Beyond Fine-Tuning: Transferring Behavior in Reinforcement Learning

Anonymous Authors[1]

Abstract

Designing agents that acquire knowledge autonomously and use it to solve new tasks efficiently is an important challenge in reinforcement learning. Knowledge acquired during an unsupervised pre-training phase is often transferred by fine-tuning neural network weights once rewards are exposed, as is common practice in supervised domains. Given the nature of the reinforcement learning problem, we argue that standard fine-tuning strategies alone are not enough for efficient transfer in challenging domains. We introduce *Behavior Transfer* (BT), a technique that leverages pre-trained policies for exploration and that is complementary to transferring neural network weights. Our experiments show that, when combined with large-scale pre-training in the absence of rewards, existing intrinsic motivation objectives can lead to the emergence of complex behaviors. These pre-trained policies can then be leveraged by BT to discover better solutions than without pre-training, and combining BT with standard fine-tuning strategies results in additional benefits. The largest gains are generally observed in domains requiring structured exploration, including settings where the behavior of the pre-trained policies is misaligned with the downstream task.

1. Introduction

Transfer in deep learning is often performed through parameter initialization followed by fine-tuning, a technique that allows to leverage the power of deep networks in domains where labelled data is scarce (Yosinski et al., 2014; Donahue et al., 2014; Zeiler & Fergus, 2014; Girshick et al., 2014; Devlin et al., 2019). This builds on the intuition that the pre-trained model will map inputs to a feature space where the downstream task is easy to perform. When combined with methods that can leverage massive amounts of unlabelled data for pre-training, this transfer strategy has led to unprecedented results in domains like computer vision (Hénaff et al., 2019; He et al., 2019) and natural language processing (Devlin et al., 2019; Radford et al., 2019). The success of these approaches has led to an ever-growing interest in developing techniques for pre-training large scale models on unlabelled data (Brown et al., 2020; Chen et al., 2020; Grill et al., 2020).

In the reinforcement learning (RL) context, unsupervised methods that learn in the absence of reward have also garnered much research attention (Gregor et al., 2016; Florensa et al., 2017; Pathak et al., 2017; Eysenbach et al., 2019; Hazan et al., 2019). The benefits of unsupervised pre-training are typically evaluated by their ability to enable efficient transfer to previously unseen reward functions (Hansen et al., 2020). In spite of their different approaches to unsupervised RL, most of the top-performing methods in this setting transfer knowledge through neural network weights. Such approaches deal with the data inefficiency associated to training neural networks with gradient descent, similarly to what is done in supervised learning, e.g. by pre-training encoders that extract representations from observations (Yarats et al., 2021). However, RL introduces a challenge that is not present in supervised learning: the agent is responsible for collecting the right data to learn from. This introduces a second source of inefficiency from which transfer approaches can also suffer if they rely on unstructured exploration strategies after pre-training, as these can lead to exponentially larger data requirements in complex downstream environments (Osband et al., 2016b;a). To address this problem, one could consider fine-tuning policies that produce meaningful behavior (Mutti et al., 2021; Schwarzer et al., 2021), but this approach quickly disregards the pre-trained behavior when learning in the downstream task due to catastrophic forgetting.

In this work, we explicitly separate the transfer of behaviour and weights. We propose to make use of the pre-trained behaviour itself (i.e., the pre-trained policy mapping from observations to actions) in contrast to pre-trained neural

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

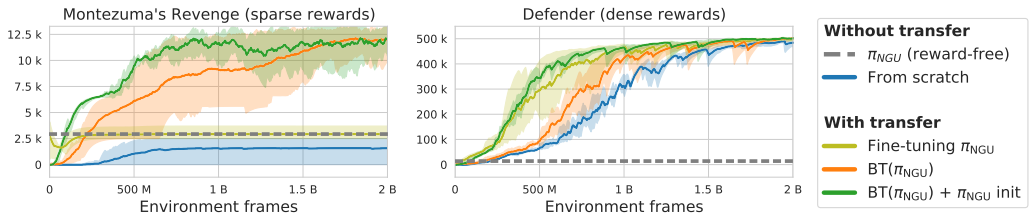

Figure 1. Comparison of transfer strategies on Montezuma's Revenge and Defender after pre-training a policy with NGU (Puigdomènech Badia et al., 2020b) in the absence of reward. The benefits of our proposed approach to leverage pre-trained behavior for exploration, Behavior Transfer (BT), are complementary to the gains provided by pre-trained weight initialization followed by fine-tuning.

network weights for further fine-tuning. While pre-trained behavior has been used before for *exploitation* (Barto & Mahadevan, 2003; Sutton et al., 1999; Barreto et al., 2017; 2018), our approach employs pre-trained policies to aid with *exploration* as well to collect experience that can be leveraged via off-policy learning. This strategy accelerates learning, as the agent is exposed to potentially useful experience earlier in training, without compromising the quality of the discovered solution when the pre-trained behavior is not aligned with the downstream task. We expose the pre-trained behaviour to the downstream agent in two ways: firstly, as an extra exploratory strategy that, when randomly activated, persists for a number of steps, and secondly as an additional pseudo-action for the learned value function where the agent may elect to defer action selection to the pre-trained policy instead of choosing itself. We call this approach Behavior Transfer (BT).

Defining unsupervised RL objectives remains an open problem, and solutions are generally influenced by how the acquired knowledge will be used for solving downstream tasks. Instead of proposing yet another objective for unsupervised pre-training, we turn to existing techniques for training policies in the absence of reward and make our choice based on two general requirements. First, the objective should scale gracefully with increased compute and data. This has been key for the success of self-supervised approaches in other domains (Brown et al., 2020; Kaplan et al., 2020), and we argue that it is an important property for unsupervised RL as well. Second, the pre-training stage should return a policy that produces complex behavior that may be leveraged in a subsequent transfer stage. The *Never Give Up* (NGU) (Puigdomènech Badia et al., 2020b) intrinsic reward meets both requirements, and our experiments show that large-scale pre-training with this objective leads to state of the art scores in the reward-free Atari benchmark.

Figure 1 exemplifies our main findings. We pre-train behaviour using the intrinsic NGU reward during a long unsupervised phase without rewards. This gives rise to exploratory behaviors that seek to visit many different states throughout an episode, and we then compare different strategies for leveraging the acquired knowledge once rewards are reinstated. While fine-tuning the pre-trained weights enables faster learning, the exploratory behavior of the pre-trained policy is quickly disregarded as it is exposed to rewards. On the other hand, Behavior Transfer (BT) does not modify the pre-trained policy while learning in the new task and is able to achieve higher end scores thanks to better exploration. These two strategies are not mutually exclusive, and BT also benefits from the faster convergence provided by initializing neural networks with pre-trained weights when these encode useful information for solving the downstream task.

Our contributions can be summarized as follows. (1) We propose *Behavior Transfer* (BT), a technique that leverages pre-trained policies for exploration by treating them as black boxes that are not modified during learning on the downstream task. BT uses the pre-trained policy to collect experience in two ways, namely randomly-triggered temporally-extended exploration and one-step calls based on value estimates. (2) Our experiments show that large-scale unsupervised pre-training with existing intrinsic rewards can produce meaningful behavior, achieving state of the art results in the reward-free Atari benchmark. These results suggest that scale is key for unsupervised RL, akin to what has been observed in supervised settings. (3) We provide extensive empirical evidence demonstrating the benefits of leveraging pre-trained behavior via BT. Our approach obtains the largest gains in hard exploration games, where it almost doubles the median human normalized score achieved by our strongest baseline. Furthermore, we show that BT is able to leverage a single task-agnostic policy to solve multiple tasks in the same environment and to achieve high performance even when the pre-trained policies are misaligned with the task being solved. (4) BT brings benefits to the table that are complementary to those provided by reusing pre-trained neural network weights, and we empirically show that combining these two strategies can result in larger gains.

2. Preliminaries

The interaction between the agent and the environment is modelled as a Markov Decission Process (MDP) (Puterman, 1994). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, d_0, R, \gamma)$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P(s'|s,a)$ is the probability of transitioning from state s to s' after

taking action a, $d_0(s)$ is the probability distribution over initial states, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. The goal is to find a policy $\pi(a|s)$ that maximizes the expected return, $G_t = \sum_{t=0}^{\infty} \gamma^t R_t$, where $R_t = r(S_t, A_t, S_{t+1})$. A principled way to address this problem is to use methods that compute action-value functions, $Q^\pi(s, a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a]$, where $\mathbb{E}_\pi[\cdot]$ denotes expectation over transitions induced by π (Puterman, 1994).

We consider a setting where the agent is allowed to first learn within an MDP without rewards, $\mathcal{M}^{\not{R}} = (\mathcal{S}, \mathcal{A}, P, d_0)$, for a long period of time. The knowledge acquired during the reward-free stage is later leveraged when maximizing reward in new MDPs that share the same underlying dynamics but have different reward functions, $\mathcal{M}_i = (\mathcal{S}, \mathcal{A}, P, d_0, R_i, \gamma_i)$. Interactions between the agent and the environment are often assumed to incur a cost, but we will consider this cost to be relevant only for transitions with reward (Hansen et al., 2020). Even if the cost of unsupervised pre-training becomes non-negligible, it can be amortized when the acquired task-agnostic knowledge is leveraged to solve multiple tasks efficiently (Devlin et al., 2019; Brown et al., 2020). Indeed, we would expect this transfer setting to become more relevant as the community moves towards more complex environments, where one may want to train agents to maximize multiple reward functions under constant dynamics. In the limit, one could consider the real world: it has constant or slowly changing dynamics, and humans are able to leverage previously acquired skills to quickly master new tasks.

3. Behavior Transfer

Transfer in supervised domains often exploits the fact that related tasks might be solved using similar representations. This practice deals with the data inefficiency of training large neural networks with stochastic gradient descent. However, there is an additional source of data inefficiency when training RL agents: unstructured exploration. Fine-tuning a pre-trained exploratory policy arises as a potential strategy for overcoming this problem, as the agent will observe rich experience much earlier in training than when initializing the policy randomly, but this approach suffers from important limitations. Learning in the downstream task can lead to catastrophically forgetting the pre-trained policy, thus prematurely disregarding its exploratory behavior. Moreover, the same neural network architecture needs to be used for both the pre-trained and the downstream policies, which in practice also imposes a limitation on the type of RL methods that can be employed in the adaptation stage (for instance, if the pre-trained policy was trained using a policy-based method, it might not be possible to fine-tune it using a value-based approach).

Let us assume that we have access to a pre-trained policy that exhibits exploratory behavior, and defer the discussion on how to train this policy to Section 4. Following such a policy might bring the agent to states that are unlikely to be visited with unstructured exploration techniques such as ϵ-greedy (Sutton & Barto, 2018). This property has the potential of accelerating learning even when the behavior of the pre-trained policy is not aligned with the downstream task, as it will effectively shorten the path between otherwise distant states (Liu & Brunskill, 2018). Leveraging pre-trained policies for exploration differs from other approaches in the literature that use such policies directly for exploitation, e.g. via zero-shot transfer (Eysenbach et al., 2019), methods that define a higher-level policy that alternates between the given policies (Barto & Mahadevan, 2003; Sutton et al., 1999), or within the framework of generalized policy updates (Barreto et al., 2020). Exploring with pre-trained policies can accelerate convergence by providing useful experience to the agent, which is possible even when the pre-training and downstream tasks are misaligned. However, strategies that directly use the pre-trained policies for exploitation may result in sub-optimal solutions in such scenario (Barreto et al., 2017).

We propose to leverage the behavior of pre-trained policies during transfer to aid with exploration. An explicit distinction between behavior and representation is made by considering pre-trained policies as black boxes that take observations and return actions. This strategy is agnostic to how the pre-trained behavior is encoded and is not restricted to learned policies. We rely on off-policy learning methods during transfer to leverage the behavior of a pre-trained policy $\pi_p(a|s)$. We keep π_p fixed during transfer, which prevents catastrophic forgetting of the original behavior when it is parameterized by a neural network (i.e., we instantiate and train a new policy with its own set of parameters). We propose *Behavior Transfer* (BT), which leverages two complementary strategies to achieve this. Since BT is agnostic to the method used to pre-train policies, $BT(\pi_p)$ refers to behavior being transferred from policy π_p. We formalize BT in the context of value-based Q-learning agents, although similar derivations are in principle possible for alternative off-policy learning methods. Pseudo-code for BT is provided in Algorithm 1.

Temporally-extended exploration. We draw inspiration from Lévy flights (Viswanathan et al., 1996), a class of ecological models for animal foraging, where a fixed direction is followed for a duration sampled from a heavy-tailed distribution. This principle was implemented in the context of exploration in RL by ϵz-greedy (Dabney et al., 2021), which encodes the notion of direction in the environment via exploration options that repeat the same action throughout the entire flight. Since π_p is more likely to encode a meaningful notion of direction in complex environments than action

repeats, we propose a variant of ϵz-greedy where π_p is used as the exploration option. An exploratory flight might be started at any step with some probability. The duration for the flight is sampled from a heavy-tailed distribution (Zeta with $\mu = 2$ in all our experiments), and control is handed over to π_p during the complete flight. When not in a flight, actions are sampled from the behavior policy obtained while maximizing the task reward (e.g. an ϵ-greedy derived from the estimated Q values).

