

# Boosting Robustness in Preference-Based Reinforcement Learning with Dynamic Sparsity

Extended Abstract

Calarina Muslimani  
University of Alberta  
Canada  
musliman@ualberta.ca

Bram Grooten  
Eindhoven University of Technology  
Netherlands  
b.j.grooten@tue.nl

Deepak R.S. Mamillapalli  
University of Alberta  
Canada  
mamillap@ualberta.ca

Mykola Pechenizkiy  
Eindhoven University of Technology  
Netherlands  
m.pechenizkiy@tue.nl

Decebal C. Mocanu  
University of Luxembourg  
Luxembourg  
decebal.mocanu@uni.lu

Matthew E. Taylor  
University of Alberta  
Alberta Machine Intelligence Institute  
Canada  
matthew.e.taylor@ualberta.ca

## ABSTRACT

To integrate into human-centered environments, autonomous agents must learn from and adapt to humans in their native settings. Preference-based reinforcement learning (PbRL) can enable this by learning reward functions from human preferences. However, humans live in a world full of diverse information, most of which is irrelevant to completing any particular task. It then becomes essential that agents learn to focus on the subset of task-relevant state features. To that end, this work proposes R2N (Robust-to-Noise), the first PbRL algorithm that leverages principles of dynamic sparse training to learn robust reward models that can focus on task-relevant features. In experiments with a simulated teacher, we demonstrate that R2N can adapt the sparse connectivity of its neural networks to focus on task-relevant features, enabling R2N to significantly outperform several sparse training and PbRL algorithms across simulated robotic environments. We open-source our code at the following link: <https://github.com/cmuslima/R2N>.

## KEYWORDS

Reinforcement learning, preference learning, sparse training

### ACM Reference Format:

Calarina Muslimani, Bram Grooten, Deepak R.S. Mamillapalli, Mykola Pechenizkiy, Decebal C. Mocanu, and Matthew E. Taylor. 2025. Boosting Robustness in Preference-Based Reinforcement Learning with Dynamic Sparsity: Extended Abstract. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025)*, Detroit, Michigan, USA, May 19 – 23, 2025, IFAAMAS, 35 pages.

## 1 INTRODUCTION

Recent advances in reinforcement learning (RL) are bringing us closer to a future where RL agents aid humans in their daily lives [10, 13, 30]. Preference-based RL (PbRL) is a promising paradigm that allows RL agents to leverage human preferences to adapt their

behavior to better align with human intentions [7, 23]. However, to effectively integrate agents into human-centered environments, autonomous agents should be able to learn from humans in their natural settings. Unfortunately, human environments are inherently noisy. For example, a household robot should be able to effectively learn from human preferences despite receiving a continuous stream of information regarding various household chores. Suppose a household robot is tasked with learning to clean a toy room, and a human provides the robot with preferences on how the room should be cleaned. In this scenario, the robot might receive distracting information such as sounds from children playing, colors and shapes of various toys, room temperature, etc. Only a certain subset of the robot’s perceptions is relevant to the task. Identifying this subset of task-relevant features can boost performance and sample efficiency. However, there has been little attention in PbRL on how to learn in such noisy environments. More often, research in PbRL focuses on hand-engineered environments that contain only task-relevant features [19, 23, 24, 28, 31]. There has been little attention in PbRL on how to learn in noisy environments where identifying all relevant features can be difficult and time-consuming. This lack of attention is problematic as we consider more real-world settings, like the household robot example. In such settings, recent algorithmic improvements for PbRL may not be helpful in environments containing irrelevant features. This paper asks the question:

How can autonomous agents learn from human preferences in *noisy environments* with many irrelevant features?

