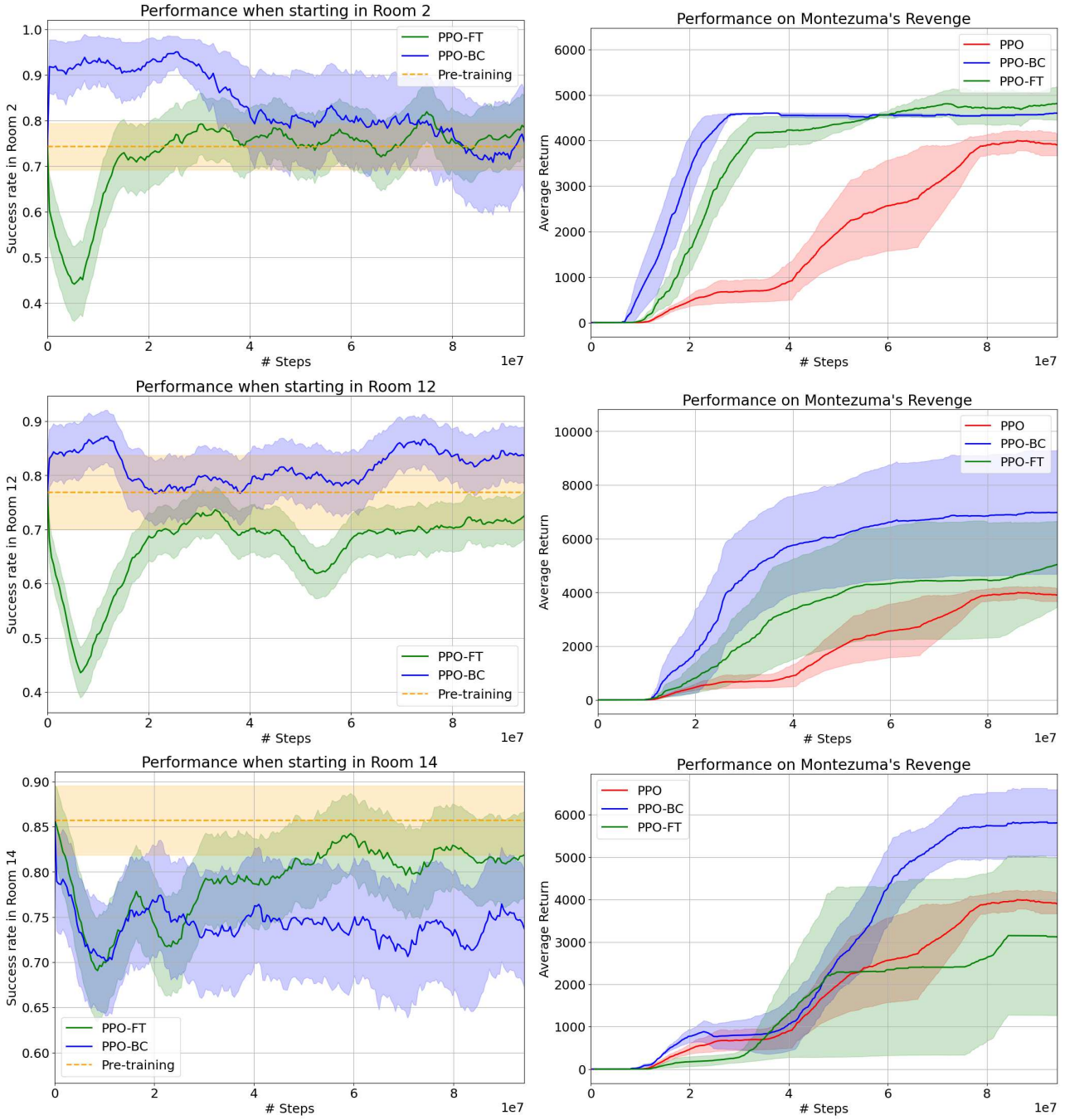


Figure 16: Density plots showing maximum dungeon level achieved compared to the total number of turns (units of in-game time). Brighter colors indicate higher visitation density.

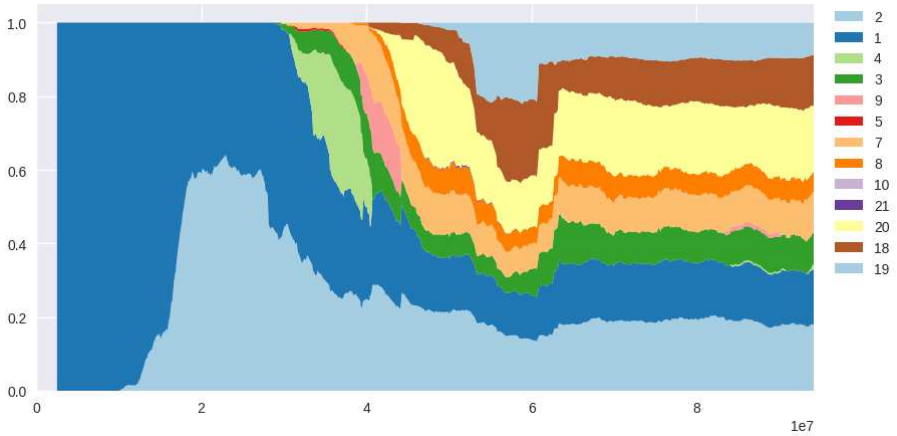
E. Additional Montezuma's Revenge results



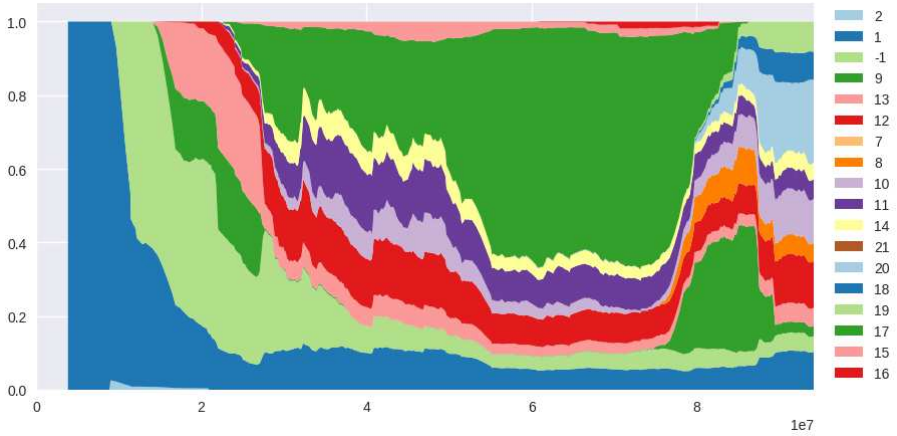
(a) Success rate in rooms during fine-tuning when initialized in that room.