Extra action. The previous approach switches to π_p during experience collection blindly, and we now consider an alternative strategy for triggering these switches based on value. This can be easily implemented through an extra action which samples an action from π_p, which also allows the agent to use the pre-trained policy at test time if deemed beneficial. More formally, this amounts to training a policy over an expanded action set $\mathcal{A}^+ = \mathcal{A} \cup \{a_+\}$, where a_+ is resolved by sampling an action from π_p, $a' \sim \pi_p(s)$ (with $a' \in \mathcal{A}$). The additional action can be seen as an option that can be initiated from any state and always terminates after a single step. Note that selecting the option will lead to the same outcome as if the agent had selected a' as a primitive action, and we take advantage of this observation by using the return of following the option as target to fit both $Q(s, \pi_p(s))$ and $Q(s, a')$. Intuitively, this approach induces a bias that favours actions selected by π_p, accelerating the collection of rewarding transitions when the pre-trained policy is somewhat aligned with the downstream task. Otherwise, the agent can learn to ignore π_p as training progresses by selecting other actions.

Algorithm 1: Experience collection in BT

Input: Action set, $\mathcal{A}$; additional action, a_+; extended action set, $\mathcal{A}^+ = \mathcal{A} \cup \{a_+\}$; pre-trained policy, π_p; Q-value estimate for the current policy, $Q^\pi(s, a)\, \forall a \in \mathcal{A}^+$; probability of taking an exploratory action, ϵ; probability of starting a flight, ϵ_{levy}; flight length distribution, $\mathcal{D}(\mathbb{N})$

```
while True do
    n ← 0                    // flight length
    while episode not ended do
        Observe state s
        if n == 0 and random() ≤ ε_levy then
            n ~ D(N)         // sample length
        if n > 0 then
            n ← n − 1
            a ~ π_p(s)
        else
            if random() ≤ ε then
            a ~ Uniform(A+)  else
            a ← arg max_{a'∈A+} [Q^π(s, a')]
            if a == a_+ then  a ~ π_p(s)
        end
        Take action a
    end
end
```

4. Reward-free pre-training

It is a common practice to derive objectives for proxy tasks in order to drive learning in the absence of reward functions, and there exists a plethora of different approaches in the literature. Model-based approaches can learn world models from unsupervised interaction (Ha & Schmidhuber, 2018). However, the diversity of the training data will impact the accuracy of the model (Sekar et al., 2020) and deploying this type of approach in visually complex domains like Atari remains an open problem (Hafner et al., 2019). Unsupervised RL has also been explored through the lens of *empowerment* (Salge et al., 2014; Mohamed & Rezende, 2015), which studies agents that aim to discover intrinsic options (Gregor et al., 2016; Eysenbach et al., 2019). While these options can be leveraged by hierarchical agents (Florensa et al., 2017) or integrated within the universal successor features framework (Barreto et al., 2017; 2018; Borsa et al., 2019; Hansen et al., 2020), their potential lack of coverage generally limits their applicability to complex downstream tasks (Campos et al., 2020). An alternative objective is that of exploring the environment by finding policies that induce maximally entropic state distributions (Hazan et al., 2019; Lee et al., 2019), although this might become extremely inefficient in high-dimensional state spaces without proper priors (Liu & Abbeel, 2021; Yarats et al., 2021).

Recall that our goal is to devise a pre-training objective that can help reduce the amount of interaction needed by the agent to collect relevant experience when learning in a downstream task. We argue that such objective needs to meet two requirements. First, as suggested by results in other domains (Brown et al., 2020; Kaplan et al., 2020), it should scale gracefully as the amount of compute and experience used for pre-training are increased. This contrasts with the training regimes used in most unsupervised RL approaches, which use a relatively small amount of experience (Hansen et al., 2020; Liu & Abbeel, 2021; Yarats et al., 2021) when compared to distributed agents that do make use of rewards (Horgan et al., 2018; Espeholt et al., 2018; Kapturowski et al., 2019). Second, it must encourage the emergence of complex behaviors such as navigation or manipulation skills. It has been argued that exploring the environment efficiently will serve as a proxy for developing such behaviors (Kearns & Singh, 2002), and exploration bonuses have been shown to produce meaningful behavior in the absence of reward (Pathak et al., 2017; Burda et al.,

2018a). However, many exploration bonuses vanish over the course of training and thus may not be well-suited for a long unsupervised pre-training phase. It can be shown that many intrinsic rewards aim at maximizing the entropy of all states visited during training, and so the final policy does not necessarily exhibit exploratory behavior (Lee et al., 2019).

We propose to use Never Give Up (NGU) (Puigdomènech Badia et al., 2020b) as a means for training exploratory policies in an unsupervised setting. The NGU intrinsic reward proposes a curiosity-driven approach for training persistent exploratory policies which combines per-episode and life-long novelty. The per-episode novelty, r_t^{episodic}, rapidly vanishes over the course of an episode, and it is designed to encourage self-avoiding trajectories. It is computed by comparing a representation of the current observation, $f(s_t)$, to those of all the observations visited in the current episode, $M = \{f(s_0), f(s_1), \ldots, f(s_{t-1})\}$, where $f : \mathcal{S} \to \mathbb{R}^p$ is an embedding function trained using a self-supervised inverse dynamics model (Pathak et al., 2017). Such a mapping concentrates on the controllable aspects of the environment, ignoring all the variability present in the observation that is not affected by the action taken by the agent. The life-long novelty, α_t, slowly vanishes throughout training, and it is computed by using Random Network Distillation (RND) (Burda et al., 2018b). With this, the intrinsic reward r_t^{NGU} is defined as follows:

$$
\begin{aligned}
r_t^{\text{NGU}} &= r_t^{\text{episodic}} \cdot \min\left\{\max\left\{\alpha_t, 1\right\}, L\right\}, \\
r_t^{\text{episodic}} &= \frac{1}{\sqrt{\sum_{f(s_i)\in N_k} K(f(s_t), f(s_i))} + c}
\end{aligned}
\tag{1}
$$

where L is a fixed maximum reward scaling, N_k is the set containing the k-nearest neighbors of $f(s_t)$ in M, c is a constant and $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^+$ is a kernel function satisfying $K(x, x) = 1$ (which can be thought of as approximating pseudo-counts (Puigdomènech Badia et al., 2020b)). The episodic component of the reward in Equation 1 is reset by emptying M with each episode, thus the NGU reward does not vanish throughout the training process. This makes it suitable for driving learning in task-agnostic settings.

5. Experiments

Agents are evaluated in the Atari suite (Bellemare et al., 2013), a benchmark that presents a variety of challenges and that is a common test ground for RL agents with unsupervised pre-training (Hansen et al., 2020; Liu & Abbeel, 2021; Schwarzer et al., 2021). Experiments are run using the distributed R2D2 agent (Kapturowski et al., 2019) with 256 CPU actors and a single GPU learner. Policies use the same Q-Network architecture as Agent57 (Puigdomènech Badia et al., 2020a), which is composed by a convolutional torso followed by an LSTM (Hochreiter & Schmidhuber, 1997) and a dueling head (Wang et al., 2016). Hyperparameters and a detailed description of the full distributed setting are provided in the supplementary material. All reported results are the average over three random seeds.

Reward-free learning. The amount of task reward collected by unsupervised policies is often used as a proxy to measure their quality (Eysenbach et al., 2019). While the actual utility of these policies will not be revealed until they are leveraged for transfer, this proxy lets us evaluate whether the discovered behavior changes as longer pre-training budgets are allowed. We compare unsupervised NGU policies against VISR (Hansen et al., 2020) and APT (Liu & Abbeel, 2021), which utilize a small amount of supervised interaction to adapt the pre-trained policies. We also consider two additional unsupervised baselines: *(i)* a constant positive reward at each timestep that favours long episodes, which correlate with high scores in some games (Burda et al., 2018a), and *(ii)* RND (Burda et al., 2018b), which rewards life-long novelty. Note that the RND reward vanishes, but we include it in our analysis because it was previously used by Burda et al. (2018a) in this setting and implementation choices such as reward normalization may prevent it from fading in practice. Figure 2 (left) shows how the zero-shot transfer performance of unsupervised policies evolves during a long pre-training phase. NGU reaches the highest scores, but both NGU and RND eventually outperform VISR and APT even though these used supervised interaction. In Table 2 of Appendix C we show that unsupervised NGU policies largely outperform several other baselines using the standard pre-training and adaptation setting. These results highlight the importance of large-scale unsupervised pre-training in RL, similarly to the trend observed in supervised domains (Brown et al., 2020).

Transfer setting. Transfer approaches are typically evaluated in the Atari benchmark with a budget of 100k RL interactions with reward (400k frames), but we propose to allow a longer adaptation phase. Randomly initialized networks tend to overfit in these very low data regimes without strong regularization (Kostrikov et al., 2021), and we are interested in studying the impact of leveraging behavior both in isolation and combined with transfer via pre-trained weights. Moreover, since the pre-trained policies are already competent in the downstream tasks, 100k interactions are exhausted after few episodes and may be insufficient for improving performance. For these reasons, we provide results with up to 1.25B RL steps of supervised interaction (5B frames). This allows evaluating both convergence speed and asymptotic performance, while still being a relatively small budget for these distributed agents with hundreds of actors (Puigdomènech Badia et al., 2020a).

Transfer via behavior. We start by studying the impact of leveraging behavior in isolation, i.e. without transfer-

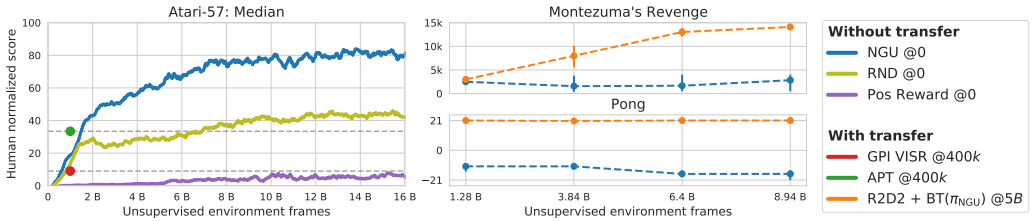

Figure 2. Performance as a function of the pre-training budget. @N represents the number of frames with reward utilized for transfer. **(Left)** Median human normalized score across the 57 games in the Atari suite. We observe the emergence of useful behavior when optimizing an intrinsic reward during a long unsupervised pre-training of 16B frames, which contrasts with the shorter pre-training of 1B frames in previous works (Hansen et al., 2020; Liu & Abbeel, 2021). **(Right)** Scores in the games of Montezuma's Revenge (sparse rewards) and Pong (dense reward), before and after transfer, as a function of the pre-training budget. A longer pre-training benefits transfer in hard exploration games even if the zero-shot transfer score of the unsupervised policies does not increase.

ring pre-trained weights, when learning in downstream tasks. We compare BT against two baselines that do not use pre-trained behavior, namely the standard R2D2 agent (Kapturowski et al., 2019) that uses ϵ-greedy policies for exploration (Sutton & Barto, 2018), as well as a variant of R2D2 with ϵz-greedy exploration (Dabney et al., 2021). Figure 3 shows that BT is superior to both baselines for any amount of environment interaction with rewards, converging faster early in training and also obtaining higher asymptotic performance. These results also demonstrate the generality of the proposed approach, as it is able to benefit from both RND and NGU policies. Note that BT performs particularly well in the set of six hard exploration games[1] defined by Bellemare et al. (2016), which is aligned with our intuition that reusing behavior helps overcoming the inefficiency associated to unstructured exploration. Figure 2 (right) confirms that a long pre-training phase is especially important in hard exploration games such as Montezuma's Revenge, even it they do not translate into higher zero-shot transfer scores, as it produces more exploratory behavior. On the other hand, the performance after transfer is independent of the amount of pre-training in dense reward games like Pong, where unstructured exploration is enough to reach optimal scores.

Ablation studies. In order to gain insight on each of the components in BT, we run experiments on a subset of 12 games[2] requiring different amounts of exploration and featuring both dense and sparse rewards. BT(π_{NGU}) achieves a median score of 368 in this subset, which compares favorably to the 196 median score of R2D2 with ϵ-greedy exploration. Removing either the extra action or the temporally-extended exploration reduces the median score of BT(π_{NGU}) to 224. These results suggest that the gains provided by both strategies are complementary, and both are responsible for the strong performance of BT. To provide further insight about the benefits of BT, Figure 4 reports the fraction of steps per episode in which the extra action is selected by the greedy policy. It hints at the emergence of a schedule over the usage of the pre-trained policy, which increases early in training and decays afterwards. We hypothesize that this is due to the fact that the unsupervised policies obtain large episodic returns, but their behavior is suboptimal when maximizing discounted rewards. These policies take many exploratory actions in between rewards, and so the agent eventually figures out more efficient strategies for reaching rewarding states by using primitive actions.