To study this, we consider the Extremely Noisy Environment (ENE) problem setting [16]. Given a traditional RL environment, an ENE expands the size of the state space by adding features of random noise sampled from  $\mathcal{N}(0, 1)$ . These new features are task-irrelevant and the agent does not have any information on the utility of each feature. Learning becomes quite difficult without knowing which features are task-relevant and which are noise. While previous work in dynamic sparse training (DST) has shown promising results in filtering out irrelevant features in reinforcement learning [11, 16, 27, 33], its application to preference-based reinforcement learning remains unexplored. To that end, we present R2N, a *robust-to-noise* PbRL algorithm that leverages principles of dynamic sparse training to learn a robust reward model, effectively filtering out



This work is licensed under a Creative Commons Attribution International 4.0 License.

irrelevant features while reinforcing neural network connections to pertinent ones. Importantly, this filtering and policy learning is done solely from preferences and not from a ground truth reward function or other background knowledge.

This paper’s core contributions are as follows:

- We are the first to consider the Extremely Noisy Environment setting for preference-based reinforcement learning.
- We propose *R2N*, a noise-robust PbRL algorithm that enables its learned reward model and RL agent to focus on relevant environmental features through dynamic sparse training.
- We demonstrate that *R2N* can maintain competitive performance, but often outperform four sparse training algorithms in five DeepMind Control environments [34] with high added noise.
- We integrate *R2N* with three state-of-the-art PbRL algorithms, leading to significant performance improvements. This demonstrates the versatility of *R2N* and its ability to enhance multiple PbRL methods.

The primary goal of this paper is to highlight the importance of continued research in PbRL for noisy environments, as it is necessary for PbRL to be effective in real-world settings. We demonstrate the potential of dynamic sparse training as a promising avenue towards achieving this goal.

## 2 RELATED WORK

*Human-in-the-Loop RL.* Human-in-the-loop RL consists of a growing set of methods that allow an RL agent to leverage human feedback to improve its behavior. Various types of human feedback have been considered, including demonstrations, action advice, scalar feedback, and preferences. By learning from demonstrations, teachers provide  $\langle \text{state}, \text{action} \rangle$  trajectories of the desired agent behavior [1, 2, 6]. Similarly, in the action advising setting, a teacher suggests actions for the RL agent to take [12, 35]. As an alternative, other approaches consider scalar and preference-based feedback. In learning from scalar feedback, teachers provide a scalar rating of an agent’s behavior [15, 21, 25, 26, 37, 38]. In PbRL, human feedback is even simpler, only requiring a preference between two recorded segments of the agent’s behavior. The preferences are often used to learn an estimate of the true reward function, which the RL agent then maximizes [7, 23]. To reduce the number of human preferences necessary, works have considered a variety of techniques, including data augmentation [31], uncertainty-based exploration [24], meta-learning [19], semi-supervised learning [31], and pre-training with sub-optimal data [28]. Other work has focused on leveraging preferences without explicitly modeling a reward function [18, 32]. PbRL has been popularized in recent years, particularly due to its success in improving large language models [30, 41].

*Sparsity in Neural Networks.* Sparsity provides a means to reduce the parameter count of neural networks without decreasing performance or representational power [3, 4, 9, 11, 27, 39, 40]. Dynamic sparse training is a subfield of sparse training, with methods such as SET [27] and RigL [11], that start from a random sparse neural network and improve its topology over time. In both algorithms, the neural network is randomly pruned at initialization up to a certain sparsity level  $s \in (0, 1)$ . During training, the network is periodically updated by pruning and growing connections in each

sparse layer. More specifically, both SET and RigL drop a fraction of connections with the lowest weight magnitude. However, to grow new weights, SET selects random locations, whereas RigL grows new weights in locations where the gradient magnitude is the highest. This means RigL needs to compute the gradient of all weights (including the inactive ones) during a sparse topology update. RigL has been shown to outperform SET in supervised learning tasks [11], but in RL, there seems to be no significant difference [14]. The fact that DST methods are dynamic (i.e., able to update the network’s topology over time) is crucial. In reinforcement learning Sokar et al. [33] applied DST, successfully improving performance using only ~50% of the weights. In noisy RL environments, Grooten et al. [16] further showed that specifically sparsifying the input layer improves the robustness of RL algorithms. Graesser et al. [14] also provided a large overview of the state of sparse training in a diverse set of RL environments, algorithms, and sparsity levels.