(b) Average return throughout the training. PPO represents training from scratch, PPO-FT is vanilla fine-tuning and PPO-BC is fine-tuning with the behavioral cloning loss.

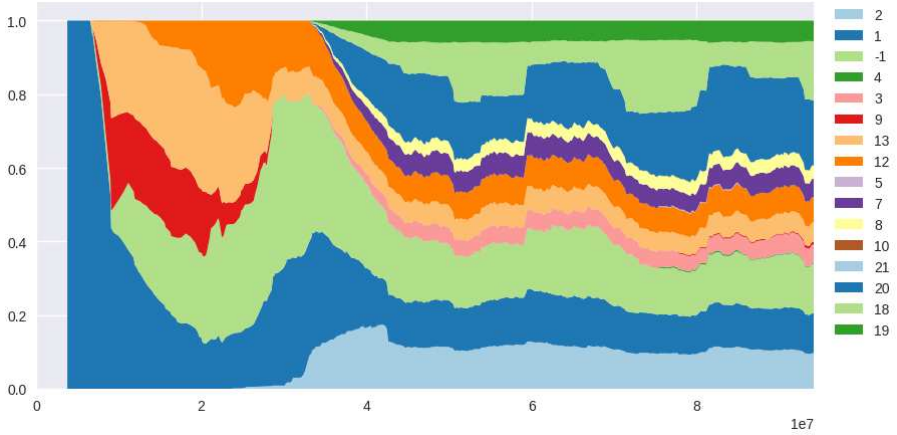
Figure 17: State coverage gap in Montezuma's Revenge.



(a) Room visitation for training from scratch

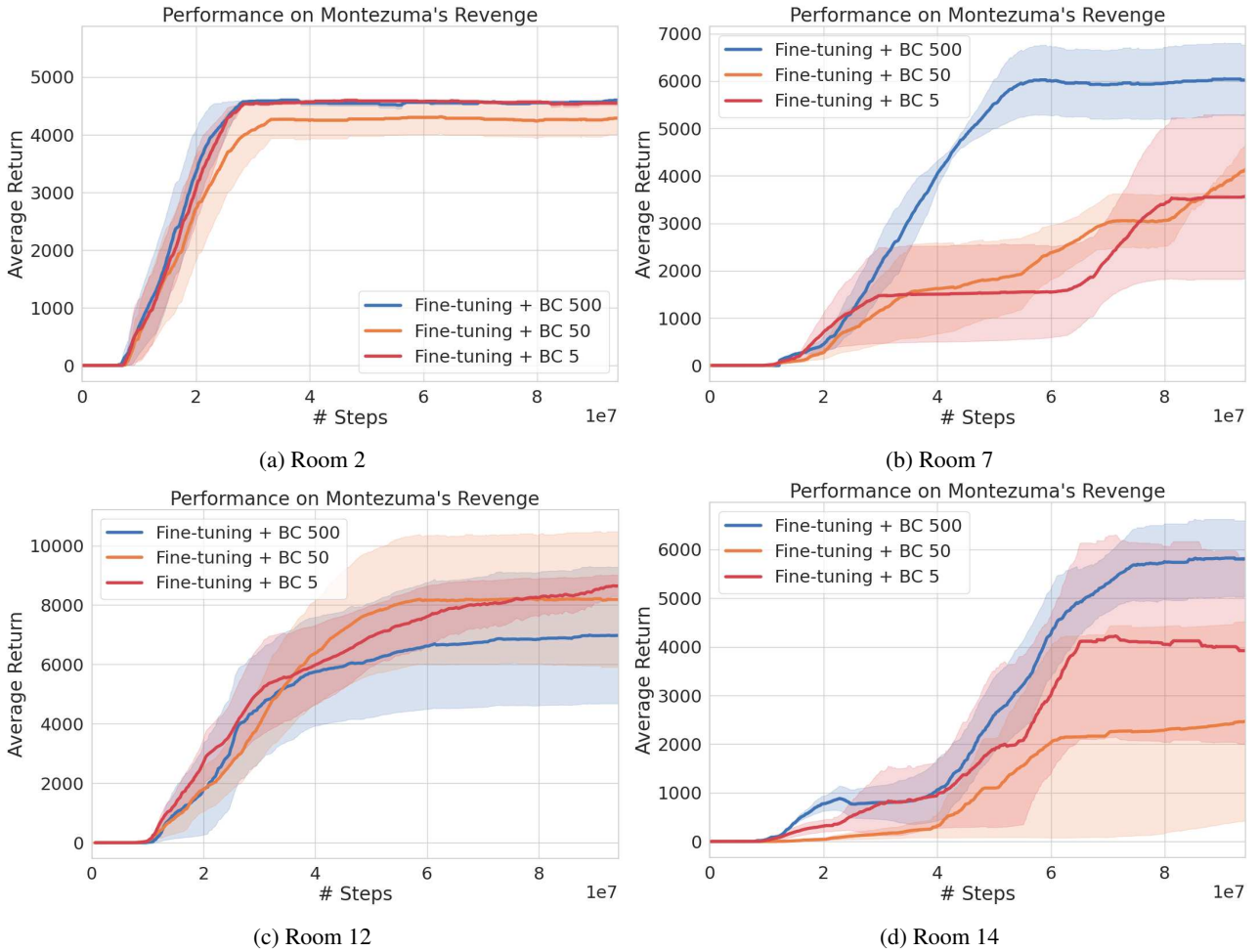


(b) Room visitation for fine-tuning



(c) Room visitation for fine-tuning + BC

Figure 18: Time spent in different rooms across training for training from scratch (top), fine-tuning (middle), and fine-tuning + BC (bottom). The agent trained from scratch struggles to explore rooms at the beginning of the training and eventually visits fewer of them than fine-tuned agents.



(a) Room 2

(b) Room 7

(c) Room 12

(d) Room 14

Figure 19: Results for different buffer sizes in Montezuma's Revenge.

Analysis of forgetting with different pre-training schemes We perform additional experiments on three different rooms in a setting analogous to the one from the main paper (see Section 3 for details). In particular, we are interested in the behavior of the pre-trained model from a specific room while fine-tuned. Figure 17 shows a significant drop in performance for vanilla fine-tuned models without additional knowledge retention methods (PPO-FT) just after fine-tuning starts. In contrast, PPO-BC (i.e. fine-tuning + BC) mitigates this effect except for Room 14. For all pre-training types, PPO-BC outperforms PPO-FT with respect to the score.