Transfer to multiple tasks. An appealing property of task-agnostic knowledge is that it can be leveraged to solve multiple tasks. In the RL setting, this can be evaluated by leveraging a single task-agnostic policy for solving multiple tasks (i.e. reward functions) in the same environment. We evaluate whether the unsupervised NGU policies can be useful beyond the standard Atari tasks by creating two alternative versions of Ms Pacman and Hero with different levels of difficulty. The goal in the modified version of Ms Pacman is to eat vulnerable ghosts, with pac-dots giving 0 (easy version) or -10 (hard version) points. In the modified version of Hero, saving miners gives a fixed return of 1000 points and dynamiting walls gives either 0 (easy version) or -300 (hard version) points. The rest of rewards are removed, e.g. eating fruit in Ms Pacman or the bonus for unused power units in Hero. Note that even in the easy version of the games exploration is harder than in their original counterparts, as there are no small rewards guiding the agent towards its goals. Exploration is even more challenging in the hard version of the games, as the intermediate rewards work as a deceptive signal that takes the agent away from its actual goal. In this case, finding rewarding behaviors requires a stronger commitment to an exploration strategy. Unsupervised NGU policies often achieve very low or even negative rewards in this setting, which contrasts with the strong performance they showed when evaluated under the standard game reward. Figure 5 shows that leveraging the behavior of pre-trained exploration policies provides important gains even in this adversarial scenario. These results

[1] `gravitar`, `montezuma_revenge`, `pitfall`, `private_eye`, `solaris`, `venture`

[2] Obtained by combining games used to tune hyperparameters in (Hansen et al., 2020) with games where ϵz-greedy provides clear gains over ϵ-greedy as per (Dabney et al., 2021): `asterix`, `bank_heist`, `frostbite`, `gravitar`, `jamesbond`, `montezuma_revenge`, `ms_pacman`, `pong`, `private_eye`, `space_invaders`, `tennis`, `up_n_down`.

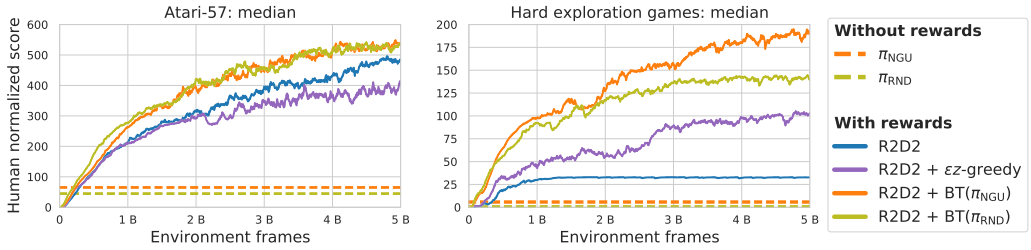

Figure 3. Median human normalized scores for R2D2-based agents trained from scratch. (**Left**) Full Atari suite. (**Right**) Subset of hard exploration games.

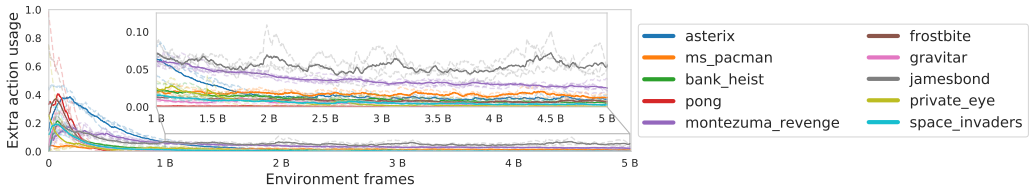

Figure 4. Usage of the extra action in BT(π_{NGU}), computed as the fraction of steps within an episode in which it is selected by the agent. The usage peaks early in training and slowly decreases afterwards as the new policy becomes stronger at the task.

suggest that the strong performance observed under the standard game rewards is not due to an alignment between the NGU reward and the game goals, but due to an efficient usage of pre-trained exploration policies.

Combining pre-trained behavior and weights. Our last batch of experiments focuses on studying transfer via pre-trained weights and its compatibility with BT. Policies are composed of a convolutional torso, an LSTM, and a dueling head. We consider two initialization strategies: a *partial initialization* approach that loads the torso and the LSTM, but initializes the head randomly; and a *full initialization* scheme where all weights are loaded. The former can be understood as transferring learned representations (Yarats et al., 2021), but deferring exploration to a random policy. On the other hand, the full initialization approach can be seen as directly transferring the policy and is usually referred to as fine-tuning the pre-trained policy (Mutti et al., 2021; Liu & Abbeel, 2021; Schwarzer et al., 2021). Note that these approaches only change how weights are initialized *before* training. As in previous experiments, all parameters in the new policy are trained and π_p is kept fixed when using BT. Figure 6 (top) compares agents with and without BT for different amounts of transfer via weights on the Atari benchmark. Loading pre-trained weights results in faster learning early in training, both with and without BT. The largest gains are observed in dense reward games, which translates into higher median scores across the full suite because most games belong to this category. Weights alone are not enough in hard exploration games, where leveraging the pre-trained policy via BT provides clear benefits. Perhaps surprisingly, we observe that transferring representations outperforms fine-tuning the pre-trained policy, and we hypothesize that the former is more robust to misalignments between the pre-trained policy and the downstream task. This intuition is further supported by the experiments on games with modified reward functions reported in Figure 6 (middle & bottom), where the faster learning provided by pre-trained weights often comes at the cost of lower end scores. On the other hand, BT is crucial in tasks with sparse and deceptive rewards and also benefits from pre-trained weights in tasks where positive transfer is observed.

6. Related work

Our work uses the experimental methodology presented by Hansen et al. (2020). Whereas that work only considered a fast, simplified adaptation process that limited the final performance on the downstream task, we focus on the more general case of using a previously trained policy to aid in solving the full RL problem. Hansen et al. (2020) use successor features to identify which of the pre-trained tasks best matches the true reward structure, which has previously been shown to work well for multi-task transfer (Barreto et al., 2018). Bagot et al. (2020) augments an agent with the ability to utilize another policy, which is learned in tandem based on an intrinsic reward function. This promising direction is complementary to our work, as it handles the case wherein there is no unsupervised pre-training phase.

Gupta et al. (2018) provides an alternative method to meta-learn a solver for reinforcement learning problems from unsupervised reward functions. This method utilizes gradient-based meta-learning (Finn et al., 2017), which makes the adaptation process standard reinforcement learning updates. This means that even if the downstream reward is far outside of the training distribution, final performance would not necessarily be affected. However, these methods are hard to scale to the larger networks considered here, and followup work (Jabri et al., 2019) changed to memory-based

Figure 5. Scores in Atari games with modified reward functions. We train a single task-agnostic policy per environment, and leverage it to solve three different tasks: the standard game reward, a task with sparse rewards (easy), and a variant of the same task with deceptive rewards (hard).

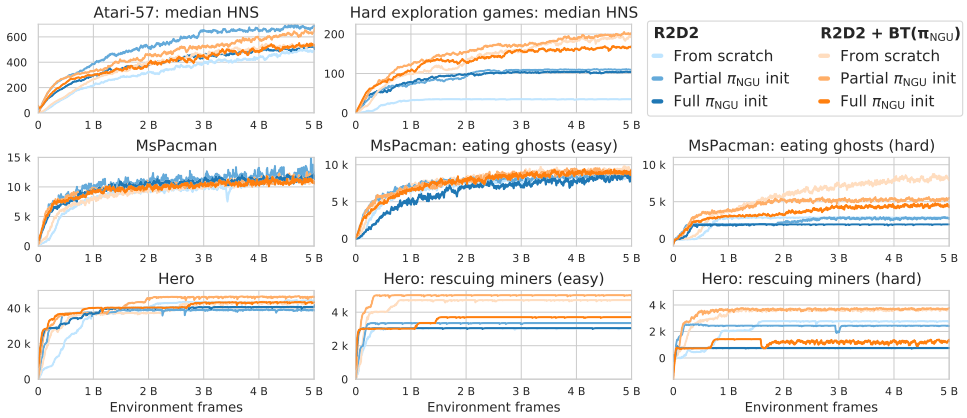

Figure 6. Performance of R2D2-based agents with different amounts of transfer via weights. Policies are composed of a CNN encoder followed by an LSTM and a dueling head. We compare training from scratch, loading all weights (Full π_{NGU} init) or all weights except those in the dueling head (Partial π_{NGU} init). (**Top**) Median human normalized scores (HNS) in the full Atari suite (left) and the subset of hard exploration games (right). (**Middle & Bottom**) Games with modified reward functions as in Figure 5.

meta-learning (Duan et al., 2016) which relies on information about rewards staying in the recurrent state. This makes it unsuitable to the sort of hard exploration problem our method excels at. Recent work has shown success in transferring representations learned in an unsupervised setting to reinforcement learning tasks (Stooke et al., 2020). Our representation transfer experiments suggest that this might handicap final performance, but the possibility also exists that different unsupervised objectives should be used for representation transfer and policy transfer.

7. Discussion

We studied the problem of transferring pre-trained behavior for exploration in reinforcement learning, an approach that is complementary to the common practice of transferring neural network weights. Our proposed approach, Behavior Transfer (BT), relies on the pre-trained policy for collecting experience in two different ways: (*i*) through temporally-extended exploration, which can be triggered with some probability at any step, and (*ii*) via one-step calls to the pre-trained policy based on value estimates. BT results in strong transfer performance when combined with exploratory policies pre-trained in the absence of reward, with the most important gains being observed in hard exploration tasks. These benefits are not due to an alignment between our pre-training and downstream tasks, as we also observed positive transfer in games where the pre-trained policy obtained low scores. In order to provide further evidence for this claim, we designed alternative tasks for Atari games involving hard exploration and deceptive rewards. Our transfer strategy outperformed all considered baselines in these settings, even when the pre-trained policy obtained very low or even negative scores, demonstrating the generality of the method. Besides disambiguating the role of the alignment between pre-training and downstream tasks, these experiments demonstrate the utility of a single task-agnostic policy for solving multiple tasks in the same environment. Finally, we also demonstrated that BT can be combined with transfer via neural network weights to provide further gains. Our experimental results highlight the importance of scale when training RL agents in reward-free settings, which is one of the key factors behind the recent success of unsupervised approaches in other domains. This contrasts with the small budgets considered for reward-free RL in previous works and motivates further research in unsupervised RL approaches that scale with increased data and compute. We argue that scale is one of the missing components in reward-free RL, and it will be a necessary condition to unfold its full potential.

References

Bagot, L., Mets, K., and Latré, S. Learning intrinsically motivated options to stimulate policy exploration. In *ICML Workshop on LifeLong Learning*, 2020.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. Successor features for transfer in reinforcement learning. In *NeurIPS*, 2017.

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Zidek, A., and Munos, R. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *ICML*, 2018.

Barreto, A., Hou, S., Borsa, D., Silver, D., and Precup, D. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 2020.

Barto, A. G. and Mahadevan, S. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 2003.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *NeurIPS*, 2016.

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*, 2013.

Borsa, D., Barreto, A., Quan, J., Mankowitz, D., Munos, R., van Hasselt, H., Silver, D., and Schaul, T. Universal successor features approximators. In *ICLR*, 2019.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018a.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018b.

Campos, V., Trott, A., Xiong, C., Socher, R., Giro-i Nieto, X., and Torres, J. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *ICML*, 2020.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

Dabney, W., Ostrovski, G., and Barreto, A. Temporally-extended ϵ-greedy exploration. In *ICLR*, 2021.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *ICML*, 2018.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2019.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR*, 2017.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

Gupta, A., Eysenbach, B., Finn, C., and Levine, S. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2019.

Hansen, S., Dabney, W., Barreto, A., Van de Wiele, T., Warde-Farley, D., and Mnih, V. Fast task inference with variational intrinsic successor features. In *ICLR*, 2020.

Hazan, E., Kakade, S. M., Singh, K., and Van Soest, A. Provably efficient maximum entropy exploration. In *ICML*, 2019.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

Hénaff, O. J., Razavi, A., Doersch, C., Eslami, S., and Oord, A. v. d. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 1997.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Jabri, A., Hsu, K., Gupta, A., Eysenbach, B., Levine, S., and Finn, C. Unsupervised curricula for visual meta-reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 10519–10531, 2019.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *ICLR*, 2019.

Kearns, M. and Singh, S. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 2002.

Kostrikov, I., Yarats, D., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *ICLR*, 2021.

Lee, L., Eysenbach, B., Parisotto, E., Xing, E., Levine, S., and Salakhutdinov, R. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

Liu, H. and Abbeel, P. Behavior from the void: Unsupervised active pre-training. *arXiv preprint arXiv:2103.04551*, 2021.

Liu, Y. and Brunskill, E. When simple exploration is sample efficient: Identifying sufficient conditions for random exploration to yield pac rl algorithms. *arXiv preprint arXiv:1805.09045*, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Mohamed, S. and Rezende, D. J. Variational information maximisation for intrinsically motivated reinforcement learning. In *NeurIPS*, 2015.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *NeurIPS*, 2016.

Mutti, M., Pratissoli, L., and Restelli, M. Task-agnostic exploration via policy gradient of a non-parametric state entropy estimate. In *AAAI*, 2021.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016a.