Other forms of dynamic sparse training can include DropConnect [36] and L1 regularization [29]. DropConnect introduces dynamic sparsity into a model by randomly setting a subset of weights to zero during each forward pass. In contrast, L1 regularization adds a term to the model’s loss function that is proportional to the sum of the absolute value of the model’s weights. This induces sparsity by driving the weights of less important features to zero, effectively performing feature selection. Unlike DST approaches, *static* sparse training prunes a set of weights at initialization to a fixed sparsity level, and this sparsity pattern remains fixed throughout training. However, static sparse training has generally been found to be less effective [14, 27].

## 3 BACKGROUND

In reinforcement learning, an RL agent interacts with an environment to maximize the expected cumulative (discounted) reward. This interaction process is modeled as a Markov Decision Process (MDP) consisting of  $\langle \mathcal{S}, \mathcal{A}, T, r, \gamma \rangle$ . At each interaction step  $t$ , the agent receives a state  $s_t \in \mathcal{S}$  and takes an action  $a_t \in \mathcal{A}$  according to its policy  $\pi(s|a)$ . The environment then provides a reward  $r_{t+1} = r(s_t, a_t)$  and transitions to the next state  $s_{t+1}$  according to the transitions dynamics  $T(s_{t+1}|s_t, a_t)$ . The agent attempts to learn a policy that maximizes the discounted return  $G = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ .

This work assumes an MDP\|R setting, where access to the environmental reward function is not provided. The goal is to learn a good policy while simultaneously learning a proper estimate of the reward function from human preferences.

*Preference-based Reinforcement Learning.* PbRL considers trajectory segments  $\sigma$ , where each segment consists of a sequence of states and actions  $\{s_t, a_t, s_{t+1}, a_{t+1}, \dots, s_{t+k}, a_{t+k}\}$ , where  $k$  is the trajectory segment length. Two segments,  $\sigma^0$  and  $\sigma^1$ , are then compared by a teacher. If the teacher prefers segment  $\sigma^1$  over segment  $\sigma^0$ , then the target label  $y = 1$ , and if the converse is true,  $y = 0$ . If both segments are equally preferred, then  $y = 0.5$ . As feedback is provided, it is stored as tuples  $(\sigma^0, \sigma^1, y)$  in a dataset  $D$ . Then, we follow the Bradley-Terry model [5] to define a preference predictor  $P_\theta$  using the reward function estimator  $\hat{r}_\theta$ :

$$P_\theta(\sigma^1 > \sigma^0) = \frac{\exp(\sum_t \hat{r}_\theta(s_t^1, a_t^1))}{\sum_{i \in \{0,1\}} \exp(\sum_t \hat{r}_\theta(s_t^i, a_t^i))} \quad (1)$$

Intuitively, if segment  $\sigma^i$  is preferred over segment  $\sigma^j$  in Equation (1), then the cumulative predicted reward for  $\sigma^i$  should be greater than for  $\sigma^j$ . To train the reward function, we can use supervised learning where the teacher provides the labels  $y$ . We can then update  $\hat{r}_\theta$  through  $P_\theta$  by minimizing the standard binary cross-entropy objective.

$$L^{CE}(\theta, D) = -E_{(\sigma^0, \sigma^1, y) \sim D} \left[ (1 - y) \log P_\theta(\sigma^0 > \sigma^1) + y \log P_\theta(\sigma^1 > \sigma^0) \right] \quad (2)$$

In Equation (2), the loss increases as the predicted probability that  $\sigma^1 > \sigma^0$  diverges from the true label (e.g.,  $y = 1$ ). This loss drives the reward model to update its weights to output a greater predicted total reward for  $\sigma^1$  than for  $\sigma^0$ . The learned reward function,  $\hat{r}_\theta$ , is then used in place of the environmental reward function in the typical reinforcement learning interaction loop.

