A Demonstration is Worth a Million Exploration Steps?

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep reinforcement learning has proven an effective method to solve many intricate tasks, yet it still struggles in data efficiency and generalization to novel scenarios. Recent approaches to deal with this include (1) unsupervised pretraining of the agent in an environment without reward signals, and (2) training the agent using offline data coming from various possible sources. In this paper we propose to consider both of these approaches together and argue that this results in a more realistic setting where different types of data are available, and fast online adaptation to new tasks is required. Towards this goal we extend the Unsupervised RL Benchmark to include access to both unsupervised exploratory data, and offline expert demonstrations, when testing the agents online performance on novel tasks in the environment. Using this setup we solve unaddressed issues in previous work. Specifically, we show how to make unsupervised data more effective by using a reward predictor that is trained from a small amount of supervised offline and online data, and we demonstrate how world models can serve as a way to consolidate agent training from various types of data, leading to faster online adaptation.

1 Introduction

Deep reinforcement learning (RL) has achieved remarkable success in addressing complex control tasks, yet a significant challenge persists in its limited ability to generalize to novel tasks. While transfer learning practices excel in simpler supervised tasks, RL traditionally treats each task in isolation, hindering the utilization of knowledge from prior experiences.

This creates two related big challenges: first, this setup is extremely inefficient because of the vast compute that is necessary to train policies from scratch, when even small changes in the task requires the agent to expose itself to millions of experiences in the environment; and second, this results in brittle policies that fail when facing perturbations to the task or environment.

Two recent approaches try to deal with this problem, namely unsupervised pretraining and using offline data. In the first approach, an agent is first pretrained using interactions in a similar environment, but without access to a specific task and its reward function. The assumption is that this can lead to a better initialization of the network weights for the second phase, when the agent starts to receive a reward signal and uses it to finetune its weights towards the desired task.

In the second approach, the agent first acquires offline data that contains demonstrations of interactions with the environment and the task reward. The agent can then use this data in different ways to extract the optimal policy for the task at hand.

In this paper we propose to unify these two approaches. In our setup an agent has unlimited access to unsupervised data in the environment, and in addition has access to a small amount of expert offline data. The main question we ask is how can the agent use this data to learn a new task faster. We

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

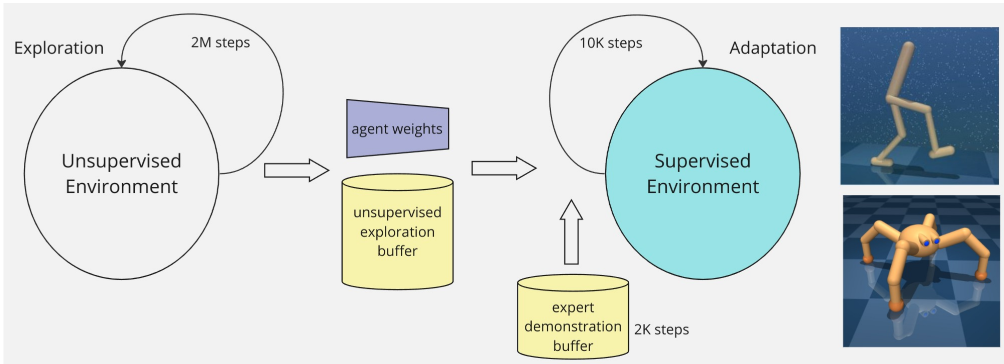

Figure 1: The proposed setup, and the two environments we use.

argue that (1) it is better to treat unsupervised data as offline data rather than a method to pretrain the agent's weights, and (2) rather than measuring the ability to learn from offline data alone, it is better to measure its usefulness by the acceleration it provides to subsequent online reinforcement learning.

Recent work [18] has already shown a step in the direction of unifying offline and unsupervised data, as it shows that for offline RL, using data that was acquired through unsupervised exploration has benefits over using the data from some reward maximizing policy. However the setting is not completely realistic as it assumes the ground truth reward function is known after the data was collected. Here we show how to get away from this assumption, by training a reward predictor using a few supervised offline expert demonstrations, and a small amount of supervised online interactions.

To further explore possible methods for marrying unsupervised and offline data, we consider world models. A recent paper [13] showed that world models are effective in an unsupervised pretraining setup, achieving excellent results in the Unsupervised RL Benchmark (URLB) [8]. We claim that the reason for this is that unsupervised pretraining is best used as a source of exploratory data rather than a way to achieve better initialization of the policy weights. By extending the method in Rajeswar et al. [13], we show significant improvements in adapting to novel tasks in an order of magnitude less online steps. We also show that access to a few expert demonstrations further accelerates online training in the URLB.