Osband, I., Van Roy, B., and Wen, Z. Generalization and exploration via randomized value functions. In *ICML*, 2016b.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Peng, J. and Williams, R. J. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*. Elsevier, 1994.

Puigdomènech Badia, A., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *ICML*, 2020a.

Puigdomènech Badia, A., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never give up: Learning directed exploration strategies. In *ICLR*, 2020b.

Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

Salge, C., Glackin, C., and Polani, D. Empowerment – an introduction. In *Guided Self-Organization: Inception*. Springer, 2014.

Schwarzer, M., Rajkumar, N., Noukhovitch, M., Anand, A., Charlin, L., Hjelm, R. D., Bachman, P., and Courville, A. Pretraining reward-free representations for data-efficient reinforcement learning. In *Self-Supervision for Reinforcement Learning Workshop - ICLR 2021*, 2021. URL https://openreview.net/forum?id=o5z9Le5drua.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. In *ICML*, 2020.

Stooke, A., Lee, K., Abbeel, P., and Laskin, M. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

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

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999.

Viswanathan, G. M., Afanasyev, V., Buldyrev, S., Murphy, E., Prince, P., and Stanley, H. E. Lévy flight search patterns of wandering albatrosses. *Nature*, 1996.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. Reinforcement learning with prototypical representations. In *ICML*, 2021.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*, 2014.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

A. Pseudo-code

Algorithm 2 provides pseudo-code for the flight logic that controls how the pre-trained policy is used for temporally-extended exploration. At each step, a flight is started with probability ϵ_{levy}. The duration of the flight is sampled from a heavy-tailed distribution, $\mathcal{D}(\mathbb{N})$, similarly to ϵz-greedy (c.f. Appendix B for more details). When not in a flight, the exploitative policy that maximizes the extrinsic reward is derived from the estimated Q-values using the ϵ-greedy operator. This ensures that all state-action pairs will be visited given enough time, as exploring only with π_p does not guarantee such property.

Algorithm 3 provides pseudo-code for the actor logic when using the augmented action set, $\mathcal{A}^+ = \mathcal{A} \cup \{\pi_p(s)\}$. It derives an ϵ-greedy policy over $|\mathcal{A}| + 1$ actions, where the $(|\mathcal{A}| + 1)$-th action is resolved by sampling from $\pi_p(s)$.

Algorithm 2: Experience collection pseudo-code for BT with temporally-extended exploration

```
Input: Action set A
Input: Q-value estimate for the current policy, Q^π(s, a) ∀a ∈ A
Input: Pre-trained policy, π_p
Input: Probability of starting a flight, ε_levy
Input: Flight length distribution, D(N)
while True do
    n ← 0                                   // flight length
    while episode not ended do
        Observe state s
        if n == 0 and random() ≤ ε_levy then
            n ∼ D(N)                        // sample from distribution over lengths
        end
        if n > 0 then
            n ← n − 1
            a ∼ π_p(s)
        else
            a ∼ ε-greedy[Q^π(s, a)]
        end
        Take action a
    end
end
```

Algorithm 3: Experience collection pseudo-code for BT with an extra action

```
Input: Action set 𝒜
Input: Additional action, a_+
Input: Extended action set, 𝒜^+ = 𝒜 ∪ {a_+}
Input: Pre-trained policy, π_p
Input: Q-value estimate for the current policy, Q^π(s, a) ∀a ∈ 𝒜^+
Input: Probability of taking an exploratory action, ϵ
while True do
    while episode not ended do
        Observe state s
        if random() ≤ ϵ then
            a ~ Uniform(𝒜^+)
        else
            a ← arg max_{a'∈𝒜^+} [Q^π(s, a')]
        end
        if a == a_+ then
            a ~ π_p(s)
        end
        Take action a
    end
end
```

B. Hyperparameters

All policies use the same Q-Network architecture as Agent57 (Puigdomènech Badia et al., 2020a), which is composed by a convolutional torso followed by an LSTM (Hochreiter & Schmidhuber, 1997) and a dueling head (Wang et al., 2016). When leveraging the behavior of the pre-trained policy to solve new tasks, we instantiate a new network with independent weights (c.f. Figure 7). One can initialize some of the components of the new network using pre-trained weights without tying their values (as in common fine-tuning approaches).

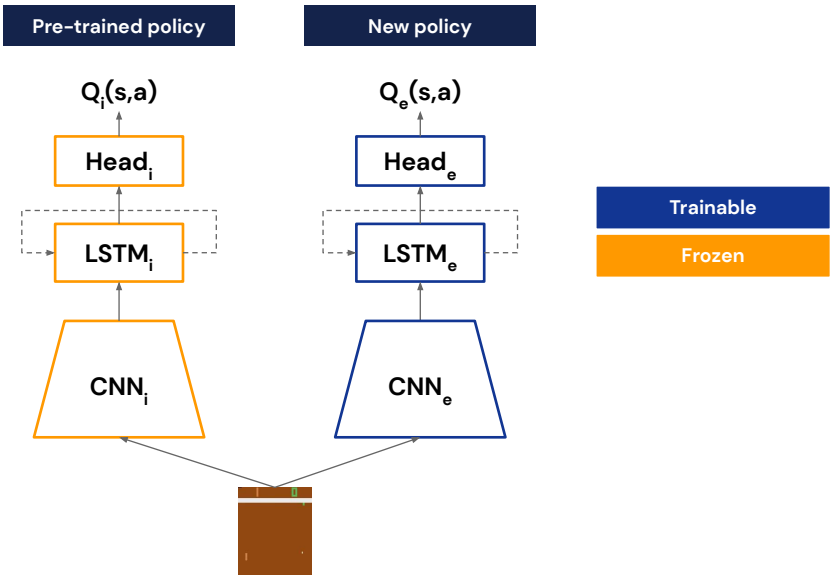

Figure 7. Q-Network architecture for the reinforcement learning stage. The networks use independent sets of parameters, and the weights of the pre-trained policy are kept fixed to preserve the learned behavior.

Table 1 summarizes the main hyperparameters of our method. The pre-trained policies were optimized using Retrace (Munos et al., 2016). Learning with rewards was performed with Peng's Q(λ) (Peng & Williams, 1994) instead, which we found to be much more data efficient in our experiments. The reason for this difference is that the benefits of Q(λ) were observed once unsupervised policies had been trained on all Atari games.

It should be noted that our ϵz-greedy baseline under-performs relative to Dabney et al. (2021). This is due to our hyper-parameters and setting being derived from Puigdomènech Badia et al. (2020b), which adopts the standard Atari pre-processing (e.g. gray scale images and frame stacking). In contrast, Dabney et al. (2021) use color images, no frame stacking, a larger neural network and different hyper-parameters (e.g. smaller replay buffer). Studying if the performance of NGU, RND and BT is preserved in this setting is an important direction for future work. We suspect that improving the performance of our ϵz-greedy ablation will also improve our method, since exploration flights are central to both.

Table 1. Hyperparameter values used in R2D2-based agents. The rest of hyperparameters use the values reported by Kapturowski et al. (2019).

Hyperparameter	**Value**
Number of actors	256
Actor parameter update interval	400 environment steps
Sequence length	160 (without burn-in)
Replay buffer size	12.5×10^4 part-overlapping sequences
Priority exponent	0.9
Importance sampling exponent	0
Learning rule (downstream tasks)	Q(λ), $\lambda = 0.7$
Learning rule (NGU pre-training)	Retrace(λ), $\lambda = 0.95$
Discount (downstream tasks)	0.99
Discount (NGU pre-training)	0.99
Minibatch size	64
Optimizer	Adam
Optimizer settings	$\varepsilon = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$
Learning rate	2×10^{-4}
Target network update interval	1500 updates
ϵ_{levy} distribution	Log-Uniform$[0, 0.1]$
Flight length distribution	Zeta with $\mu = 2$

C. Extended reward-free RL results

We compare the results of our unsupervised pre-training stage against other unsupervised approaches, standard RL algorithms in the low-data regime and methods that perform unsupervised pre-training followed by an adaptation stage. Since the considered intrinsic rewards are non-negative, we consider a baseline where the agent obtains a constant positive reward at each step in order to measure the performance of policies that seek to stay alive for as long as possible. Results for this baseline were already considered by Hansen et al. (2020) (*Pos Reward NSQ*), but we run our own version of this baseline using the distributed setting and longer pre-training of 16B frames considered in our experiments (*Pos Reward R2D2*). Table 2 shows that unsupervised RND and NGU outperform all baselines by a large margin, confirming the intuition that exploration is a good pre-training objective for the Atari benchmark. These results suggest that there is a strong correlation between exploration and the goals established by game designers (Burda et al., 2018a). In spite of the strong results, it is worth noting that unsupervised RND and NGU achieve lower scores than random policies in some games, and can be quite inefficient at collecting rewards in some environments (e.g. they needs long episodes to obtain high scores). These observations motivate the development of techniques to leverage these pre-trained policies without compromising performance even when there exists a misalignment between objectives.

Table 2. Atari Suite comparisons, adapted from Hansen et al. (2020) and Liu & Abbeel (2021). $@N$ represents the amount of RL interaction with reward utilized, with four frames observed at each iteration. *Mdn* and *M* are median and mean human normalized scores, respectively; > 0 is the number of games with better than random performance; and $> H$ is the number of games with human-level performance as defined in Mnih et al. (2015). **Top**: unsupervised learning only. **Mid**: data-limited RL. **Bottom**: RL with unsupervised pre-training.

	26 Game Subset Kaiser et al. (2019)				47 Game Subset Burda et al. (2018a)				Full 57 Games Mnih et al. (2015)			
Algorithm	Mdn	M	>0	$>H$	Mdn	M	>0	$>H$	Mdn	M	>0	$>H$
IDF Curiosity @0	–	–	–	–	8.46	24.51	34	5	–	–	–	–
RF Curiosity @0	–	–	–	–	7.32	29.03	36	6	–	–	–	–
Pos Reward NSQ @0	2.18	50.33	14	5	0.69	57.65	26	8	0.29	41.19	28	8
Pos Reward R2D2 @0	9.44	59.55	21	4	14.16	57.53	39	5	3.46	45.23	46	5
Q-DIAYN-5 @0	0.17	−3.60	13	0	0.33	−1.23	25	2	0.34	−2.18	30	2
Q-DIAYN-50 @0	−1.65	−21.77	4	0	−1.69	−16.26	8	0	−3.16	−20.31	9	0
VISR @0	5.60	81.65	19	5	4.04	58.47	35	7	3.77	49.66	40	7
RND@0	48.35	334.65	23	8	41.28	259.43	40	14	40.86	243.01	47	16
NGU @0	80.92	**494.54**	25	12	**96.10**	**310.27**	**45**	**23**	**81.72**	**320.06**	**52**	**27**
SimPLe @100k	9.79	36.20	26	4	–	–	–	–	–	–	–	–
DQN @10M	27.80	52.95	25	7	9.91	28.07	41	7	8.61	27.55	48	7
DQN @200M	**100.76**	267.51	**26**	**13**	–	–	–	–	80.81	239.29	46	20
Rainbow @100k	2.23	10.12	25	1	–	–	–	–	–	–	–	–
PPO @500k	20.93	43.74	25	7	–	–	–	–	–	–	–	–
NSQ @10M	8.20	33.80	22	3	7.29	29.47	37	4	6.80	28.51	43	5
SPR @100k	41.50	70.40	–	7	–	–	–	–	–	–	–	–
CURL @100k	17.50	38.10	–	2	–	–	–	–	–	–	–	–
DrQ @100k	28.42	35.70	–	2	–	–	–	–	–	–	–	–
Q-DIAYN-5 @100k	0.01	16.94	13	2	1.31	19.64	28	6	1.55	16.65	33	6
Q-DIAYN-50 @100k	−1.64	−27.88	3	0	−1.66	−16.74	8	0	−2.53	−24.13	9	0
RF VISR @100k	7.24	58.23	20	6	3.81	42.60	33	9	2.16	35.29	39	9
VISR @100k	9.50	128.07	21	7	9.42	121.08	35	11	6.81	102.31	40	11
GPI RF VISR @100k	5.55	58.77	20	5	4.24	48.38	34	9	3.60	40.01	40	10
GPI VISR @100k	6.59	111.23	22	7	11.70	129.76	38	12	8.99	109.16	44	12
MEPOL @100k	0.34	17.94	–	2	–	–	–	–	–	–	–	–
APT @100k	47.50	69.55	–	7	–	–	–	–	33.41	47.73	–	12

D. Extended Atari-57 results

Table 3. Atari Suite comparisons for R2D2-based agents. $@N$ represents the amount of frames with reward utilized, with four frames observed per RL interaction. *Mdn*, *M* and *CM* are median, mean and mean capped human normalized scores, respectively.