Room visitation analysis Since exploration is a crucial problem in Montezuma's Revenge, we check how well different types of agents manage to explore the maze throughout the game. In Figure 18, we show how the time spent in different rooms changes across the training for an agent trained from scratch, the fine-tuned agent, and the fine-tuned agent with BC loss. For simplicity, we focus on our primary setting, i.e. the one where pre-training starts from Room 7.

The agent trained from scratch spends a significant amount of time learning to escape the first two rooms and navigate the maze. Interestingly, both vanilla fine-tuning and fine-tuning + BC retain the capability for exploration obtained in pre-training, as they exit the first room quickly, even though it was not seen at all during pre-training. However, in the later phase of fine-tuning, the agent with knowledge retention manages to see a wider variety of rooms than the one without it, which spends a significant amount of time in e.g. Room 9. This suggests that forgetting of pre-trained capabilities also applies to exploration capabilities and knowledge retention methods can mitigate their loss.

Impact of the buffer size Finally, we check how the size of the replay buffer for Fine-tuning + BC impacts the results. Results presented in Figure 19 show that indeed having a larger buffer is always the best option, although the performance

gap vanishes in some settings.

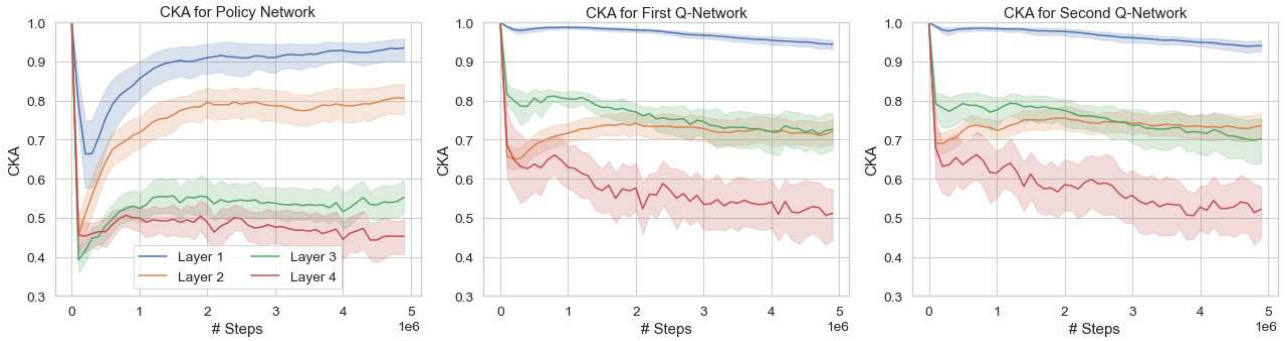


Figure 20: The CKA values throughout vanilla fine-tuning (without knowledge retention methods), computed between the activations of the pre-trained model and the activations of the current model. The higher the values, the more similar the representations.

F. Analysis of forgetting in robotic manipulation tasks

In this section, we present additional results for our robotic manipulation experiments based on Meta-World.

Unless specified otherwise, we use the experimental setting from Section 3. We adopt the forward transfer metric used previously in (Wołczyk et al., 2021; Bornschein et al., 2022) to measure how much pre-trained knowledge helps during fine-tuning:

$$\text{Forward Transfer} := \frac{\text{AUC} - \text{AUC}^b}{1 - \text{AUC}^b}, \quad \text{AUC} := \frac{1}{T}\int_0^T p(t)\mathrm{d}t, \quad \text{AUC}^b := \frac{1}{T}\int_0^T p^b(t)\mathrm{d}t,$$

where $p(t)$ is the success rate of the pre-trained model at time t, p^b denotes the success rate of a network trained from scratch, and T is the training length. Intuitively, it measures how much faster the fine-tuned model learns than the one trained from scratch.

Analysis of internal representations We examine how activations of the actor and critic networks in SAC change throughout fine-tuning when we do not use any knowledge retention methods, with the goal of pinpointing the structure of forgetting. To measure the representation shift in the network, we use the Central Kernel Alignment (CKA) (Kornblith et al., 2019) metric, which was previously used in studying forgetting in the supervised learning paradigm (Ramasesh et al., 2020; Mirzadeh et al., 2022). Before starting the fine-tuning process, we collect optimal trajectories from the pre-trained model along with the activations of the networks after each layer. Then, at multiple points throughout the training process, we feed the same trajectories through the fine-tuned network and compare its activations to the prior activations using CKA. Figure 20 shows that, in general, later layers change more than the early layers, which is consistent with previous studies (Ramasesh et al., 2020). This is particularly visible in the policy network, while the tendency is not as strong for the critic networks, suggesting that the TD-learning guiding the critic leads to different representation learning dynamics.

In the policy network, representations in the early layers change rapidly at the beginning of the fine-tuning process. Then, interestingly, as we solve the new tasks and revisit the tasks from pre-training, CKA increases and the activations become more similar to the pre-trained ones. As such, the re-learning visible in per-task success rates in Figure 7 is also reflected in the CKA here. However, this phenomenon does not hold for the later layers in the policy network or the Q-networks. This suggests that the solution we find is significantly different.

Impact of the network size Previous studies in supervised continual learning showed that forgetting might start disappearing as we increase the size of the neural network (Ramasesh et al., 2022; Mirzadeh et al., 2022), and here we investigate the same point in RL using our `RoboticSequence` setting. We run a grid of experiments with hidden dimensions in $\{256, 512, 1024\}$ and number of layers in $\{2, 3, 4\}$. For each of these combinations, we repeat the experiment from the main text, namely, we measure how fine-tuning from a pre-trained solution compares to starting from random initialization and how the results change when we apply continual learning methods. The results are presented in Figure 27.

The results do not show any clear correlations between the network size and forgetting, hinting at more complex interactions than these previously showed in continual supervised learning literature (Ramasesh et al., 2022). The fine-tuning approach