Following our findings, we propose this setup as an extension to the Unsupervised RL Benchmark that studies the unified space of using unsupervised exploratory data, supervised expert demonstrations, and supervised online interactions. We believe this is a natural setup to start benchmarking, as it represents the realistic setting where different possible data sources are available, and extremely rapid adaptation is required. Furthermore, our results on this setup suggests that a world model is a convenient method to fuse different data sources, leading to a method that effectively consolidates policy training using unsupervised exploratory data, expert demonstrations and online interactions.

Our main contributions can be summarized as following:

1. We propose to study a unified problem dealing with unsupervised pretraining, offline expert demonstrations, and online interactions in the environment. Towards this goal we publish an extension to the Unsupervised RL Benchmark, that can serve as an initial test-bed for this setup.

2. Compared to previous work on unsupervised offline RL [18], we drop the assumption of a known ground-truth reward function, and propose a way to train a reward predictor using a small amount of supervised offline and online data. Furthermore, in contrast to previous work on state space only, we show results in pixel space, making the overall setup more realistic.

3. Compared to previous work on using world models for unsupervised pretraining [13], we improve the training method, leading to better performance in fast adaptation, and we show how to incorporate offline data of a few expert demonstrations, leading to an even larger improvment of the results on the URLB. Our results suggest that world models are an effective tool to merge different types of data sources for RL, without loosing information in the way.

All data and code implementing the benchmark and tested methods will be made available upon publication.

2 Preliminaries and Related Work

2.1 Unsupervised Reinforcement Learning Benchmark

The Unsupervised Reinforcement Learning Benchmark (URLB) Laskin et al. [8] is a benchmark that compares the adaptation capabilities of different RL unsupervised exploration algorithms. The setup is separated into two stages: pretraining and fine-tuning. During the pretraining stage, a policy is trained on an intrinsic reward, which is specified by the tested exploration method. At the second stage, the resulted policy from the previous stage is used for fine-tuning on a specific task from the same domain as the one used during pre-training. The underlining RL algorithm is DrQv2 [17], which is a variant of DDPG [10].

In the pretraining stage, the tested exploration methods are separated into knowledge based, data based and competence methods. Knowledge-based methods focus on maximizing the entropy of visited states. Competence methods focus on maximizing the entropy of the learned policy. Data-based methods learn an explicit skill vector w by maximizing the mutual information between the encoded observation and skill. For each exploration method, the pretraining stage is performed environments from the *dm_control* suit: Walker, Quadruped and Jaco. Here we focus on Walker and Quadruped, as they present a more interesting challenge of exploration vs. fast adaptation.

In the finetuning stage, the exploration strategy used during pretraining is evaluated on set of tasks. First, the policy weights and part of the critic network are loaded from the pre-training stage. Then, the policy is trained on the task using the baseline RL algorithm DrQv2. The method is evaluated after 100K steps of finetuning, which is used as a measure of adaptation to a novel tasks.

2.2 Intrinsic Reward Models

Intrinsic reward method are used to facilitate better exploration in sparse reward environments. The idea is to use an additional reward to promote the visiting of novel states. Intrinsic reward methods are specially important when dealing with unsupervised environments, as they form the only source of reward. In this work we focus on four leading methods for exploration as used in Laskin et al. [7] and Rajeswar et al. [13]. The methods we consider are ICM [12], RND [1], RE3 [14], and Latent Bayesian Surprise (LBS) [11]. The latter is used in DreamerMPC [13], on which we base our method.

2.3 ExORL

ExORL [18] is a different benchmark that tests ways to utilize unsupervised exploration data from the perspective of offline RL. In URLB, the setup builds on pretraining a policy and later finetuning on a specific task. In contrast, ExORL assumes there is access to the reward function, that can be used to fix the unsupervised exploration buffer, which originally does not contain the reward signal. This is done without collecting any additional data. In this work, they argue that using the whole unsupervised buffer as the finetuning data source is more beneficial then using the policy that was used during the unsupervised data collection stage. All this, assuming the reward function is known and is easy to re-compute for each task.

2.4 World Model Pretraining

A major development in model based RL is the success of learned world models, which allow simulating environment transitions, and can facilitate sample efficient training. A learned world model is a data driven model that can estimate the distribution over the next state given a history of previous transitions:

$$p_\theta(s_{t+1}, r_{t+1} | s_t, a_t, s_{t-1}, \dots). \tag{1}$$

In essence, a well trained world model can replace a complex simulation of the environment. This, in turn, allows training the agent with almost no interaction with the actual environment, when the underlining environment dynamics stay the same. Additionally, planning techniques can be used by generating possible futures from the current states.