	Full 57 Games			Hard Exploration		
Algorithm	Mdn	M	CM	Mdn	M	CM
R2D2 $@1B$	229.75	864.69	84.56	31.07	39.40	34.75
R2D2 + ϵz-greedy $@1B$	204.52	578.73	85.11	42.55	53.90	46.21
R2D2 + BT(π_{NGU}) $@1B$	273.49	**1517.13**	86.38	**100.89**	**94.20**	63.95
R2D2 + BT(π_{RND}) $@1B$	**280.04**	1396.78	**87.43**	93.52	86.75	**67.40**
R2D2 $@5B$	490.12	1742.92	90.37	32.49	67.41	44.74
R2D2 + ϵz-greedy $@5B$	418.41	1275.86	92.49	103.62	95.46	67.85
R2D2 + BT(π_{NGU}) $@5B$	538.50	2262.21	**93.31**	**193.15**	**160.02**	76.92
R2D2 + BT(π_{RND}) $@5B$	**571.57**	**2304.19**	92.03	144.78	123.38	**76.93**

Table 4. Atari Suite comparisons with rewards for R2D2-based agents with different amounts of transfer via weights at 5B training frames. Policies are composed of a CNN encoder followed by an LSTM and a dueling head. We compare training from scratch, loading all weights (Full π_{NGU} init) or all weights except those in the dueling head (Partial π_{NGU} init). *Mdn*, *M* and *CM* are median, mean and mean capped human normalized scores, respectively. (**Top**) Without BT. (**Bottom**) With BT(π_{NGU}).

	Full 57 Games			Hard Exploration		
Algorithm	Mdn	M	CM	Mdn	M	CM
R2D2, from scratch	490.12	1742.92	90.37	32.49	67.41	44.74
R2D2, partial π_{NGU} init	**668.80**	2020.81	**93.00**	**109.33**	**123.40**	**67.18**
R2D2, full π_{NGU} init	507.58	**2359.25**	89.91	104.98	101.52	66.20
R2D2 + BT(π_{NGU}), from scratch	538.50	2262.21	93.31	193.15	160.02	76.92
R2D2 + BT(π_{NGU}), partial π_{NGU} init	**626.34**	1966.83	**94.07**	**200.32**	**164.54**	**76.93**
R2D2 + BT(π_{NGU}), full π_{NGU} init	529.78	**2467.02**	92.79	168.18	137.65	**76.93**

Table 5. Human normalized scores after 5B frames with rewards for R2D2-based agents at different percentiles. Note that the 50th percentile corresponds to the median score across the 57 games. We compare training from scratch, loading all weights (Full π_{NGU} init) or all weights except those in the dueling head (Partial π_{NGU} init).

	Percentile				
Method	50th	40th	20th	10th	5th
R2D2, from scratch	490.12	220.97	132.77	92.31	25.57
R2D2 + BT(π_{NGU}), from scratch	538.50	**316.30**	**163.20**	**104.41**	**65.17**
R2D2 + BT(π_{RND}), from scratch	**571.57**	279.97	133.61	87.05	54.27
R2D2, partial π_{NGU} init	**668.80**	434.02	164.62	105.08	53.59
R2D2 + BT(π_{NGU}), partial π_{NGU} init	626.34	**489.14**	**171.69**	**113.92**	**93.52**
R2D2, full π_{NGU} init	507.58	293.54	137.79	59.72	41.06
R2D2 + BT(π_{NGU}), full π_{NGU} init	**529.78**	**384.39**	**163.55**	**123.56**	**51.98**

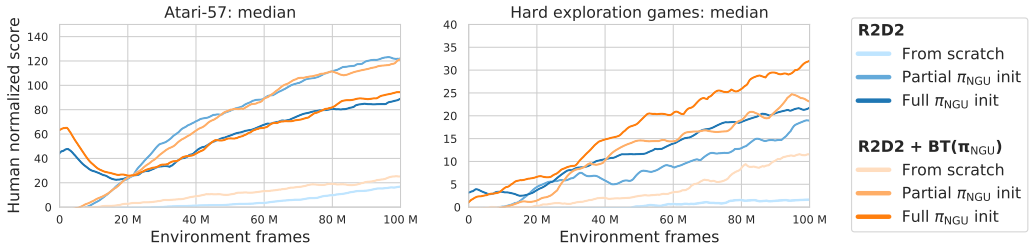

Figure 8. Median human normalized scores for R2D2-based agents with different amounts of transfer via weights during the first 100M frames of training. Policies are composed of a CNN encoder followed by an LSTM and a dueling head. We compare training from scratch, loading the CNN and the LSTM (Partial π_{NGU} init), and loading all weights including the dueling head (Full π_{NGU} init). (**Left**) Full Atari suite. (**Right**) Subset of hard exploration games.

E. Alternative reward functions

MsPacman: eating ghosts

- Pac-dots: 0 points (easy) or -10 points (hard)
- Eating vulnerable ghosts:
  - #1 in succession: 200 points
  - #2 in succession: 400 points
  - #3 in succession: 800 points
  - #4 in succession: 1600 points
- Other actions: 0 points

Hero: rescuing miners

- Dynamiting walls: 0 points (easy) or -300 points (hard)
- Rescuing a miner: 1000 points
- Other actions: 0 points

F. Distributed setting

All experiments are run using a distributed setting. The evaluation we do is also identical to the one done in R2D2 (Kapturowski et al., 2019): parallel evaluation workers, which share weights with actors and learners, run the Q-network

against the environment. This worker and all the actor workers are the two types of workers that draw samples from the environment. For Atari, we apply the standard DQN pre-processing, as used in R2D2. The next subsections describe how actors, evaluators, and learner are run in each stage.

F.1. Unsupervised stage

The computation of the intrinsic NGU reward, r_t^{NGU}, follows the method described in Puigdomènech Badia et al. (2020b, Appendix A.1). In particular, we use the version that combines episodic intrinsic rewards with the intrinsic reward from Random Network Distillation (RND) (Burda et al., 2018b).

We now describe the distributed setup used for NGU, which is largely the same as the one used for RND. Note that RND can be recovered by removing the components needed for the episodic reward.

Learner

- Sample from the replay buffer a sequence of intrinsic rewards r_t^{NGU}, observations x and actions a.
- Use Q-network to learn from (r_t^{NGU}, x, a) with Retrace (Munos et al., 2016) using the same procedure as in R2D2.
- Use last 5 frames of the sampled sequences to train the action prediction network in NGU. This means that, for every batch of sequences, all time steps are used to train the RL loss, whereas only 5 time steps per sequence are used to optimize the action prediction loss.
- Use last 5 frames of the sampled sequences to train the predictor of RND.

Actor

- Obtain x_t and r_{t-1}^{NGU}.
- With these inputs, compute forward pass of R2D2 to obtain a_t.
- With x_t, compute r_t^{NGU} using the embedding network in NGU.
- Insert x_t, a_t and r_t^{NGU} in the replay buffer.
- Step on the environment with a_t.

Evaluator

- Obtain x_t and r_{t-1}^{NGU}.
- With these inputs, compute forward pass of R2D2 to obtain a_t.
- With x_t, compute r_t^{NGU} using the embedding network in NGU.
- Step on the environment with a_t.

Distributed training

As in R2D2, we train the agent with a single GPU-based learner and a fixed discount factor γ. All actors collect experience using the same policy, but with a different value of ϵ. This differs from the original NGU agent, where each actor runs a policy with a different degree of exploratory behavior and discount factor.

In the replay buffer, we store fixed-length sequences of (x, a, r) tuples. These sequences never cross episode boundaries. Given a single batch of trajectories we unroll both online and target networks on the same sequence of states to generate value estimates. We use prioritized experience replay with the same prioritization scheme proposed in (Kapturowski et al., 2019).

F.2. Transfer with BT

LEARNER

- Sample from the replay buffer a sequence of extrinsic rewards r_t, observations x and actions a.
- (expanded action set) Duplicate transitions collected with π_p and relabel the duplicates with the primitive action taken by π_p when acting.
- Use Q-network to learn from (r_t, x, a) with Peng's Q(λ) (Peng & Williams, 1994) using the same procedure as in R2D2.

ACTOR

- (once per episode) Sample ϵ_{levy}.
- Obtain x_t.
- If not on a flight, start one with probability ϵ_{levy}.
- If on a flight, compute forward pass with π_p to obtain a_t. Otherwise, compute forward pass of R2D2 to obtain a_t. If $a_t = |\mathcal{A}| + 1$, $a_t \leftarrow \pi_p(x)$.
- Insert x_t, a_t and r_t in the replay buffer.
- Step on the environment with a_t.

EVALUATOR

- Obtain x_t.
- Compute forward pass of R2D2 to obtain a_t. If $a_t = |\mathcal{A}| + 1$, $a_t \leftarrow \pi_p(x)$.
- Step on the environment with a_t.

DISTRIBUTED TRAINING

As in R2D2, we train the agent with a single GPU-based learner and a fixed discount factor γ. All actors collect experience using the same policy, but with a different value of ϵ.

In the replay buffer, we store fixed-length sequences of (x, a, r) tuples. These sequences never cross episode boundaries. Given a single batch of trajectories we unroll both online and target networks on the same sequence of states to generate value estimates. We use prioritized experience replay with the same prioritization scheme proposed in (Kapturowski et al., 2019).

G. Intrinsic rewards

G.1. Random Network Distillation

The RND (Burda et al., 2018b) intrinsic reward is computed by introducing a random, untrained convolutional network $g : \mathcal{S} \to \mathbb{R}^d$, and training a network $\hat{g} : \mathcal{S} \to \mathbb{R}^d$ to predict the outputs of g on all the observations that are seen during training by minimizing the prediction error $\text{err}_{\text{RND}}(s_t) = ||\hat{g}(s_t; \theta) - g(s_t)||^2$ with respect to θ. The intuition is that the prediction error will be large on states that have been visited less frequently by the agent. The dimensionality of the random embedding, d, is a hyperparameter of the algorithm.

The RND intrinsic reward is obtained by normalising the prediction error. In this work, we use a slightly different normalization from that reported in (Burda et al., 2018b). The RND reward at time t is given by

$$r_t^{\text{RND}} = \frac{\text{err}_{\text{RND}}(s_t)}{\sigma_e} \tag{2}$$

where σ_e is the running standard deviation of $\text{err}_{\text{RND}}(s_t)$.

G.2. Never Give Up

The NGU intrinsic reward modulates an episodic intrinsic reward, r_t^{episodic}, with a life long signal α_t:

$$r_t^{\text{NGU}} = r_t^{\text{episodic}} \cdot \min\left\{\max\left\{\alpha_t, 1\right\}, L\right\}, \tag{3}$$

where L is a fixed maximum reward scaling. The life-long novelty signal is computed using RND with the normalisation:

$$\alpha_t = \frac{\text{err}_{\text{RND}}(s_t) - \mu_e}{\sigma_e} \tag{4}$$

where $\text{err}_{\text{RND}}(x_t)$ is the prediction error described in Appendix G.1, and μ_e and σ_e are its running mean and standard deviation, respectively. The episodic intrinsic reward at time t is computed according to formula:

$$r_t^{\text{episodic}} = \frac{1}{\sqrt{\sum_{f(s_i)\in N_k} K(f(s_t), f(s_i))} + c} \tag{5}$$

where N_k is the set containing the k-nearest neighbors of $f(s_t)$ in M, c is a constant and $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^+$ is a kernel function satisfying $K(x, x) = 1$ (which can be thought of as approximating pseudo-counts (Puigdomènech Badia et al., 2020b)). Algorithm 4 shows a detailed description of how the episodic intrinsic reward is computed. Below we describe the different components used in Algorithm 4:

- M: episodic memory containing at time t the previous embeddings $\{f(s_0), f(s_1), \ldots, f(s_{t-1})\}$. This memory starts empty at each episode
- k: number of nearest neighbours
- $N_k = \{f(s_i)\}_{i=1}^k$: set of k-nearest neighbours of $f(s_t)$ in the memory M; we call $N_k[i] = f(s_i) \in N_k$ for ease of notation
- K: kernel defined as $K(x, y) = \frac{\epsilon}{\frac{d^2(x,y)}{d_m^2} + \epsilon}$ where ϵ is a small constant, d is the Euclidean distance and d_m^2 is a running average of the squared Euclidean distance of the k-nearest neighbors
- c: pseudo-counts constant
- ξ: cluster distance
- s_m: maximum similarity

Algorithm 4: Computation of the episodic intrinsic reward at time t: r_t^{episodic}.

Input : M; k; $f(s_t)$; c; ϵ; ξ; s_m; d_m^2
Output : r_t^{episodic}

Compute the k-nearest neighbours of $f(s_t)$ in M and store them in a list N_k
Create a list of floats d_k of size k
/* The list d_k will contain the distances between the embedding $f(s_t)$ and its neighbours N_k. */
for $i \in \{1, \ldots, k\}$ **do**
  $d_k[i] \leftarrow d^2(f(s_t), N_k[i])$
end
Update the moving average d_m^2 with the list of distances d_k
/* Normalize the distances d_k with the updated moving average d_m^2. */
$d_n \leftarrow \frac{d_k}{d_m^2}$
/* Cluster the normalized distances d_n i.e. they become 0 if too small and 0_k is a list of k zeros. */
$d_n \leftarrow \max(d_n - \xi, 0_k)$
/* Compute the Kernel values between the embedding $f(s_t)$ and its neighbours N_k. */
$K_v \leftarrow \frac{\epsilon}{d_n + \epsilon}$
/* Compute the similarity between the embedding $f(s_t)$ and its neighbours N_k. */
$s \leftarrow \sqrt{\sum_{i=1}^{k} K_v[i]} + c$
/* Compute the episodic intrinsic reward at time t: r_t^i. */
if $s > s_m$ **then**
  $r_t^{\text{episodic}} \leftarrow 0$
else
  $r_t^{\text{episodic}} \leftarrow 1/s$

H. Scores per game

Table 6. Results per game for R2D2-based agents at 5B training frames.