**Noise in Human-in-the-loop RL.** RL agents can experience different types of noise while interacting and learning from humans: measurement error and distracting features. Measurement error results from uncertainty in perception. For example, if a human is providing feedback to an RL agent, they might be unsure or indecisive about what feedback to provide. In the preference learning literature, some work has studied this by incorporating “imperfect” simulated teachers that provide random preference orderings to some percentage of their total preference queries [22]. However, this type of noise is outside the scope of this work.

**Problem Setting.** We study the setting in which noise is classified as a distracting feature in an environment. For example, consider the household cleaning robot that receives preferences on cleaning styles. In this task, the robot can receive excess information about the house that is not necessary for the cleaning task. Therefore, the robot needs to learn to filter through the noise to focus solely on task-relevant features. We consider the Extremely Noisy Environment (ENE) [16] to study this setting. In an ENE, the state space of a regular RL environment is increased by concatenating a large number of irrelevant features. More specifically, an ENE enlarges the state space such that a certain fraction  $n_f \in [0, 1]$  of the total state space is random noise. These irrelevant features are produced by sampling i.i.d. from  $\mathcal{N}(0, 1)$ . We use this setting in our experiments unless stated otherwise. The PbRL algorithms must identify the most relevant features to (1) learn a robust reward function and (2) learn an adequate policy.

## 4 ROBUST-TO-NOISE PREFERENCE LEARNING

In this section, we introduce *R2N*, a preference-based RL algorithm specifically designed to handle noisy environments (see Figure 1). The goal of R2N is to learn useful reward functions from feedback in environments that contain large numbers of task-irrelevant features (e.g., noise features). To do so, R2N applies dynamic sparse training techniques to PbRL algorithms for the learned reward model to focus on relevant environment features. Algorithm 1 outlines how DST is applied to the reward learning module in a PbRL algorithm, with novel components highlighted in blue. R2N consists of two primary steps. First, at initialization, R2N randomly prunes

the input layer of the reward model to a pre-defined sparsity level  $s^R$  (see line 2 in Algorithm 1). This is an important step, as prior works have shown that sparse neural networks can outperform their dense counterparts in both the supervised learning and RL settings [11, 27, 33]. Second, after every  $\Delta T^R$  weight updates in the training loop, we prune the weakest active connections in the reward model’s input layer (see line 7 in Algorithm 1). The strength of a connection is defined by the absolute value of its weight. After dropping a certain fraction  $d_f^R \in (0, 1)$  of the overall active connections, R2N grows the same number of connections in new locations (see line 8 in Algorithm 1). Growing new connections ensures we maintain the same sparsity level throughout training. To choose which inactive connections to grow, we use the RigL algorithm [11], which activates connections with the highest gradient magnitude. As training the reward model is done via supervised learning, RigL is a more suitable DST candidate due to its demonstrated advantage over SET in supervised learning tasks [11]. This step enables the reward model to concentrate on the most pertinent features during training (see Figure 4c and Appendix B.1). We also repeat this procedure for the input layers of the actor and critic networks of the RL agent, as done in Grooten et al. [16].

A key distinction between R2N and previous work on noisy settings is that R2N is specifically designed for PbRL. R2N is the first to apply DST to the reward learning module. Prior work focuses solely on the deep RL setting, only applying DST to the RL agent’s networks [16]. This difference in setting further resulted in variations in the underlying DST algorithms. R2N purposely uses RigL as the reward models are trained via supervised learning, whereas Grooten et al. [16] is based on the SET algorithm. Figure 2 provides support for this design choice by demonstrating the advantages of using RigL over SET for preference learning. We further illustrate the importance of R2N leveraging DST in both reward learning and RL modules for PbRL algorithms in Figure 4d and Appendix B.6. We observe that applying DST to only the reward module or only the RL module leads to significantly lower performance compared to R2N, which applies DST in both.