An example of such an architecture for world models is Dreamer [3] which uses an RNN to learn the environment dynamics in a sequential way. In addition, it learns the reward model $R(s, a)$. This allows generating partial trajectories, using some policy, and training an off-policy agent on the generated data. Recently, excellent results were achieved on the tasks tested in URLB. The method, which we call here **DreamerMPC** [13], uses the Dreamer architecture as a world model, where the data collected during the pretraining stage is used to train it, and LBS is used as the intrinsic reward. Additionally, during the fine-tuning stage, a planning strategy called Model Predictive Control (MPC) [4] is used to generate trajectories.

In MPC, the planning policy consists of a network that predicts the value function $V_\xi(s)$ and Gaussian action sequence distribution estimator $p(a_1, a_2, \ldots, a_t|s_0)$. For each state, several planning iterations are done, to gradually find the optimal trajectory and then the first planned action is used to proceed in the actual environment. For each planning iteration, given the sequence distribution, an expected return is estimated using the world model, and the value function. The return estimation is done in the following way:

1. the state s_0 is received from the environment
2. an initialized Gaussian sequence distribution $p(a_1, a_2, \ldots, a_t|s_0)$ is used to generate a sequence of actions.
3. the world model $p_\theta(s_{t+1}, r_{t+1}|s_t, a_t)$ is used to predict the reward at each time step until s_t
4. the rest of the expected return is calculated by $V_\xi(s_{t+1})$

Top sampled sequences are used to update the distribution, and another set of sequences are sampled. This process continues for a set amount of iterations. DreamerMPC used a Dreamer based world model and MPC, and showed superior performance on a variety of tasks compared to standard RL policy optimization via SGD.

2.5 Offline RL

In offline RL, rather than collecting data by interacting with the environment, data is collected by a different, usually unknown method, and used by the agent as an dataset to learn the policy. Data can come from expert demonstrations, historical data, or simulated environments. The aim is to reduce the need of costly interaction with the environment. For an overview of the approach see Levine et al. [9], and for different datasets see Fu et al. [2].

One of the main issues with offline RL relates to the ability to generalize from one setup to another. Several methods where developed to address this problem, such as CQL [6], IQL [5] and CRR [16].

3 Motivation: Unsupervised vs. Offline Data

The setup we use in this paper combines both unsupervised and offline data. We argue that this is a natural setup to consider, as it deals with related issues that arise in recent work, namely URLB [8], ExORL [18] and DreamerMPC [13]. In this section we discuss these issues based on experiments we performed on URLB, which serve as the motivation for the final setup we propose.

The results shown in Fig. 2, and Fig. 3, demonstrate different ways in which unsupervised exploration can be used. The standard method, as proposed in the original benchmark of URLB, is to pretrain an agent in the unsupervised environment, using various possible exploration methods, and then use the weights as initial values before finetuning them using a reward function for some given task. This is denoted by **Finetune** in the figures, where the performance is measured for each task after 100K environment steps including a reward signal. While finetuning a pretrained agent leads to some acceleration in training time for some of the tasks, it is not always significantly better than initializing the weights randomly and completely discard the output of the pretraining phase (**baseline**).

We argue that this happens because the benefit of the pretrained agent does not necessarily come from the value of its weights, but rather from its behavior in the environment in the initial steps. In other words, not because its weights are already closer to their optimal values, but rather because the pretrained agent can start generating useful exploration trajectories from step one. To test this, we use a different method to exploit unsupervised exploration, where the exploration trajectories in the pretraining phase are stored in a buffer, and then used together with the online data while training the agent on a specific task.

Essentially this means the pretraining exploration is treated as offline data, however because it comes from unsupervised exploration, the corresponding reward signal is not right. To correct this,