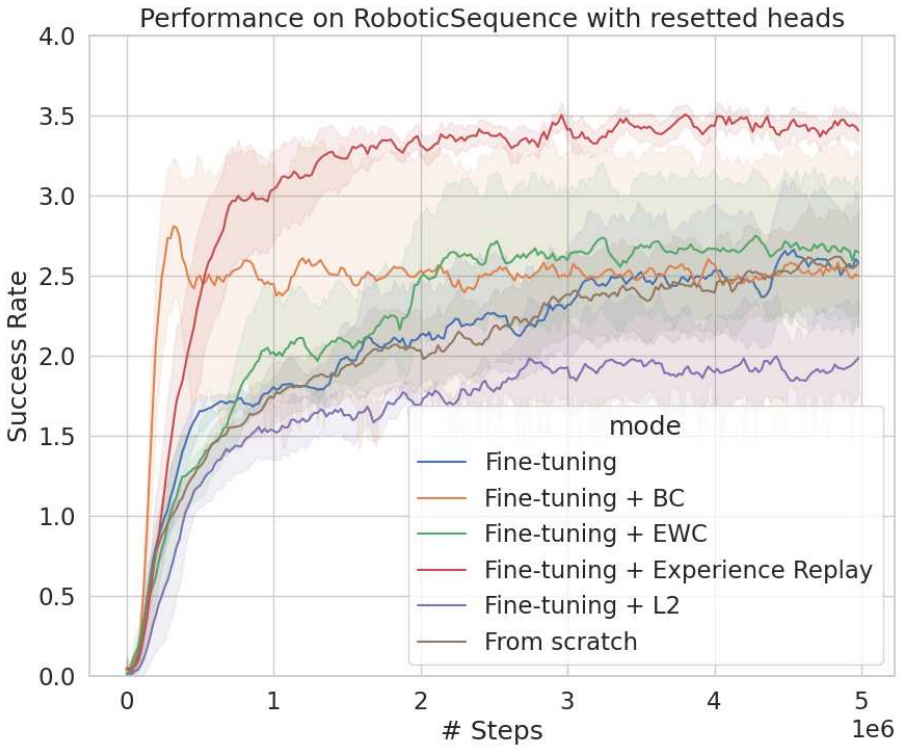


Figure 21: Performance of different methods on the RoboticSequence where we reset the last layer of the policy and critic networks. The results are worse than in the standard case, but there is still some positive transfer, suggesting that benefits come from reusing both the representations as well as the policy.

fails to achieve a significant positive transfer for two or four layers, but it does show signs of knowledge retention with three layers. Inspection of the detailed results for the three-layer case shows that the fine-tuning performance on the known tasks still falls to zero at the beginning, but it can regain performance relatively quickly. As for the CL methods, we observe that behavioral cloning performs well independently of the size of the network. On the other hand, EWC tends to fail with two layers. Since EWC directly penalizes changes in the parameters, we hypothesize that with a small, two-layer network, the resulting loss of plasticity makes it especially difficult to learn.

Impact of the number of unknown tasks In our APPLERETRIEVAL experiments, we showed that forgetting of pre-trained capabilities is more visible as we increase the amount of time spent before visiting the known part of the state space. We investigate the same question in the context of robotic manipulation tasks by changing the number of new tasks the agent has to solve prior to reaching the ones it was pre-trained on. That is, we study `RoboticSequence`s where the last two tasks are `peg-unplug-side` and `push-wall`, as previously, but the first tasks are taken as different length suffixes of `window-close`, `faucet-close`, `hammer`, `push` We call the tasks preceding the pre-trained tasks the *prefix tasks*.

We investigate how the number of the prefix tasks impacts the performance on the known tasks during the fine-tuning process. Table 6 shows the forward transfer metric computed on the pre-trained tasks for fine-tuning, EWC and BC. As the number of prefix tasks grows, the forward transfer values for fine-tuning become smaller, which means that the gains offered by the prior knowledge vanish. Interestingly, even with a single prefix task the forward transfer is relatively low. On the other hand, continual learning methods do not suffer as much from this issue. BC achieves high forward transfer regardless of the setting and EWC experiences only small deterioration as we increase the number of prefix tasks.

Impact of representation vs policy on transfer Although we see significant positive transfer once the forgetting problem is addressed, it remains an open question where this impact comes from. Although there are several studies on the impact of representation learning on transfer in supervised learning (Neyshabur et al., 2020; Kornblith et al., 2021), the same question in RL remains relatively understudied. Here, we try to understand the impact of representation and policy on transfer by

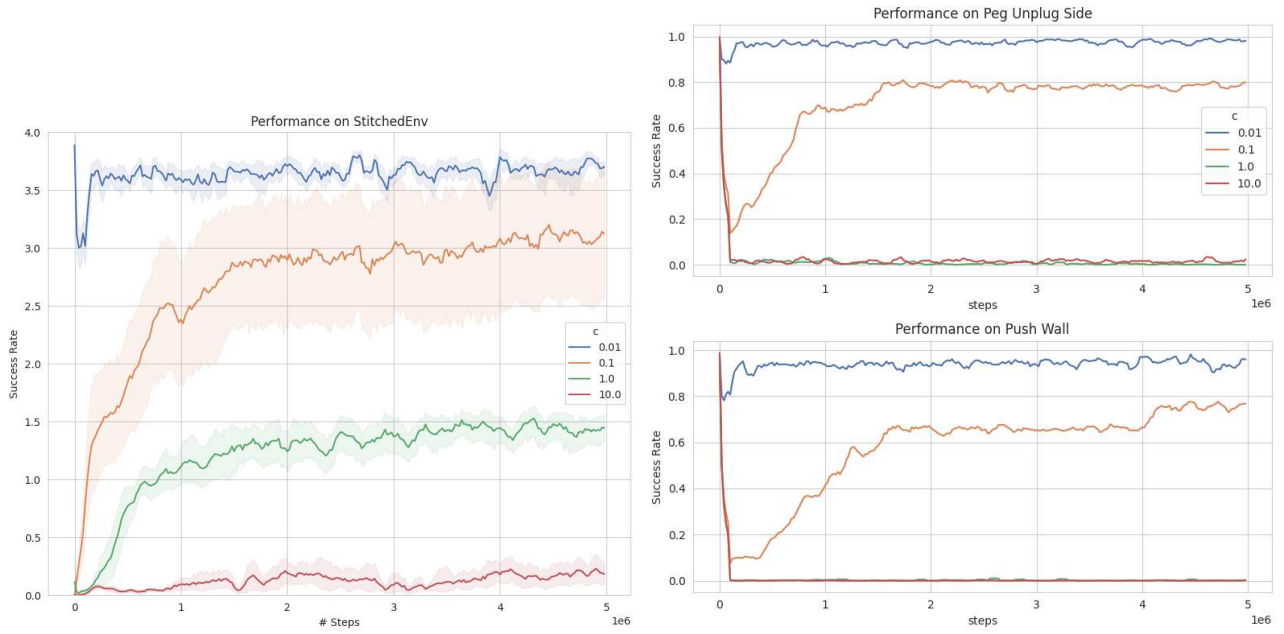


Figure 22: The performance on a robotic sequence where the sequence consists of the same tasks, but with observations translated by a constant c. We can observe forgetting even for small perturbations ($c = 0.1$).

resetting the last layer of the network before starting the training. As such, the policy at the beginning is random even on the tasks known from pre-training, but has features relevant to solving these tasks. The improvements should then only come from the transfer of representation.