---

### Algorithm 1 R2N

---

**Require:** Reward model sparsity level  $s^R$ , topology update period  $\Delta T^R$ , drop fraction  $d_f^R$

**Require:** Set of collected preference data  $D$

- 1: Initialize the reward model  $\hat{r}_\theta$
  - 2: Randomly prune the input layer of the reward model to sparsity level  $s^R$
  - 3: **for** gradient step  $t = 1 \dots T$  **do** ▷ Reward learning
  - 4:   Sample minibatch  $B\{(\sigma^0, \sigma^1, y)^i\}_{i=1}^b \sim D$
  - 5:   Update  $\hat{r}_\theta$  with  $L^{CE}, B$  (equ. 2)
  - 6:   **if**  $t \bmod \Delta T^R == 0$  **then** ▷ Update reward model topology
  - 7:     Prune a fraction  $d_f^R$  of the smallest magnitude reward model weights  $\theta$
  - 8:     Grow  $d_f^R$  new reward model weights  $\theta$  via RigL
  - 9:   **end if**
  - 10: **end for**
-

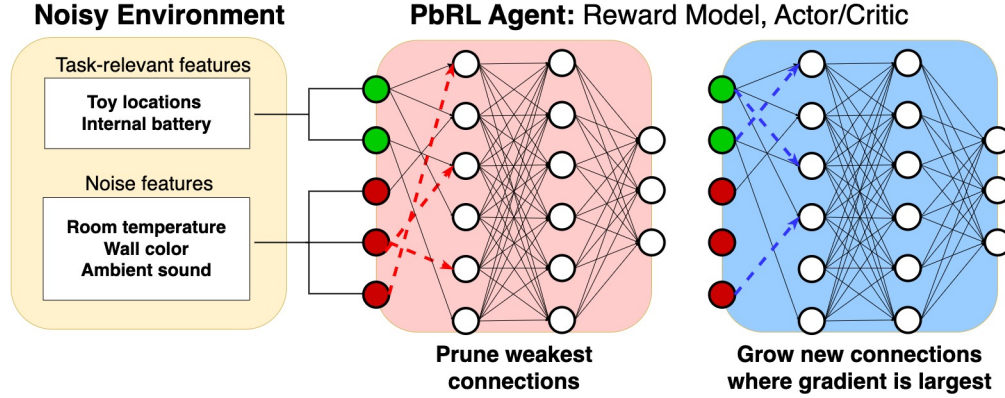


Figure 1: Overview of R2N. Consider the example of a household robot tasked with cleaning a toy room from human preferences. The robot observes many features, although only a few are relevant to the task. R2N learns to connect with the input neurons that provide useful information by *continually* pruning and growing new connections.

## 5 EXPERIMENTS

In this section, we outline the research questions we address in this work and the respective experimental design and empirical results.

### 5.1 Research Questions

We consider the five research questions listed below.

- RQ 1: Can R2N outperform other sparse training baselines in extremely noisy environments?
- RQ 2: Can the addition of R2N boost the performance of a variety of PbRL algorithms?
- RQ 3: How does the amount of noise and feedback affect the performance of R2N?
- RQ 4: What is the effect of applying DST to the reward learning versus RL module in R2N?
- RQ 5: Can R2N learn in environments where the noise features imitate the task-relevant features?

### 5.2 Experimental Design

We evaluate R2N in the DMControl Suite [34], a commonly used PbRL benchmark. More specifically, we consider the following five tasks: Cartpole-swingup, Walker-walk, Quadruped-walk, Cheetah-run, and Humanoid-stand.