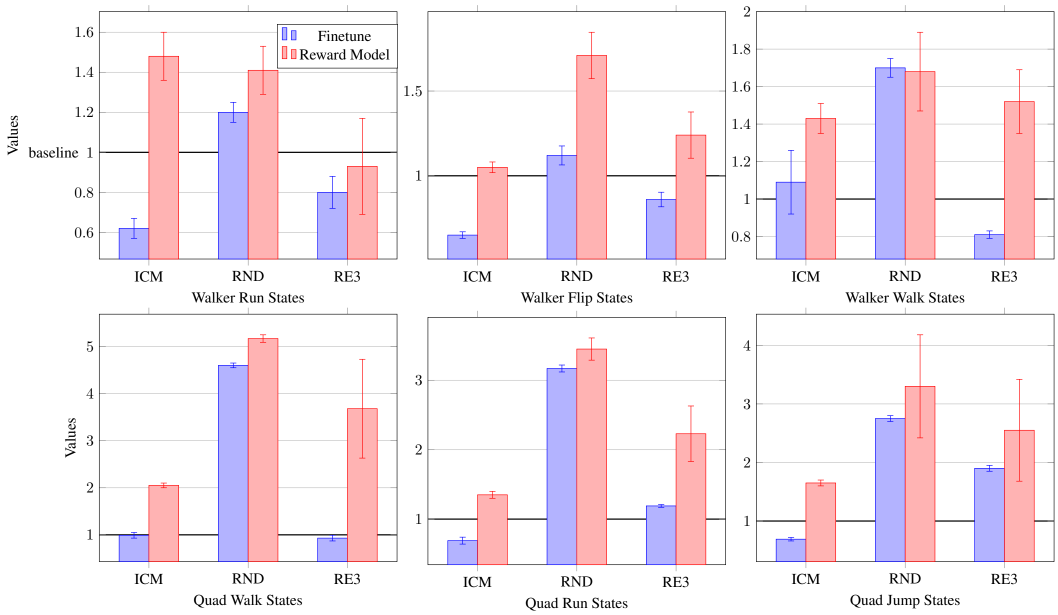

Figure 2: Average episodic returns (over 10 episodes) on the URLB benchmark with **state** observations after 100K training steps. We show two environments and three different tasks for each, and values are normalized to a baseline that does not use any pretraining information. Results show that (1) finetuning a pretrained agent (Finetune) is not always better than training from scratch; and (2) Using the unsupervised pretraining to collect offline data (Reward Model) can lead to better performance. This is implemented by training a reward model to predict the reward of the unsupervised data.

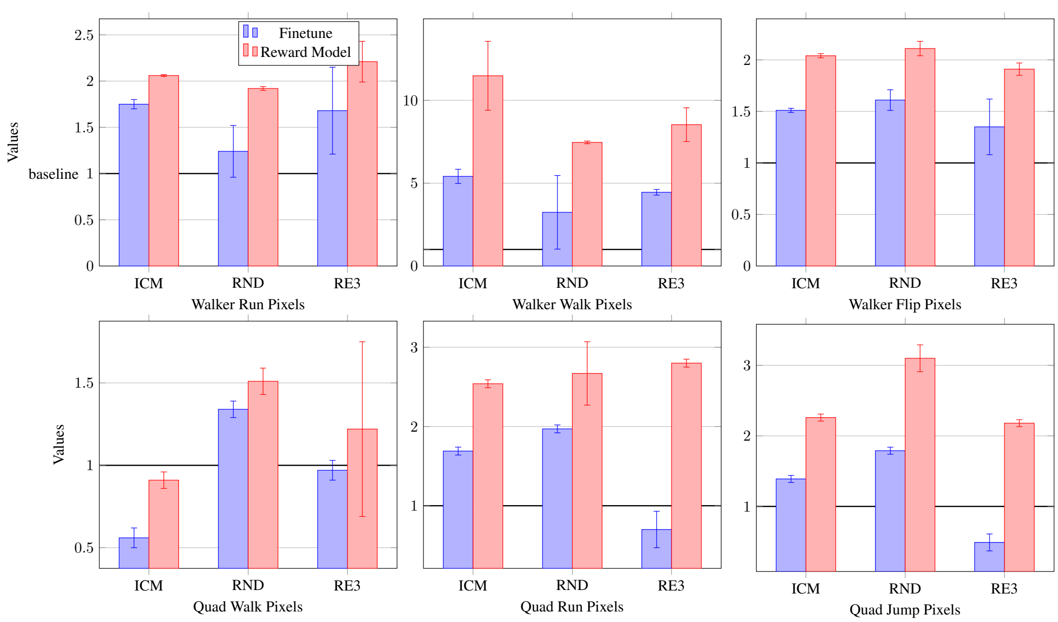

Figure 3: Average episodic returns (over 10 episodes) on the URLB benchmark for **pixel** observations after 100K training steps (normalized to a baseline with no pretraining). The behavior is similar to state observation in Fig. 2, namely (1) finetuning a pretrained agent is not always beneficial (Finetune), and (2) using the unsupervised pretraining as an offline buffer (where the reward signal is predicted by a learned reward model) can lead to better results (Reward Model).

we implement a reward-model component, that uses the online supervised data, to learn a reward predictor. In turn, the reward model can be used to predict the reward in each step of the exploratory trajectories in the pretrained buffer. The results in Fig. 2 and Fig. 3 show this method (**Reward Model**) outperforms simple finetuning of the agent weights. For more details on the implementation of this method see the appendix.