Game	R2D2	R2D2 + ϵz-greedy	R2D2 + BT(π_{NGU})	R2D2 + BT(π_{RND})
alien	10831.17 ± 2114.29	14634.02 ± 1109.15	$\mathbf{15657.57 \pm 1717.96}$	12844.24 ± 1447.72
amidar	$\mathbf{11761.67 \pm 1560.86}$	6784.28 ± 718.05	10394.96 ± 891.60	7730.43 ± 670.76
assault	$\mathbf{15940.72 \pm 3531.69}$	9177.28 ± 2170.26	15060.31 ± 740.63	11533.24 ± 809.49
asterix	472812.21 ± 222663.81	374966.62 ± 135810.51	$\mathbf{630663.91 \pm 82753.46}$	468724.08 ± 120822.86
asteroids	45716.28 ± 3642.38	$\mathbf{147005.85 \pm 44313.45}$	31957.42 ± 15540.09	37455.64 ± 9263.04
atlantis	1514724.43 ± 10941.36	1132188.04 ± 43551.36	1491384.23 ± 5978.05	$\mathbf{1545954.35 \pm 9001.60}$
bank heist	965.63 ± 133.72	1058.75 ± 135.46	13913.32 ± 3529.15	$\mathbf{82132.27 \pm 101709.64}$
battle zone	292553.41 ± 18196.77	$\mathbf{312367.76 \pm 43554.18}$	258533.57 ± 22865.64	285925.87 ± 44912.86
beam rider	18472.45 ± 1977.78	$\mathbf{22403.95 \pm 1596.92}$	16301.02 ± 1853.73	15619.99 ± 2048.77
berzerk	12343.83 ± 3331.54	3846.56 ± 1723.24	8359.80 ± 201.10	$\mathbf{14687.68 \pm 401.76}$
bowling	141.64 ± 4.52	156.32 ± 8.11	174.27 ± 0.10	$\mathbf{196.01 \pm 57.42}$
boxing	99.96 ± 0.03	99.94 ± 0.06	$\mathbf{100.00 \pm 0.00}$	99.98 ± 0.03
breakout	432.65 ± 27.35	393.19 ± 35.12	$\mathbf{441.21 \pm 15.08}$	429.38 ± 15.52
centipede	189502.66 ± 31388.08	$\mathbf{358841.20 \pm 73578.20}$	178635.17 ± 17227.15	196880.46 ± 24278.18
chopper command	611393.11 ± 65206.69	697655.53 ± 215090.74	573055.88 ± 75343.57	$\mathbf{797052.58 \pm 52012.04}$
crazy climber	$\mathbf{229992.57 \pm 17738.33}$	212001.76 ± 1853.07	226821.26 ± 3608.19	198736.17 ± 7631.83
defender	$\mathbf{547238.15 \pm 2579.38}$	516521.06 ± 11969.59	540124.74 ± 4488.40	524003.44 ± 1316.59
demon attack	143662.42 ± 88.16	141352.18 ± 3848.73	$\mathbf{143762.91 \pm 106.75}$	143578.47 ± 25.05
double dunk	$\mathbf{23.99 \pm 0.02}$	23.88 ± 0.06	23.85 ± 0.15	23.93 ± 0.05
enduro	2358.37 ± 3.32	2359.08 ± 1.03	$\mathbf{2361.56 \pm 1.03}$	2350.39 ± 8.42
fishing derby	12.80 ± 77.79	$\mathbf{64.74 \pm 0.59}$	52.58 ± 0.32	62.11 ± 5.59
freeway	$\mathbf{33.87 \pm 0.08}$	33.77 ± 0.03	33.79 ± 0.08	33.79 ± 0.07
frostbite	9287.24 ± 167.11	8504.41 ± 940.72	$\mathbf{17692.42 \pm 2871.83}$	9419.45 ± 188.92
gopher	$\mathbf{117398.58 \pm 2485.82}$	84140.40 ± 12919.83	113716.78 ± 3966.91	94670.35 ± 2285.63
gravitar	6123.08 ± 103.19	5798.68 ± 735.59	$\mathbf{8373.70 \pm 1260.75}$	7428.57 ± 2459.91
hero	$\mathbf{46048.07 \pm 6970.26}$	39700.22 ± 4379.84	40825.09 ± 3736.25	42959.86 ± 7950.56
ice hockey	32.43 ± 30.64	30.65 ± 28.17	$\mathbf{60.36 \pm 4.94}$	57.96 ± 0.90
jamesbond	$\mathbf{6056.14 \pm 1643.52}$	3843.92 ± 118.35	1484.87 ± 489.66	2870.03 ± 907.76
kangaroo	14672.37 ± 187.16	14730.99 ± 114.20	$\mathbf{15965.79 \pm 36.61}$	15128.66 ± 188.17
krull	10081.04 ± 594.10	10171.52 ± 399.81	$\mathbf{406596.00 \pm 55547.76}$	316960.78 ± 217091.10
kung fu master	200721.64 ± 2265.35	171591.29 ± 8516.87	196638.89 ± 456.09	$\mathbf{610699.23 \pm 60053.99}$
montezuma revenge	1478.38 ± 1114.20	1467.77 ± 1104.72	$\mathbf{12086.71 \pm 1217.76}$	6266.67 ± 471.40
ms pacman	$\mathbf{11212.85 \pm 103.23}$	7511.39 ± 406.77	10996.90 ± 262.74	10656.00 ± 356.46
name this game	32138.12 ± 2156.95	$\mathbf{37343.04 \pm 1917.73}$	30252.11 ± 884.84	28746.14 ± 1798.77
phoenix	$\mathbf{712101.72 \pm 62738.09}$	80611.18 ± 25316.56	553429.34 ± 24278.55	283686.99 ± 172323.63
pitfall	-0.19 ± 0.15	-12.34 ± 4.20	-0.39 ± 0.39	$\mathbf{-0.03 \pm 0.04}$
pong	20.93 ± 0.01	20.49 ± 0.10	20.90 ± 0.01	$\mathbf{20.94 \pm 0.01}$
private eye	23592.22 ± 11876.55	$\mathbf{50770.82 \pm 14984.92}$	40435.54 ± 51.04	40480.67 ± 38.23
qbert	$\mathbf{24343.75 \pm 1904.89}$	16975.13 ± 1332.44	16057.31 ± 318.87	10990.08 ± 7241.50
riverraid	$\mathbf{32325.07 \pm 1185.15}$	30582.53 ± 638.47	28550.32 ± 2298.03	30566.86 ± 1764.50
road runner	$\mathbf{423191.07 \pm 53071.15}$	88890.04 ± 24971.18	251261.09 ± 31741.38	248661.22 ± 19416.63
robotank	97.23 ± 1.22	$\mathbf{108.92 \pm 4.79}$	98.45 ± 2.85	100.57 ± 6.32
seaquest	$\mathbf{188771.84 \pm 20759.57}$	175745.09 ± 120718.82	86605.86 ± 55065.85	38185.98 ± 22949.18
skiing	-29854.11 ± 85.79	-30060.81 ± 142.32	-30121.95 ± 70.62	$\mathbf{-29589.38 \pm 69.40}$
solaris	17741.02 ± 5340.46	16127.73 ± 2975.20	$\mathbf{24366.59 \pm 4868.05}$	18727.45 ± 4806.17
space invaders	3621.76 ± 5.81	3547.78 ± 35.31	30609.21 ± 7141.11	$\mathbf{46704.49 \pm 7017.79}$
star gunner	$\mathbf{223536.63 \pm 48548.34}$	179698.69 ± 12194.36	171294.31 ± 23185.79	156691.39 ± 16704.19
surround	$\mathbf{8.24 \pm 0.48}$	1.48 ± 8.12	5.86 ± 1.44	-3.62 ± 4.79
tennis	7.99 ± 22.56	7.98 ± 22.51	$\mathbf{23.96 \pm 0.01}$	7.97 ± 22.56
time pilot	$\mathbf{139931.67 \pm 70521.78}$	71768.84 ± 2933.22	44936.87 ± 137.49	77711.97 ± 4735.53
tutankham	324.02 ± 4.26	311.65 ± 8.62	$\mathbf{420.36 \pm 30.13}$	357.26 ± 14.22
up n down	529363.05 ± 16813.20	394984.70 ± 34313.42	562739.02 ± 8527.59	$\mathbf{585355.01 \pm 4718.67}$
venture	0.00 ± 0.00	1833.85 ± 43.73	$\mathbf{2110.64 \pm 55.39}$	1910.15 ± 13.98
video pinball	454023.46 ± 377076.03	107071.98 ± 67142.18	463141.28 ± 426927.92	$\mathbf{646671.78 \pm 403584.41}$
wizard of wor	$\mathbf{40833.65 \pm 4776.81}$	38275.31 ± 4177.41	30453.12 ± 2470.20	30399.63 ± 2345.83
yars revenge	279765.86 ± 27370.20	250483.70 ± 54593.32	$\mathbf{280333.48 \pm 69704.31}$	200850.59 ± 72885.67
zaxxon	56059.14 ± 3217.77	66099.28 ± 8520.19	$\mathbf{67611.78 \pm 6226.04}$	59926.08 ± 5834.47

Table 7. Results per game for R2D2 agents with different amounts of transfer via weights at 5B training frames. Policies are composed of a CNN encoder followed by an LSTM and a dueling head. We compare training from scratch, loading all weights (Full π_{NGU} init) or all weights except those in the dueling head (Partial π_{NGU} init).