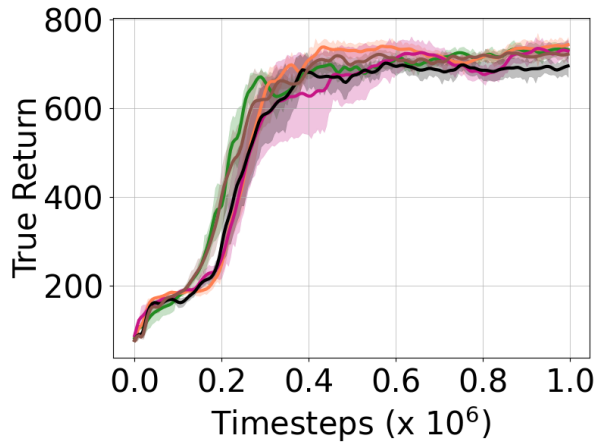
**Baselines.** To evaluate the effectiveness of R2N (RQ 1), we compare it to four sparse training baselines: SET [27], Static Sparse Training, L1 Regularization [29], and DropConnect [36]. Each baseline was integrated with the PbRL algorithm PEBBLE [23]. SET and Static Sparse Training are integrated within both the reward learning and actor/critic modules. L1 Regularization and DropConnect are integrated only within the reward learning module. PEBBLE is a PbRL algorithm that uses unsupervised exploration for policy initialization.

Next, to address RQ 2 and analyze the usefulness of R2N across diverse PbRL algorithms, we integrated it with two additional state-of-the-art PbRL algorithms: SURF and RUNE. These algorithms build upon PEBBLE by applying semi-supervised learning and data augmentation (SURF) and uncertainty-based exploration (RUNE).

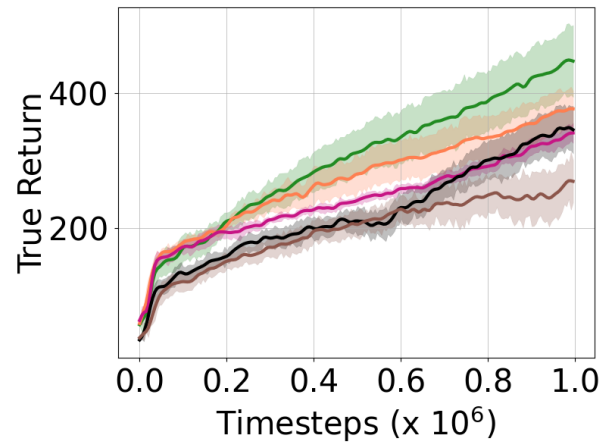
This results in the following baselines: PEBBLE, SURF, and RUNE. We further show the performance of SAC [17] and ANF-SAC [16] in Appendix B.4. However, note that these algorithms serve as oracle baselines as they have access to the ground truth reward during training, unlike the PbRL algorithms.

**Implementation Details.** The primary R2N-specific hyperparameters are the (1) reward model input layer sparsity level  $s^R$ , (2) the reward model topology update period  $\Delta T^R$ , and (3) the reward model drop fraction  $d_f^R$ . After a grid search, we set  $s^R = 80\%$ ,  $\Delta T^R = 100$ , and  $d_f^R = 0.2$ . As we also apply DST to the RL agent, we use the same sparsity level, topology update period, and drop fraction for the actor and critic networks. For any PbRL-specific hyperparameters, we use the default values. As for the RL agent, all methods use the SAC algorithm with the same neural network architecture and associated SAC hyperparameters. See Appendix A for full hyperparameter details for all baselines. For the PbRL baselines, we use a simulated teacher that provides preferences between two trajectory segments according to the ground truth reward function. Although our future work will involve human teachers, simulated teachers have commonly been used for evaluation in prior works [7, 22–24, 28, 31] to reduce the time and expense of human subject studies.

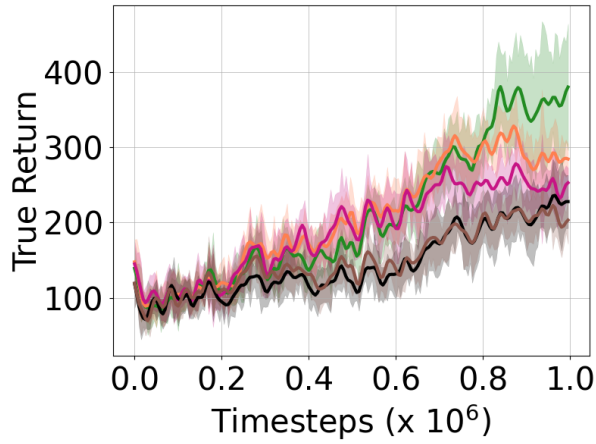
**Training and Evaluation.** We train all algorithms for 1 million timesteps. For evaluation, we show average offline performance (i.e., freeze the policy and act greedily) over ten episodes using the ground truth reward function. We perform this evaluation every 5000 timesteps. Results are averaged over 5 or 14 seeds (Figure 3) with shaded regions indicating the standard error. To test for significant differences in final performance and learning efficiency (e.g., area under the curve: AUC), we perform a one-tailed Welch’s  $t$ -test (equal variances not assumed) with a  $p$ -value significance threshold of 0.05. We use this statistical test, as it was found to be more robust to violations in test assumptions compared to other parametric and non-parametric tests [8]. See Appendix B.8–B.10 for a summary of final performance and AUC across all experiments.