The results for these experiments are presented in Figure 21. First of all, we observe that, expectedly, this setting is significantly harder, as all methods perform worse than without resetting the head. However, we still observe significant transfer for BC and EWC as they train faster than a randomly initialized model. At the same time, fine-tuning in the end manages to match the performance of BC and EWC, however at a much slower pace. We hypothesize that the gap between knowledge retention methods and fine-tuning is smaller, since now the methods have to re-learn a new policy rather than maintain the old one. This preliminary experiment suggests that the benefits of fine-tuning come from both the policy and the representation since we can still observe a significant, although reduced, transfer after resetting the heads. Maximizing transfer from the representation remains an interesting open question.

Impact of task difference The severity of forgetting is deeply connected to how different FAR and CLOSE tasks are to each other. We refer the reader to Section 6 for a short description of prior continual learning papers on this problem, and here we perform a simple experiment on this issue. We construct a RoboticSequence consisting of tasks `peg unplug (translated)`, `push wall (translated)`, `peg unplug`, `push wall` and use a model pre-trained on the last two tasks. (Translated) means that the observation vectors are shifted by a constant c. This is a very basic form of state perturbation. In this case, the non-translated (translated resp.) stages correspond to FAR (CLOSE resp.) states. We run vanilla fine-tuning experiments with values of $c \in (0.01, 0.1, 1, 10)$. We observe no forgetting for $c = 0.01$, partial forgetting for $c = 0.1$, and total forgetting for $c = 1$, and $c = 10$. We treat this result as initial evidence supporting the claim that even small discrepancies between far and close states might lead to forgetting.

Other sequences In order to provide another testbed for our investigations, we repeat the main experiments on another sequence of tasks, namely `shelf-place`, `push-back`, `window-close`, `door-close`, where again we fine-tune a model that was pre-trained on the last two tasks. The results are presented in Figure 25. We find that the main conclusions from the other sequence hold here, although, interestingly, the performance of EWC is significantly better. Additionally, we run experiments on a simple, two task `RoboticSequence` with `drawer-open` and `pick-place`, showcased in Figure 1. We used behavioral cloning as an example of a method that mitigates forgetting.

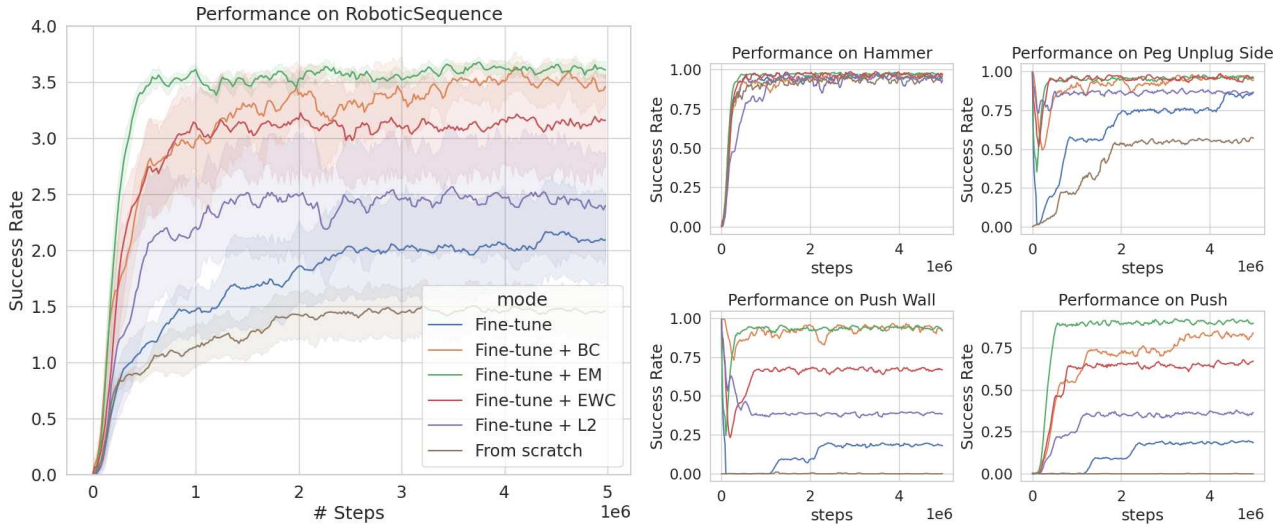


Figure 23: The performance on a robotic sequence where the known tasks are in the middle.

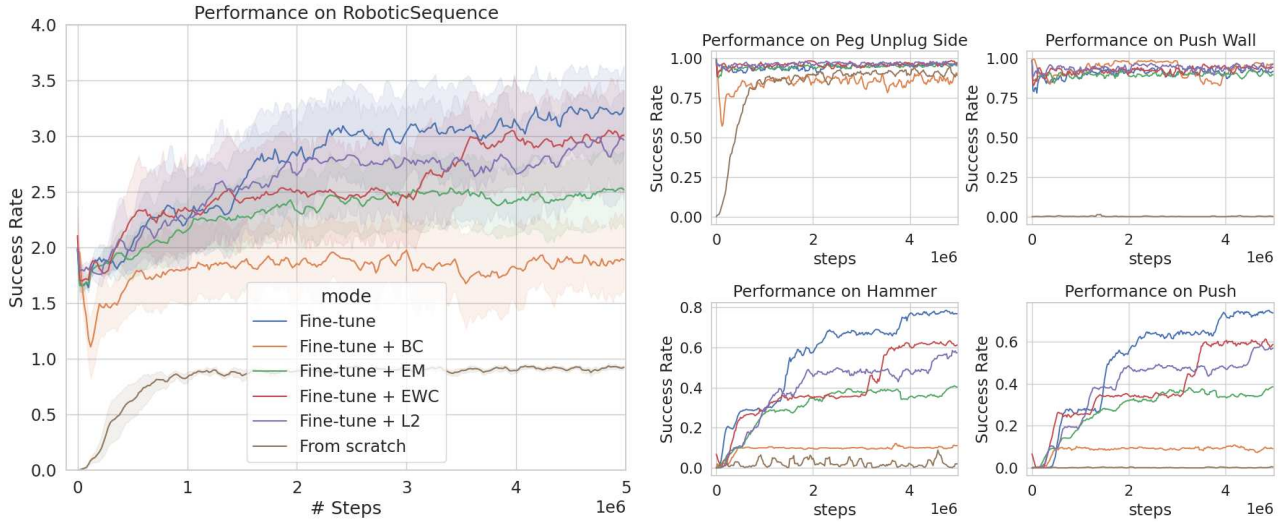


Figure 24: The performance on a robotic sequence where the known tasks are positioned at the beginning.