It is interesting to note the relation between the Reward Model method and two recently proposed approaches, namely ExORL [18], and DreamerMPC [13] (see Sec. 2 for their description). The ExORL approach claims that unsupervised exploratory data can serve as better data for offline RL, provided that the ground truth reward function can be used in the offline training phase. Our proposed Reward Model method differs from ExORL in two different ways: (1) it does not rely on access to the true reward function and instead it learns it from the few supervised online steps, and (2) its performance is measured after a few online steps in the environment rather than using the offline data only.

In DreamerMPC, an agent based on planning with a world model is shown to significantly outperform all other methods in URLB. While this can be a result of the general efficiency of planning in RL, we argue that the reasons for the favorable performance of DreamerMPC are related to the results discussed above, because training a world model on unsupervised exploration can be viewed as a smart way to store the exploration trajectories. The world model is essentially a buffer of the exploration trajectories that also allows generalization and planning. We show results with DreamerMPC in later sections.

4 URLB10K: A Benchmark for Fast Adaptation

Table 1: Average episodic return (over 10 episodes) after 10K training steps: we compare learning the reward model from 10K online steps vs. replacing 2K with expert steps (4 episodes). In both cases results are comparable or better than ExORL which uses a GT reward function.

Task/Method	ExORL	RewardModel	RewardModel+Expert
Walker Walk	260	238	**375**
Walker Run	**121**	96	106
Walker Flip	207	236	**285**
Quadruped Walk	250	298	**402**
Quadruped Run	279	299	**412**
Quadruped Jump	324	**640**	630

Motivated by the results and the points discussed above, we propose a new setup to test fast adaptation to novel tasks. The idea is to use different sources of information to achieve even faster adaptation than what is tested in URLB. We base our setup on the following:

1. The agent has access to a very large number of unsupervised interaction with the environment. This is similar to the URLB setup, however this can be used either for pretraining an agent, populating a buffer, or training a world model.
2. The agent has access to a few expert trajectories, containing the task's reward. This data can serve two purposes. First, it can be used as initial information about the task, e.g to train a reward predictor. Second, it can disentangle the problem of exploration in the environment, as using this data is somewhat equivalent to a successful exploration that ensures the important states in the environment were covered.
3. Given the above data, the agent is assessed on its performance after a small amount of supervised online interactions with the environment, measuring its capacity for fast adaptation.

Specifically, we use the URLB with the DeepMind Control Suite environments [15], and we propose to use 2M unsupervised exploration steps and 2K supervised expert demonstration steps (4 episodes), and then measure the performance of the agent after 10K online supervised steps. This is an order of magnitude less steps then what is tested in URLB.

Compared to URLB, the only addition are 4 episodes of supervised expert demonstration (2K steps) that we provide for each task. These episodes are achieved by training the DreamerMPC model on