Game	From scratch	Partial π_{NGU} init	Full π_{NGU} init
alien	10831.17 ± 2114.29	$\mathbf{27299.78 \pm 5730.57}$	18027.35 ± 6731.75
amidar	11761.67 ± 1560.86	$\mathbf{13647.09 \pm 3380.90}$	3518.30 ± 2353.96
assault	$\mathbf{15940.72 \pm 3531.69}$	14653.32 ± 2047.43	12533.61 ± 1001.68
asterix	472812.21 ± 222663.81	$\mathbf{789344.47 \pm 80638.00}$	676662.54 ± 8536.94
asteroids	45716.28 ± 3642.38	$\mathbf{73298.12 \pm 22688.38}$	23127.43 ± 5425.84
atlantis	1514724.43 ± 10941.36	1537659.81 ± 7693.86	$\mathbf{1556234.51 \pm 9709.74}$
bank heist	965.63 ± 133.72	1841.77 ± 52.75	$\mathbf{5816.24 \pm 3137.60}$
battle zone	292553.41 ± 18196.77	$\mathbf{301715.60 \pm 12875.97}$	248939.89 ± 31788.00
beam rider	$\mathbf{18472.45 \pm 1977.78}$	16179.19 ± 4179.36	11040.66 ± 799.77
berzerk	12343.83 ± 3331.54	16888.63 ± 2330.72	$\mathbf{25465.10 \pm 9886.89}$
bowling	141.64 ± 4.52	170.36 ± 16.55	$\mathbf{180.78 \pm 2.94}$
boxing	99.96 ± 0.03	$\mathbf{99.97 \pm 0.05}$	99.94 ± 0.07
breakout	432.65 ± 27.35	$\mathbf{520.19 \pm 64.65}$	487.51 ± 49.14
centipede	189502.66 ± 31388.08	$\mathbf{528000.27 \pm 10403.62}$	500534.38 ± 9267.05
chopper command	611393.11 ± 65206.69	$\mathbf{937637.69 \pm 57836.24}$	764150.71 ± 44756.44
crazy climber	229992.57 ± 17738.33	$\mathbf{275735.70 \pm 15244.51}$	246498.24 ± 12319.58
defender	$\mathbf{547238.15 \pm 2579.38}$	534656.86 ± 2880.53	523660.11 ± 2604.26
demon attack	$\mathbf{143662.42 \pm 88.16}$	143592.16 ± 77.27	143574.75 ± 69.35
double dunk	$\mathbf{23.99 \pm 0.02}$	$\mathbf{23.99 \pm 0.02}$	23.83 ± 0.06
enduro	2358.37 ± 3.32	$\mathbf{2359.39 \pm 8.12}$	2353.16 ± 1.30
fishing derby	12.80 ± 77.79	$\mathbf{68.70 \pm 2.46}$	59.22 ± 2.55
freeway	$\mathbf{33.87 \pm 0.08}$	33.83 ± 0.06	33.79 ± 0.04
frostbite	9287.24 ± 167.11	$\mathbf{161595.33 \pm 32917.44}$	10307.96 ± 1087.09
gopher	$\mathbf{117398.58 \pm 2485.82}$	113094.41 ± 4837.16	102781.75 ± 9613.77
gravitar	6123.08 ± 103.19	$\mathbf{7090.19 \pm 1359.52}$	5174.90 ± 544.76
hero	$\mathbf{46048.07 \pm 6970.26}$	43982.29 ± 4124.79	40628.07 ± 4008.99
ice hockey	32.43 ± 30.64	$\mathbf{69.57 \pm 1.18}$	47.67 ± 10.59
jamesbond	6056.14 ± 1643.52	$\mathbf{6109.60 \pm 1643.75}$	3979.12 ± 1233.92
kangaroo	14672.37 ± 187.16	14863.32 ± 259.85	$\mathbf{15192.97 \pm 832.48}$
krull	10081.04 ± 594.10	11806.49 ± 580.05	$\mathbf{372307.71 \pm 161921.43}$
kung fu master	200721.64 ± 2265.35	200305.15 ± 5711.26	$\mathbf{207401.69 \pm 1755.69}$
montezuma revenge	1478.38 ± 1114.20	$\mathbf{2666.30 \pm 235.18}$	2500.00 ± 0.00
ms pacman	11212.85 ± 103.23	$\mathbf{11795.03 \pm 640.73}$	11509.67 ± 563.98
name this game	32138.12 ± 2156.95	$\mathbf{33811.87 \pm 2091.30}$	29242.89 ± 1113.73
phoenix	712101.72 ± 62738.09	$\mathbf{812093.31 \pm 42328.98}$	801952.54 ± 33211.40
pitfall	$\mathbf{-0.19 \pm 0.15}$	-1.43 ± 1.17	-0.61 ± 0.60
pong	20.93 ± 0.01	$\mathbf{20.96 \pm 0.01}$	20.79 ± 0.13
private eye	23592.22 ± 11876.55	$\mathbf{30345.57 \pm 10971.52}$	28653.02 ± 9512.72
qbert	24343.75 ± 1904.89	40943.28 ± 16722.72	$\mathbf{62018.46 \pm 34865.08}$
riverraid	32325.07 ± 1185.15	$\mathbf{35995.19 \pm 825.70}$	35845.18 ± 3486.49
road runner	$\mathbf{423191.07 \pm 53071.15}$	311557.84 ± 59675.36	279988.24 ± 58503.61
robotank	97.23 ± 1.22	$\mathbf{111.78 \pm 4.63}$	91.93 ± 2.09
seaquest	188771.84 ± 20759.57	$\mathbf{629817.31 \pm 145648.54}$	31735.24 ± 31257.98
skiing	-29854.11 ± 85.79	$\mathbf{-29550.10 \pm 495.22}$	-29981.62 ± 564.13
solaris	17741.02 ± 5340.46	$\mathbf{29751.08 \pm 2076.41}$	22269.53 ± 6584.51
space invaders	3621.76 ± 5.81	41357.74 ± 8968.52	$\mathbf{42695.22 \pm 7148.08}$
star gunner	$\mathbf{223536.63 \pm 48548.34}$	212821.27 ± 19723.78	129058.91 ± 11260.80
surround	$\mathbf{8.24 \pm 0.48}$	6.86 ± 0.27	-3.29 ± 8.36
tennis	7.99 ± 22.56	$\mathbf{23.93 \pm 0.02}$	23.74 ± 0.16
time pilot	$\mathbf{139931.67 \pm 70521.78}$	65101.20 ± 7622.18	49957.68 ± 602.83
tutankham	324.02 ± 4.26	$\mathbf{333.37 \pm 10.62}$	312.19 ± 4.32
up n down	529363.05 ± 16813.20	572472.73 ± 4512.30	$\mathbf{595047.70 \pm 1976.45}$
venture	0.00 ± 0.00	1930.36 ± 32.97	$\mathbf{1958.13 \pm 48.98}$
video pinball	$\mathbf{454023.46 \pm 377076.03}$	113036.77 ± 3633.15	108849.58 ± 4753.59
wizard of wor	40833.65 ± 4776.81	$\mathbf{46931.25 \pm 1708.52}$	19100.00 ± 1930.29
yars revenge	279765.86 ± 27370.20	284565.01 ± 29764.09	$\mathbf{294877.82 \pm 52551.36}$
zaxxon	56059.14 ± 3217.77	$\mathbf{77649.92 \pm 15901.03}$	75850.01 ± 10805.13

Table 8. Results per game for R2D2+BT(π_{NGU}) agents with different amounts of transfer via weights at 5B training frames. Policies are composed of a CNN encoder followed by an LSTM and a dueling head. We compare training from scratch, loading all weights (Full π_{NGU} init) or all weights except those in the dueling head (Partial π_{NGU} init).

Game	From scratch	Partial π_{NGU} init	Full π_{NGU} init
alien	15657.57 ± 1717.96	$\mathbf{35441.47 \pm 3848.23}$	32822.74 ± 3181.51
amidar	10394.96 ± 891.60	$\mathbf{11564.75 \pm 1726.50}$	8185.74 ± 346.35
assault	$\mathbf{15060.31 \pm 740.63}$	12617.35 ± 3267.80	12992.70 ± 2783.13
asterix	630663.91 ± 82753.46	731452.01 ± 71621.96	$\mathbf{815753.95 \pm 91022.04}$
asteroids	31957.42 ± 15540.09	53916.46 ± 12925.49	$\mathbf{77368.55 \pm 17900.09}$
atlantis	1491384.23 ± 5978.05	1512404.85 ± 10047.26	$\mathbf{1544673.15 \pm 5590.54}$
bank heist	$\mathbf{13913.32 \pm 3529.15}$	11674.48 ± 1694.59	8565.69 ± 4070.27
battle zone	258533.57 ± 22865.64	$\mathbf{321572.13 \pm 32083.18}$	206177.95 ± 27251.07
beam rider	16301.02 ± 1853.73	17465.26 ± 4954.96	$\mathbf{21680.37 \pm 5991.66}$
berzerk	8359.80 ± 201.10	15824.26 ± 5556.26	$\mathbf{16161.60 \pm 2848.82}$
bowling	174.27 ± 0.10	$\mathbf{229.04 \pm 6.36}$	201.86 ± 25.21
boxing	$\mathbf{100.00 \pm 0.00}$	99.99 ± 0.01	99.83 ± 0.12
breakout	441.21 ± 15.08	469.52 ± 31.70	$\mathbf{474.30 \pm 37.00}$
centipede	178635.17 ± 17227.15	362169.27 ± 43577.84	$\mathbf{525652.45 \pm 19649.69}$
chopper command	573055.88 ± 75343.57	766193.62 ± 109233.61	$\mathbf{860939.78 \pm 116076.87}$
crazy climber	226821.26 ± 3608.19	224084.35 ± 7322.03	$\mathbf{256189.16 \pm 21605.50}$
defender	$\mathbf{540124.74 \pm 4488.40}$	525077.01 ± 6084.08	503190.45 ± 32182.12
demon attack	$\mathbf{143762.91 \pm 106.75}$	143537.16 ± 139.44	143554.33 ± 63.61
double dunk	23.85 ± 0.15	$\mathbf{23.90 \pm 0.07}$	23.81 ± 0.11
enduro	$\mathbf{2361.56 \pm 1.03}$	2352.77 ± 4.01	2353.81 ± 3.50
fishing derby	52.58 ± 0.32	$\mathbf{64.20 \pm 2.52}$	52.53 ± 1.24
freeway	$\mathbf{33.79 \pm 0.08}$	33.69 ± 0.13	33.57 ± 0.09
frostbite	17692.42 ± 2871.83	$\mathbf{20847.02 \pm 15492.47}$	19716.28 ± 13424.59
gopher	$\mathbf{113716.78 \pm 3966.91}$	105370.92 ± 12883.78	101383.00 ± 7891.20
gravitar	$\mathbf{8373.70 \pm 1260.75}$	8358.32 ± 1022.94	6104.39 ± 1215.73
hero	40825.09 ± 3736.25	$\mathbf{45837.71 \pm 194.61}$	43076.13 ± 4119.55
ice hockey	60.36 ± 4.94	$\mathbf{66.97 \pm 0.72}$	30.15 ± 4.43
jamesbond	1484.87 ± 489.66	1137.12 ± 89.95	$\mathbf{4119.56 \pm 490.29}$
kangaroo	$\mathbf{15965.79 \pm 36.61}$	15862.51 ± 234.05	15855.24 ± 224.30
krull	$\mathbf{406596.00 \pm 55547.76}$	154118.68 ± 179080.83	350784.05 ± 205164.12
kung fu master	$\mathbf{196638.89 \pm 456.09}$	193105.00 ± 5378.00	195990.25 ± 4969.60
montezuma revenge	12086.71 ± 1217.76	$\mathbf{12714.19 \pm 824.60}$	11472.65 ± 629.87
ms pacman	10996.90 ± 262.74	$\mathbf{11337.68 \pm 1176.17}$	10770.08 ± 1005.72
name this game	30252.11 ± 884.84	$\mathbf{30656.34 \pm 373.97}$	28103.24 ± 2023.96
phoenix	553429.34 ± 24278.55	510548.66 ± 236677.91	$\mathbf{805269.57 \pm 37169.21}$
pitfall	-0.39 ± 0.39	-0.44 ± 0.32	$\mathbf{-0.01 \pm 0.02}$
pong	20.90 ± 0.01	$\mathbf{20.93 \pm 0.02}$	20.19 ± 0.93
private eye	40435.54 ± 51.04	$\mathbf{40472.03 \pm 39.10}$	40448.67 ± 40.02
qbert	16057.31 ± 318.87	15983.72 ± 888.74	$\mathbf{17954.73 \pm 302.05}$
riverraid	28550.32 ± 2298.03	$\mathbf{34591.91 \pm 831.68}$	34268.13 ± 149.84
road runner	251261.09 ± 31741.38	307342.61 ± 41017.79	$\mathbf{308258.62 \pm 84372.17}$
robotank	98.45 ± 2.85	$\mathbf{103.82 \pm 2.98}$	90.17 ± 8.09
seaquest	86605.86 ± 55065.85	$\mathbf{259408.14 \pm 144362.40}$	88376.19 ± 105086.08
skiing	-30121.95 ± 70.62	$\mathbf{-29786.38 \pm 401.06}$	-29878.47 ± 289.38
solaris	$\mathbf{24366.59 \pm 4868.05}$	24111.78 ± 2745.89	19355.27 ± 4102.09
space invaders	30609.21 ± 7141.11	43675.67 ± 6763.52	$\mathbf{51318.54 \pm 4277.08}$
star gunner	$\mathbf{171294.31 \pm 23185.79}$	138390.23 ± 6320.24	135606.16 ± 8098.51
surround	5.86 ± 1.44	$\mathbf{6.89 \pm 1.03}$	-5.48 ± 0.88
tennis	23.96 ± 0.01	$\mathbf{23.97 \pm 0.01}$	23.86 ± 0.06
time pilot	44936.87 ± 137.49	$\mathbf{65721.81 \pm 3213.79}$	53053.12 ± 4275.77
tutankham	$\mathbf{420.36 \pm 30.13}$	385.05 ± 5.02	342.17 ± 4.79
up n down	562739.02 ± 8527.59	581518.52 ± 5198.05	$\mathbf{582923.55 \pm 2849.01}$
venture	2110.64 ± 55.39	$\mathbf{2308.26 \pm 14.97}$	2054.24 ± 65.41
video pinball	463141.28 ± 426927.92	133315.36 ± 70576.86	$\mathbf{640269.38 \pm 330619.65}$
wizard of wor	30453.12 ± 2470.20	$\mathbf{34648.51 \pm 4182.32}$	26076.64 ± 2060.62
yars revenge	280333.48 ± 69704.31	$\mathbf{320777.97 \pm 64750.83}$	267861.48 ± 81193.44
zaxxon	67611.78 ± 6226.04	75165.80 ± 4030.42	$\mathbf{82868.90 \pm 10160.99}$

Table 9. Final scores per game in our ablation study after 5B frames. We consider versions of BT(π_{NGU}) where the pre-trained policy is used for temporally-extended exploitation (flights), as an extra action (action), or both.