Additionally, we check what happens when the known tasks are "in the middle" of two known tasks. That is, we use the environment consisting of the following sequence of goals: `hammer, peg-unplug-side, push-wall, push` with a model pre-trained on `peg-unplug-side, push-wall`. With this setup, we are especially interested in the impact of different methods on the performance on the last task, i.e. can we still learn new things after visiting a known part of the state space?

The results presented in Figure 23 show that the relative performance of all methods is the same as in our original ordering, however, we observe that EWC almost matches the score of BC. The learning benefits on the last task, `push`, is somewhat difficult to estimate. That is since BC manages to maintain good performance on tasks `peg-unplug-side` and `push-wall`, it sees data from `push` much sooner than approaches that have to re-learn tasks 2 and 3. However, we observe that even after encountering the later tasks, knowledge retention methods perform much better on `push` than vanilla fine-tuning, which in turn is better than a model trained from scratch.

Finally, we verify that the gap between vanilla fine-tuning and knowledge retention methods does not appear when the relevant skills are only needed at the start of the downstream task. To do this, we use the following sequence of

goals: `peg-unplug-side`, `push-wall`, `hammer`, `push` with a model pre-trained on `peg-unplug-side`, `push-wall`. Results in Figure 24 show that indeed in this scenario there is no forgetting and fine-tuning manages just as well or sometimes even slightly better than knowledge retention methods.

Impact of the memory size on the results The memory overhead is an important consideration in fine-tuning with a behavioral cloning loss. We run experiments to check how many samples we actually need to protect knowledge of the previous tasks. Results presented in Figure 26 show that even with 100 samples we are able to keep good performance, at the cost of a higher performance drop on the pre-trained tasks at the beginning of the fine-tuning process.

Table 6: Forward transfer on the pre-trained tasks depending on the number of prefix tasks in `RoboticSequence`.

Prefix Len	`push-wall`			`peg-unplug-side`		
	FT	EWC	BC	FT	EWC	BC
1	0.18 [-0.19, 0.43]	0.88 [0.84, 0.91]	0.93 [0.89, 0.96]	0.28 [0.01, 0.46]	0.77 [0.58, 0.88]	0.92 [0.88, 0.94]
2	0.17 [-0.21, 0.44]	0.65 [0.44, 0.82]	0.97 [0.97, 0.98]	0.15 [-0.08, 0.35]	0.55 [0.37, 0.70]	0.95 [0.94, 0.96]
3	0.10 [-0.03, 0.23]	0.64 [0.50, 0.75]	0.98 [0.98, 0.98]	0.03 [0.00, 0.06]	0.41 [0.28, 0.54]	0.95 [0.95, 0.95]
4	-0.00 [-0.16, 0.10]	0.62 [0.48, 0.75]	0.97 [0.97, 0.98]	0.03 [-0.00, 0.08]	0.46 [0.33, 0.59]	0.94 [0.94, 0.95]

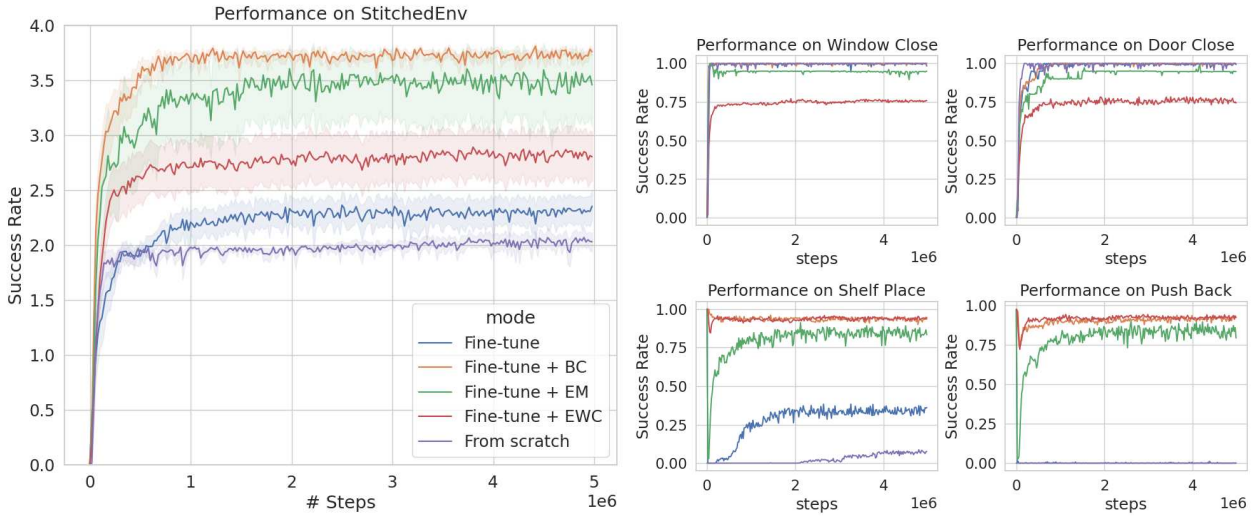


Figure 25: The performance of the fine-tuned model on `RoboticSequence` compared to a model trained from scratch and knowledge retention methods on the sequence `shelf-place`, `push-back`, `window-close`, `door-close`.