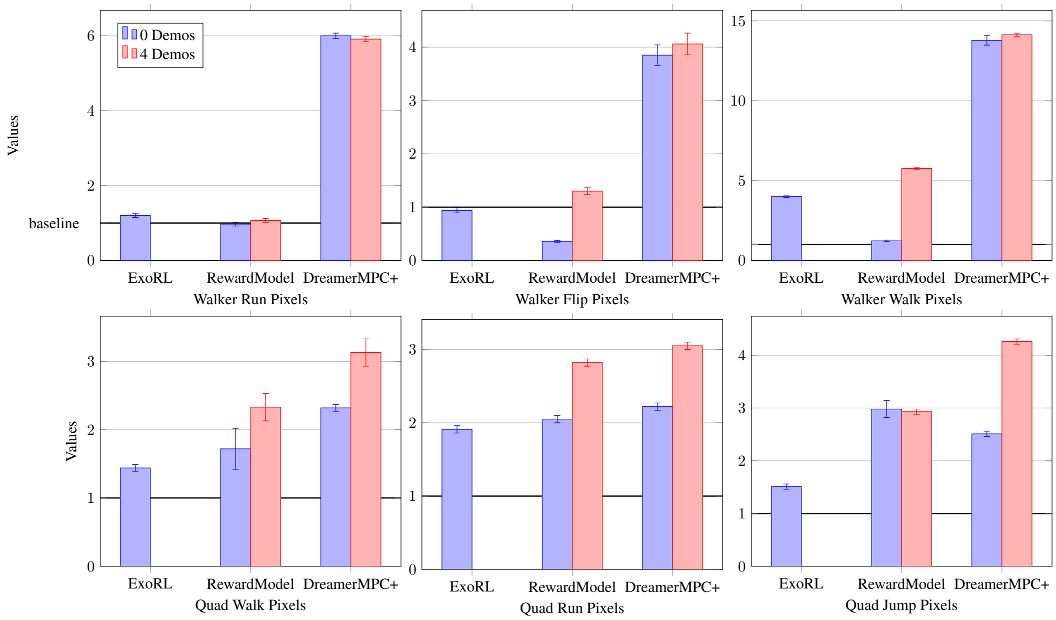

Figure 4: Results for the main methods on URLB10K. Average epidodic return (over 10 episodes). ExORL uses a GT reward function and no online data, RewardModel and DreamerMPC+ are trained on 10K online steps with/without 2K expert demonstrations, and all results are normalized relative to a baseline that was trained on 100K online steps. The results show: (1) Fast adaptation is possible (2) The method based on a world model significantly outperforms other methods. (3) Using expert data is not always beneficial, but can lead to significant improvements.

a supervised environment for 2M steps. We use a random 4 episodes from the last 100 episodes of training.

The capacity of fast adaptation can be seen in the results in table 1, presenting episode rewards after 10K supervised steps for two different environments, and three different tasks in each. For this we implemented a method that uses a reward model to predict the reward of the unsupervised steps. The unsupervised steps are collected using LBS exploration [11] (see Sec. 2), and the method then trains the reward model on data from the online supervised trajectories. We test this method both with and without extra trajectories coming from the offline expert demonstrations, where the first option uses 8K online steps and 2K offline expert steps, and the second uses 10K online steps only.

The method is compared ExORL [18], which relies on the same unsupervised buffer, but instead of using offline expert data and online interactions, it fixes the unsupervised reward signal using privileged knowledge of the ground-truth reward function.

The first thing to note in the results in table 1 is the value of replacing 4 episodes from the online interactions with expert demonstrations, when training the reward model. This generally leads to a significant improvement, although it is not always the case. The second thing to note is the comparable and sometimes favorable results of the learned reward model compared to ExORL. This shows that in this realistic setup, learning the reward from very few episodes is possible, demonstrating the potential for developing methods that exhibit extremely fast adaptation. Results can be viewed also in the bar-plots in figure 4. Details of the implementation of the method can be found in the appendix.

5 Using a World Model

Following the results of training a reward predictor for URLB10K, a natural next step is to consider a full world model. While DreamerMPC [13] showed excellent results on the original URLB, we set to test it on the fast adaptation setup we propose. In addition, we make modifications to the training method, based on the approach that the model should treat unsupervised exploration as offline data rather than pretraining data.

The modifications we make are: (1) Utilize the exploration buffer directly, sampling batches similarly to the method of Reward Model; (2) make offline updates after the online steps collection is finished; (3) increase the environment steps to network update ratio from 10 to 5; and (4) add a few finetuning updates to the world model on the online episodes and the optional expert episodes. We denote this modified model as **DreamerMPC+**, and use **DreamerMPC+Expert** when the method is also using expert demonstrations for training. Additional information on the method implementation can be found in the appendix.

Table 2 shows the results after 10K supervised online steps, or 8K online and 2K offline expert steps (marked as **Expert**). Comparing the modified method (**DreamerMPC+**) to the vanilla method (**DreamerMPC**), shows a significant improvement in all tested tasks. Moreover, replacing 4 episodes with expert demonstrations leads to a further increase in performance for almost all tasks. These results show that our training method leads a new state of the art on the URLB with or without expert data, outperforming all previous methods to date.

A summary of results on the URLB10K benchmark are presented in table 3 and figure 4, comparing our methods based on a world model and reward model that use 10K online steps, to ExORL that uses privileged information of the reward function, and to the baseline of training on 100K online steps. These results highlight the potential for fast adaptation methods, and demonstrate the superiority of world models in merging data of various types into an adaptable policy.

Table 2: Fast adaptation in DreamerMPC. Average episodic return (over 10 episodes) after 10K training steps. Our updates to the DreamerMPC model (DreamerMPC+) lead to significant improvements in fast adaptation, and can be even further improved by incorporating 4 expert demonstration episodes (DreamerMPC+Expert).