Game	R2D2	R2D2 + BT(π_{NGU}) (flights)	R2D2 + BT(π_{NGU}) (action)	R2D2 + BT(π_{NGU})
asterix	472812.21 ± 222663.81	$\mathbf{630663.91 \pm 82753.46}$	512869.97 ± 109039.77	618352.67 ± 103940.46
bank heist	965.63 ± 133.72	$\mathbf{13913.32 \pm 3529.15}$	11052.82 ± 6848.39	12424.39 ± 1443.95
frostbite	9287.24 ± 167.11	9114.77 ± 511.31	10506.15 ± 3653.44	$\mathbf{17692.42 \pm 2871.83}$
gravitar	6123.08 ± 103.19	6308.62 ± 78.84	7228.15 ± 1842.07	$\mathbf{8373.70 \pm 1260.75}$
jamesbond	$\mathbf{6056.14 \pm 1643.52}$	1615.21 ± 602.84	3962.10 ± 798.68	1484.87 ± 489.66
montezuma revenge	1478.38 ± 1114.20	11152.10 ± 664.82	6433.33 ± 372.68	$\mathbf{13265.91 \pm 372.02}$
ms pacman	$\mathbf{11212.85 \pm 103.23}$	10996.90 ± 262.74	10648.85 ± 796.10	10839.60 ± 445.00
pong	20.93 ± 0.01	20.90 ± 0.01	$\mathbf{20.94 \pm 0.04}$	20.88 ± 0.02
private eye	23592.22 ± 11876.55	$\mathbf{40492.85 \pm 27.35}$	37029.10 ± 2437.54	40435.54 ± 51.04
space invaders	3621.76 ± 5.81	$\mathbf{30671.58 \pm 4418.89}$	3597.41 ± 21.06	30609.21 ± 7141.11
tennis	7.99 ± 22.56	8.00 ± 22.54	-7.15 ± 22.00	$\mathbf{23.96 \pm 0.01}$
up n down	529363.05 ± 16813.20	562739.02 ± 8527.59	550665.39 ± 1852.05	$\mathbf{566938.61 \pm 2428.52}$

Table 10. Final scores per task in Atari games with modified reward functions. We report training results for the standard game reward, a variant with sparse rewards (easy), and a task with deceptive rewards (hard). Despite the pre-trained policy might obtain low or even negative scores in some of the tasks, committing to its exploratory behavior eventually lets the agent discover strategies that lead to high returns.

Game	R2D2	R2D2 + ϵz-greedy	Fine-tuning π_{NGU}	π_{NGU}	R2D2 + BT(π_{NGU})
Ms Pacman: original	$\mathbf{11407 \pm 122}$	8099 ± 868	8359 ± 2117	1360	10984 ± 665
Ms Pacman: ghosts (easy)	8375 ± 577	4322 ± 932	8356 ± 551	146	$\mathbf{8789 \pm 651}$
Ms Pacman: ghosts (hard)	2836 ± 26	4018 ± 1025	1891 ± 1342	-898	$\mathbf{7868 \pm 1085}$
Hero: original	$\mathbf{43762 \pm 4918}$	39018 ± 3262	46848 ± 1199	9298	42675 ± 3905
Hero: miners (easy)	3000 ± 0	3000 ± 0	3000 ± 0	1351	$\mathbf{4665 \pm 470}$
Hero: miners (hard)	2677 ± 23	2155 ± 95	700 ± 0	-1473	$\mathbf{3547 \pm 122}$

I. Learning curves

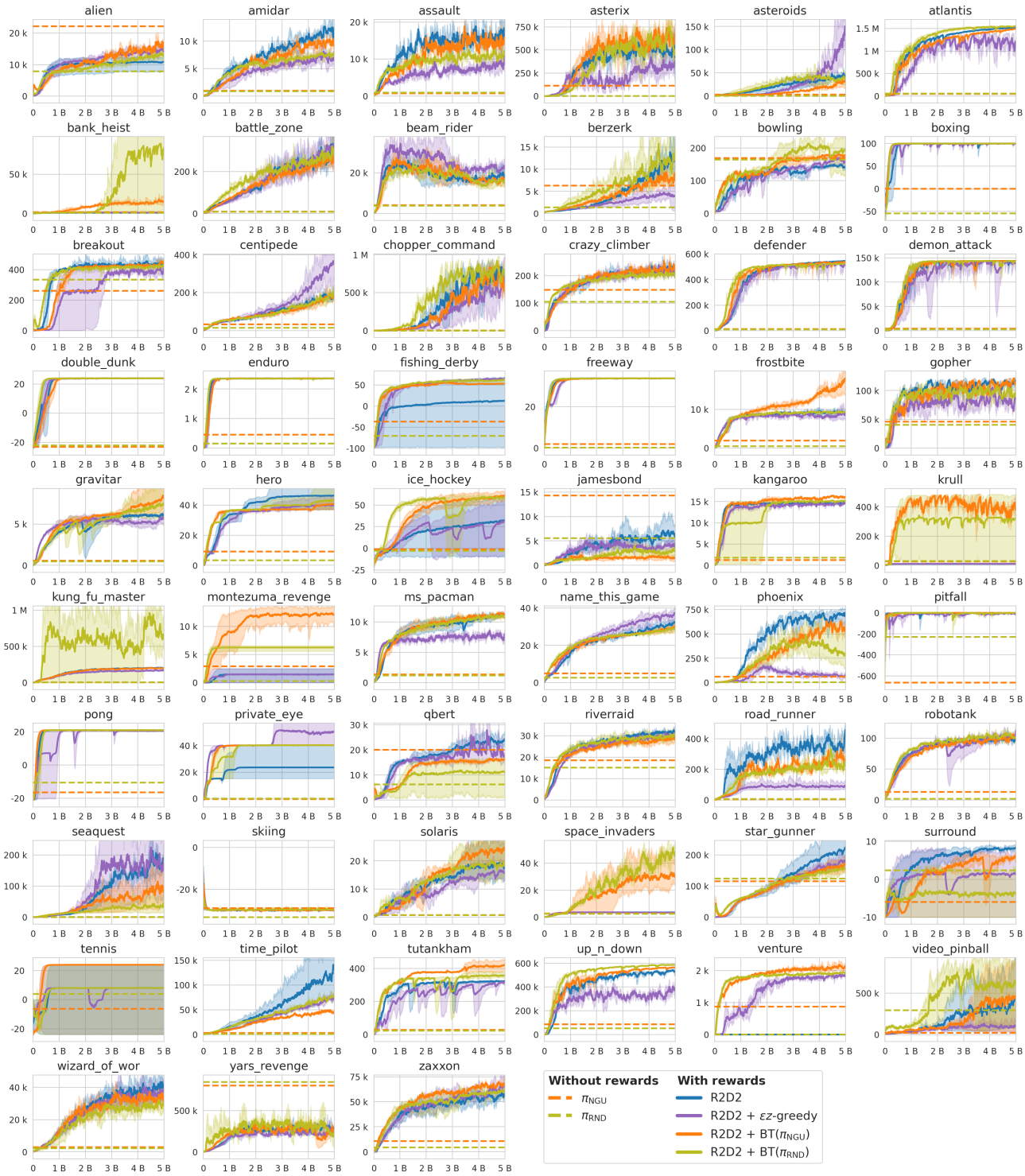

Figure 9. Training curves in all 57 Atari games for R2D2-based agents. Shading shows maximum and minimum over 3 runs, while dark lines indicate the mean.

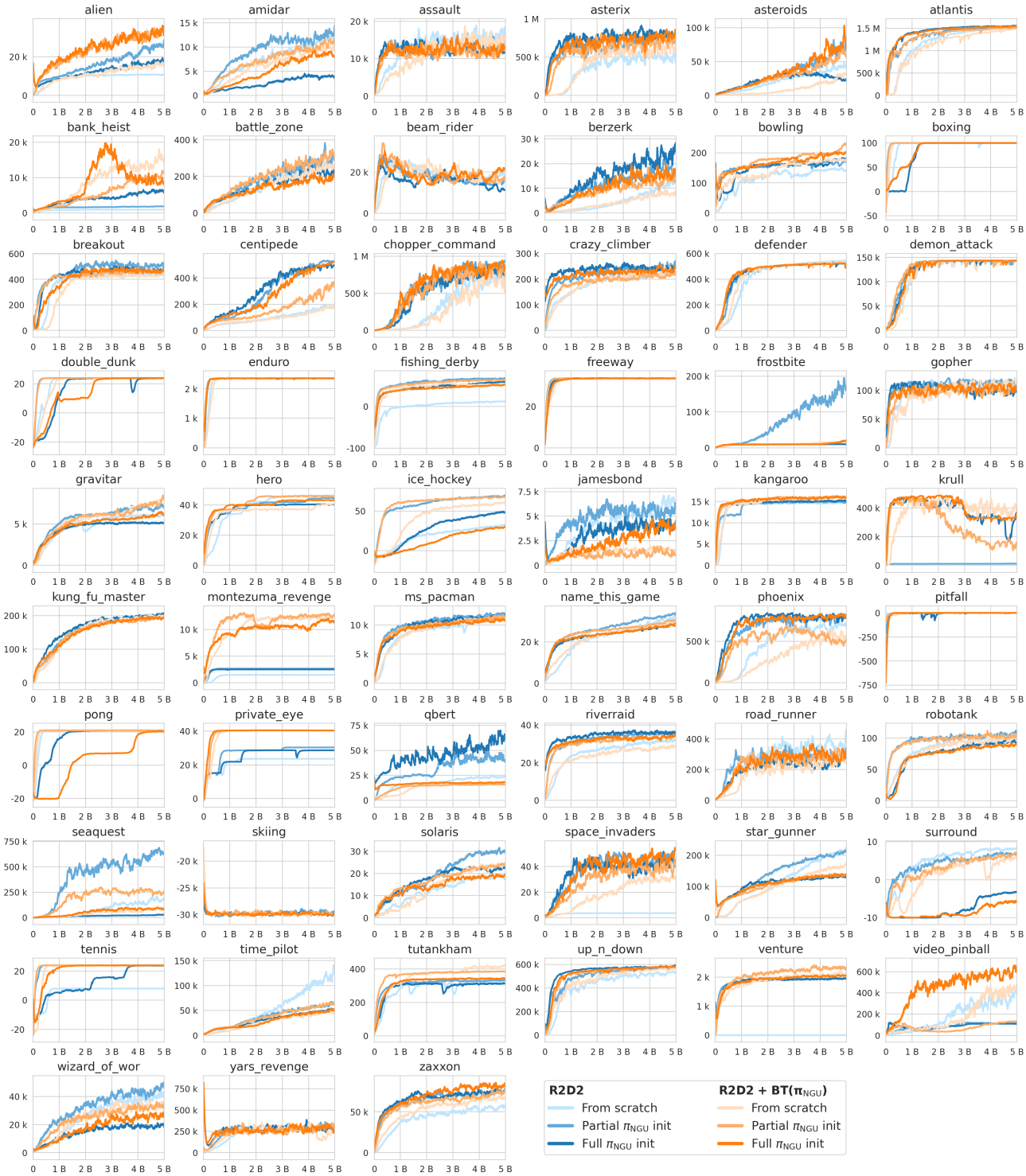

Figure 10. Training curves in all 57 Atari games for R2D2-based agents with different amounts of transfer via weights. Policies are composed of a CNN encoder followed by an LSTM and a dueling head. We compare training from scratch, loading all weights (Full π_{NGU} init) or all weights except those in the dueling head (Partial π_{NGU} init). Shading with maximum and minimum over runs is not shown for clarity, but all plots report the mean over 3 seeds.

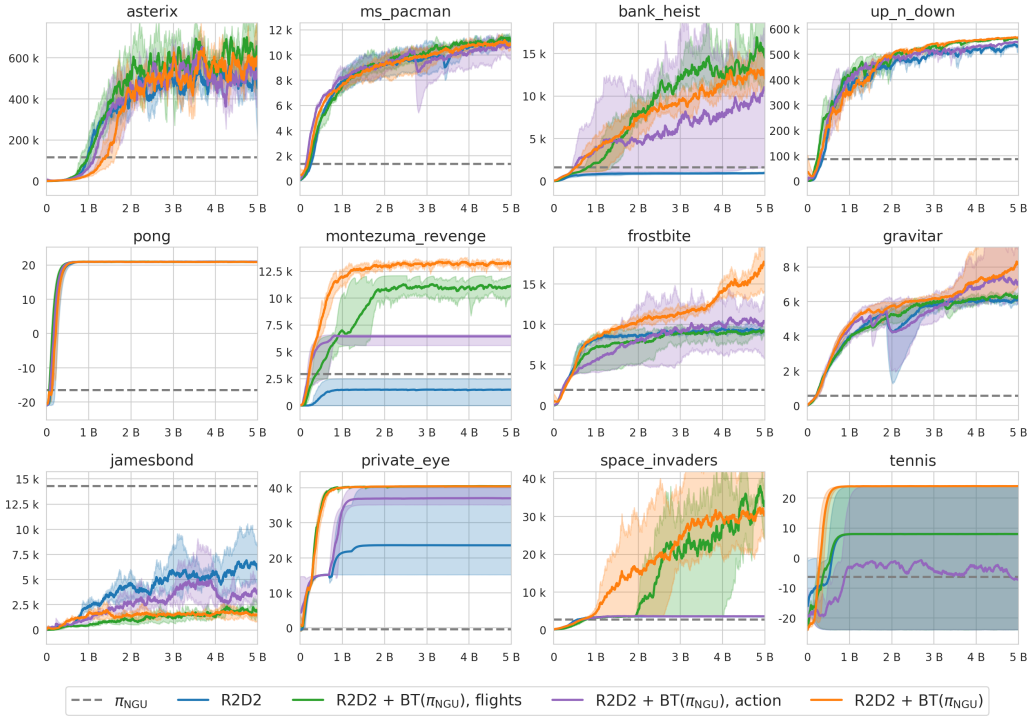

Figure 11. Training curves for ablation experiments. Shading shows maximum and minimum over 3 runs, while dark lines indicate the mean. Both ablations of BT offer benefits over the baseline, but in different sets of games. Combining them retains the best of both methods, and boosts performance even further in some games.