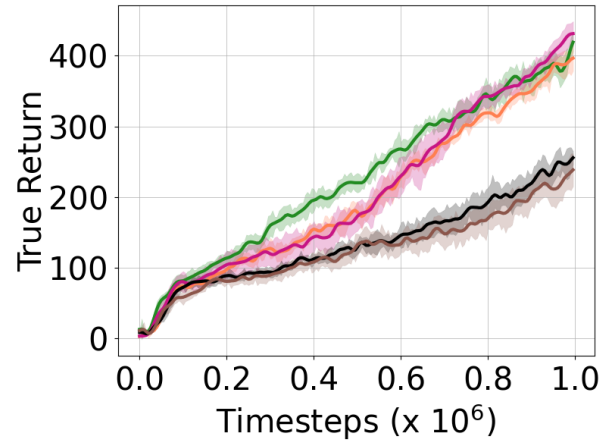
(a) Cartpole-swingup, Noise Fraction = 0.90



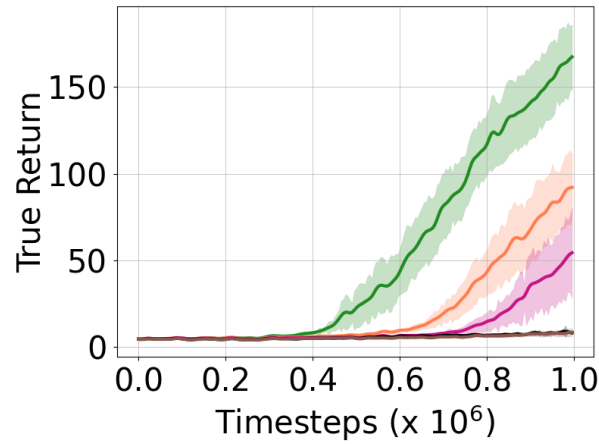
(b) Walker-walk, Noise Fraction = 0.90



(c) Quadruped-walk, Noise Fraction = 0.70



(d) Cheetah-run, Noise Fraction = 0.90

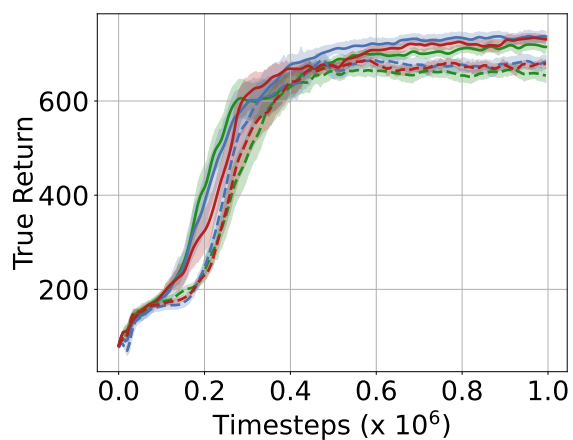


(e) Humanoid-stand, Noise Fraction = 0.70

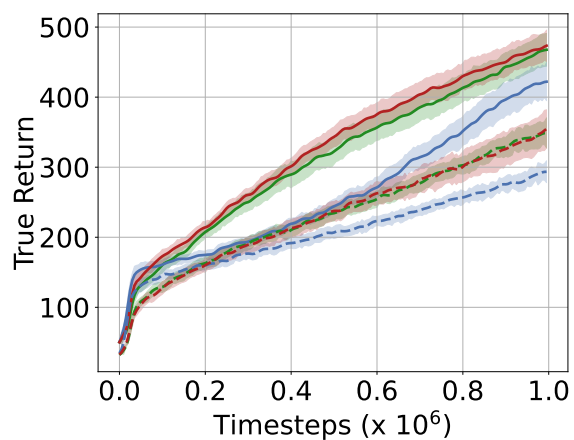
— R2N-PEBBLE    — Static-PEBBLE    — SET-PEBBLE    — L1Reg-PEBBLE    — DropConnect-PEBBLE

Figure 2: These learning curves evaluate the effectiveness of R2N against various dynamic and static sparse training algorithms. R2N (green curves) maintains