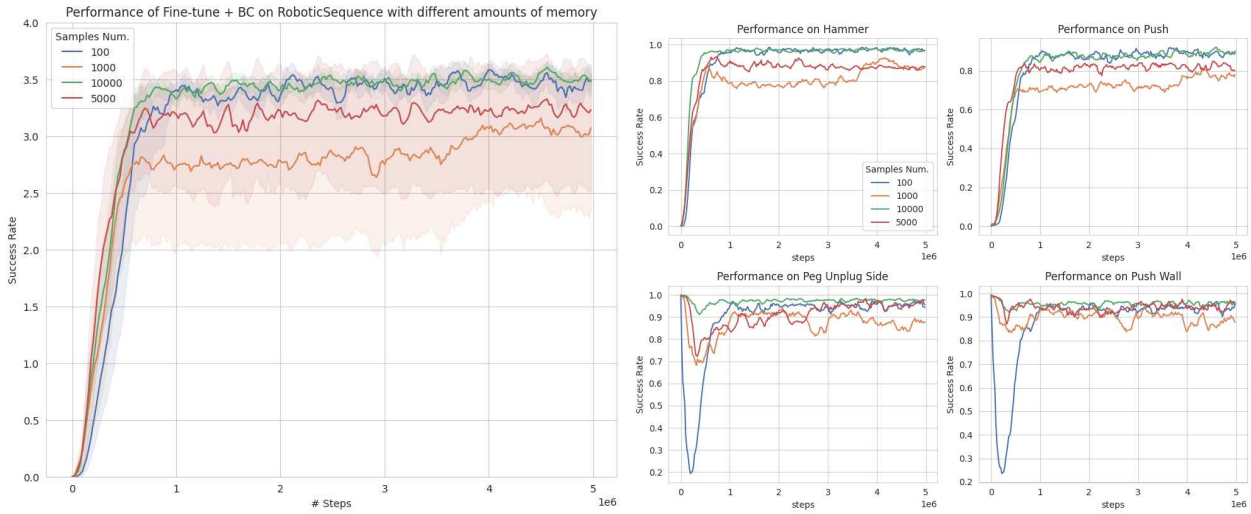
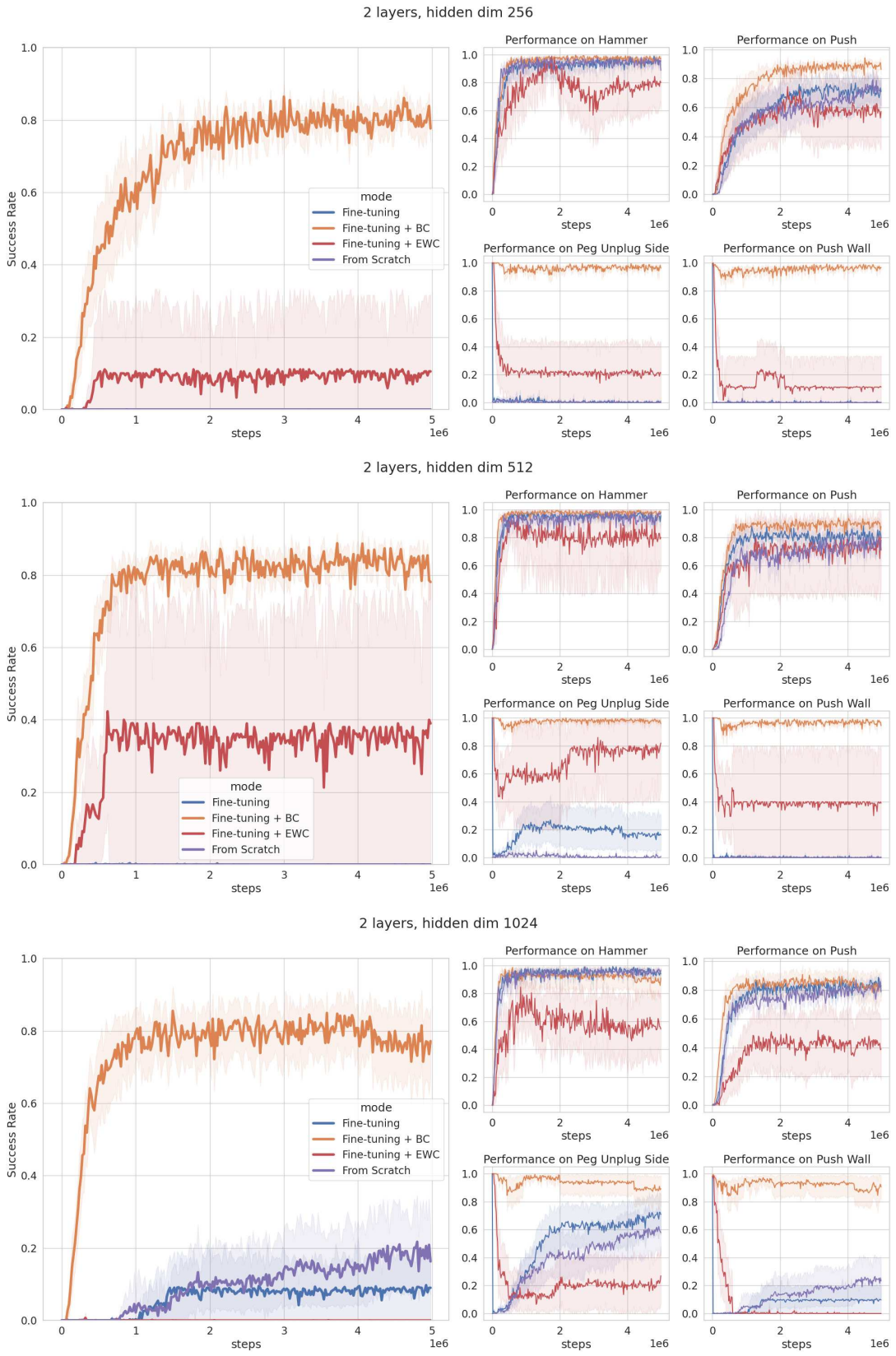
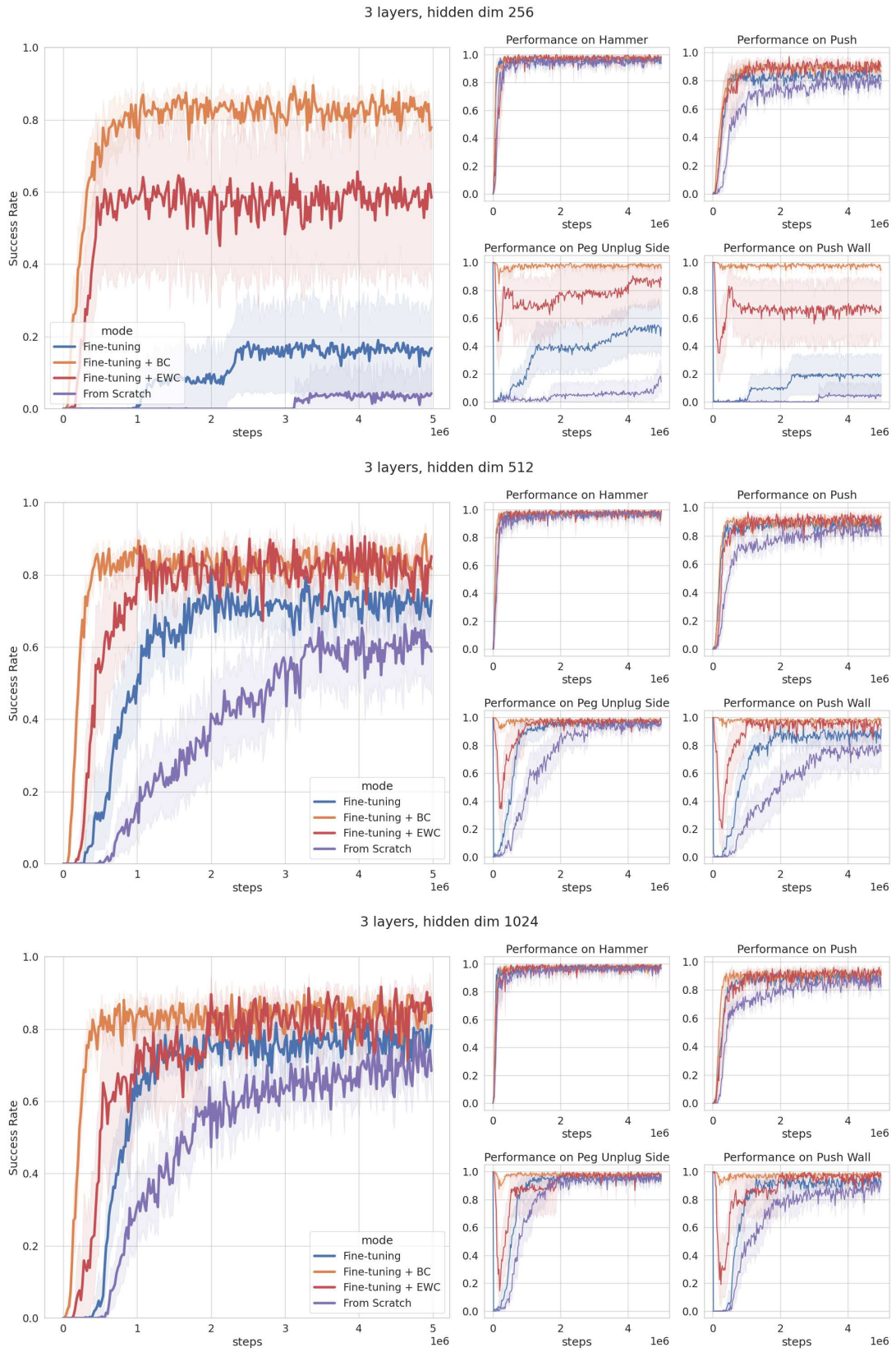
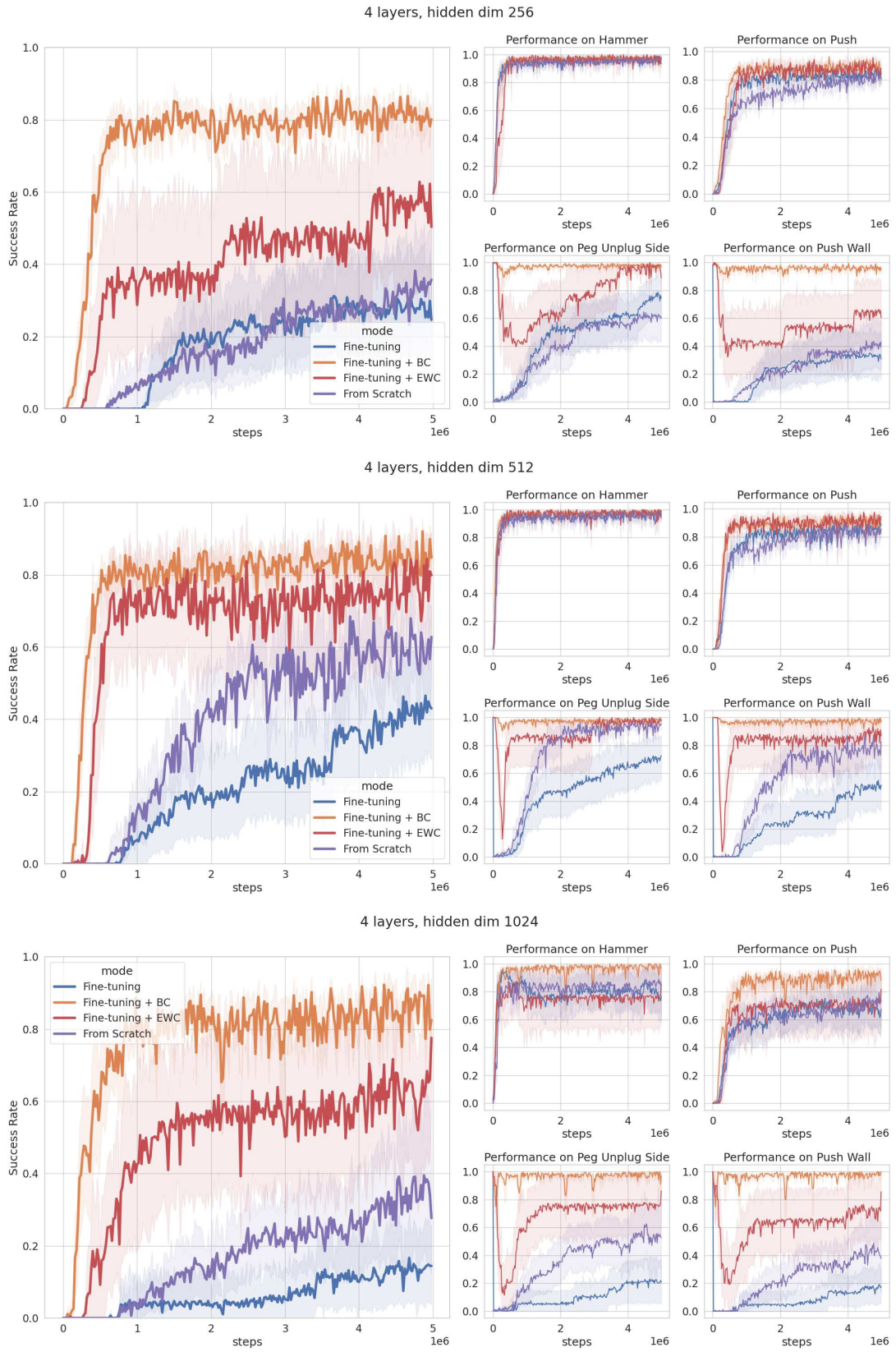


Figure 26: The performance of Fine-tune + BC with different memory sizes. Even with 100 samples we are able to retain the knowledge required to make progress in the training.

Figure 27: Training performance for different architecture choices.




3 layers, hidden dim 256
Success Rate
steps
1e6
mode
Fine-tuning
Fine-tuning + BC
Fine-tuning + EWC
From Scratch
Performance on Hammer
Performance on Push
Performance on Peg Unplug Side
Performance on Push Wall
3 layers, hidden dim 512
3 layers, hidden dim 1024


4 layers, hidden dim 256
Success Rate
steps
1e6
mode
Fine-tuning
Fine-tuning + BC
Fine-tuning + EWC
From Scratch
Performance on Hammer
Performance on Push
Performance on Peg Unplug Side
Performance on Push Wall
4 layers, hidden dim 512
4 layers, hidden dim 1024