Task/Method	DreamerMPC	DreamerMPC+	DreamerMPC+Expert
Walker Walk	630	896	**918**
Walker Run	328	**594**	585
Walker Flip	607	841	**889**
Quadruped Walk	339	401	**540**
Quadruped Run	200	324	**446**
Quadruped Jump	449	539	**915**
Normalized Score	1.0	1.4	**1.7**

Table 3: Results for the main methods on URLB10K. Average episodic returns after 10K training steps. DreamerMPC+Expert effectively combines unsupervised data and expert demonstration and significantly outperforms other methods.

Task/Method	Pix ExORL (GT reward)	RM+Expert	DreamerMPC+Expert
Walker Walk	260	375	**918**
Walker Run	121	106	**585**
Walker Flip	207	285	**889**
Quadruped Walk	250	402	**540**
Quadruped Run	279	412	**446**
Quadruped Jump	324	630	**915**

6 Discussion

In this paper we proposed to test fast adaptation to novel tasks, using only 10K online interactions with the environment. In order to achieve this, we argue that a natural setup is when the agent has access to a large amount of unsupervised experience, and a small amount of supervised expert episodes, demonstrating the desired task.

Towards this goal we presented an extension to the URLB benchmark, and studied methods that treat the pretraining data as offline data, specifically using world models. We show this approach can be related to previous work, and develop training methods that demonstrate the capacity of fast adaptation in such models.

We believe that the proposed setup can serve as an initial benchmark to study and develop methods that can efficiently process various types of available data, and improve fast adaptation in RL.

References

[1] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by Random Network Distillation, October 2018.

[2] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021.

[3] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering Atari with Discrete World Models, February 2022.

[4] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal Difference Learning for Model Predictive Control, July 2022.

[5] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[6] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191, 2020.

[7] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement Learning with Augmented Data, November 2020.

[8] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. URLB: Unsupervised Reinforcement Learning Benchmark, October 2021.

[9] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[10] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, July 2019.

[11] Pietro Mazzaglia, Ozan Catal, Tim Verbelen, and Bart Dhoedt. Curiosity-Driven Exploration via Latent Bayesian Surprise, February 2022.

[12] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction, May 2017.

[13] Sai Rajeswar, Pietro Mazzaglia, Tim Verbelen, Alexandre Piché, Bart Dhoedt, Aaron Courville, and Alexandre Lacoste. Mastering the Unsupervised Reinforcement Learning Benchmark from Pixels, May 2023.

[14] Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. State Entropy Maximization with Random Encoders for Efficient Exploration, June 2021.

[15] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

[16] Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33:7768–7778, 2020.

[17] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning, July 2021.

[18] Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't Change the Algorithm, Change the Data: Exploratory Data for Offline Reinforcement Learning, April 2022.

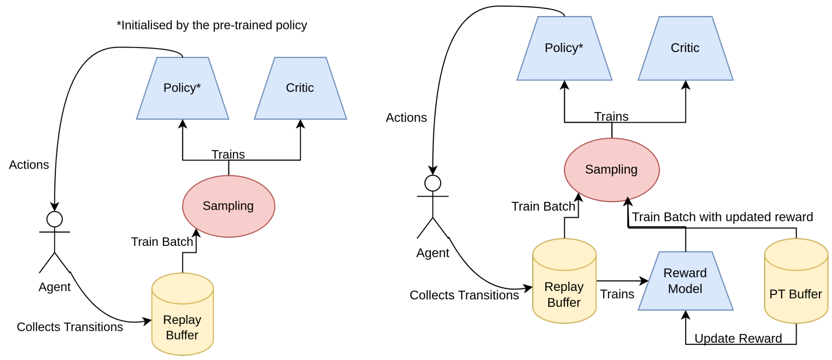

Figure 5: **Left:** Baseline off-policy RL, where we sample only from the replay buffer. **Right:** PT Buffer & Reward Model, where we additionally sample from the PT buffer and predict the task oriented reward using the reward model.

A Additional Details of Methods

A.1 Reward Model

When using the reward model approach, we base the model on the Q network architecture, but instead of training it on Q values, we train it to output $R(s, a)$. After collecting the seed frames, we start with 200 batch updates to train the model using the optional expert episodes and the seed frames.

Next, when starting the online training stage, we intermittently sample online data with the true reward, and train the world model, or when we sample from the unsupervised buffer we use the reward model to get the estimated task reward.

Training an agent for 10K online steps, including sampling from the exploration buffer takes about 2 hours and about 10 hours for additional 400K offline updates on a single A100 GPU.

A.2 DreamerMPC+

Here we do the same sampling and seed frames pre-finetuning as done in the reward model. The difference here is that instead of just learning $R(s, a)$, we learn a world model. The world model is based on Dreamer and is learned in a supervised fashion from the exploration buffer. During the task training stage, the world model is finetuned to correct its reward head, as done in [13]. One difference, is that as we include expert data, we do additional 10 pre-finetuning updates on the world model just after collecting the seed frames.

Training an agent for 10K online steps, including sampling from the exploration buffer takes about 5 hours and about 20 hours for additional 300K offline updates, on a single A100 GPU.

Algorithm 1 Finetuning With a Reward Model on Unsupervised Data

Require: Unsupervised data U, replay buffer D
Require: Reward model R_θ, pretrained policy π_ϕ, initialized critic Q_ξ
Require: number of seed steps N_s, finetune steps N_{ft}, offline updates steps N_{ol}
Require: reward model pre-train steps N_{pt}
Require: offline batches counter $c = 0$

for $t \leftarrow 1$ to N_s **do**
    $D \leftarrow D \cup (s, a, s', r)$
end for
for $t \leftarrow 1$ to N_{pt} **do**
    Update R_θ using mini-batches from D
end for
for $t \leftarrow 1$ to $N_{ft} + N_{ol}$ **do**
    sample batch $b \sim P_{bs}(U, D + c)$
    if $b \subset U$ **then** ▷ we sampled from the unsupervised data
        $c \leftarrow c + 1$
        $(s, a, s', r) \leftarrow b$
        $r' \leftarrow R_\theta(s)$
        Update π_ϕ and Q_ξ using the mini-batch (s, a, s', r')
        continue
    else if $b \subset D$ **then** ▷ we sampled from the replay buffer
        Update π_ϕ and Q_ξ using b
        Update R_θ using b
        if $t < N_{ft}$ **then**
            $a' \leftarrow \pi_\phi(s')$ ▷ collect next transition from the environment
            $o, r \sim P(.|s', a')$
            $D \leftarrow D \cup (s', a', o, r)$
        end if
    end if
end for

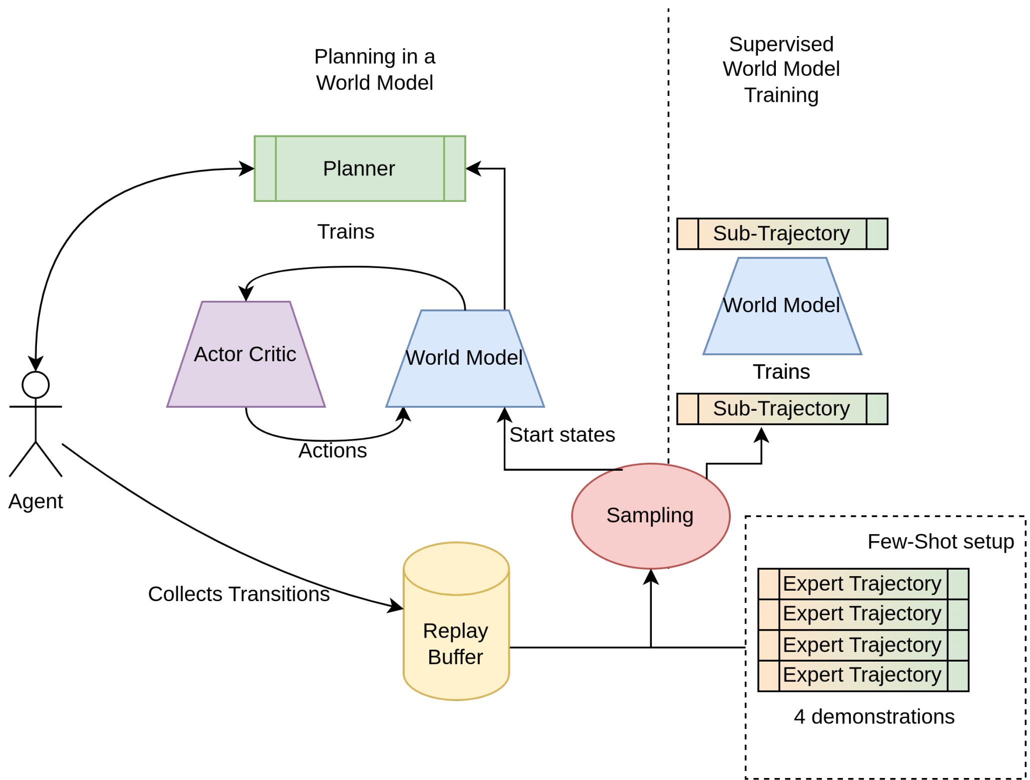

Figure 6: DreamerMPC architecture with additional information from expert demonstration