competitive or greater performance in all environments. Solid lines represent the mean, and shaded regions indicate the standard error across five runs.

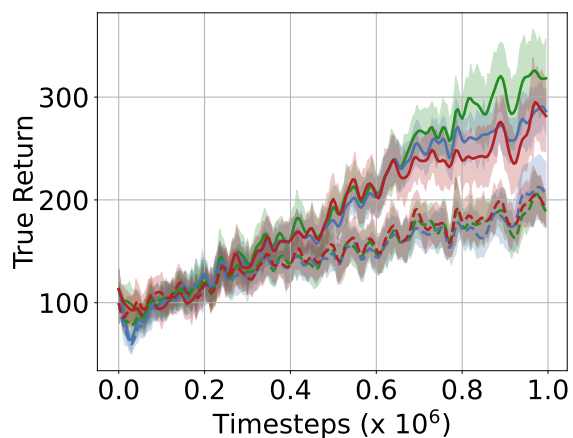




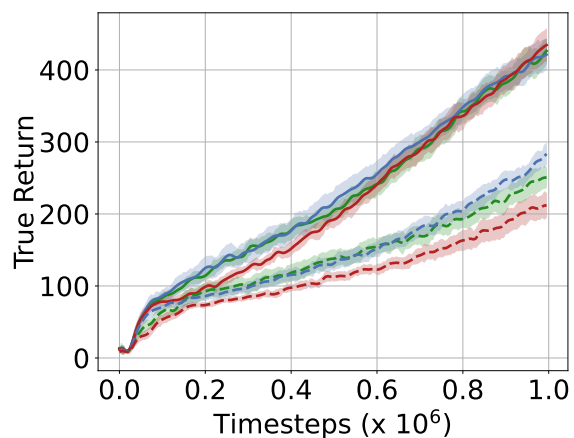
(a) Cartpole-swingup, Noise Fraction = 0.90



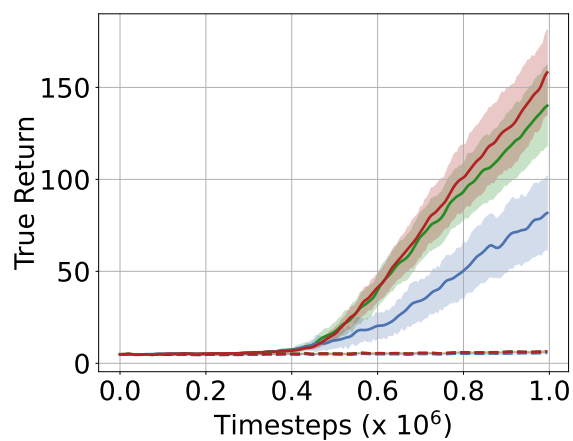
(b) Walker-walk, Noise Fraction = 0.90



(c) Quadruped-walk, Noise Fraction = 0.70



(d) Cheetah-run, Noise Fraction = 0.90



(e) Humanoid-stand, Noise Fraction = 0.70



Figure 3: These learning curves demonstrate that the use of R2N (solid curves) consistently outperforms the tested PbRL algorithms (dotted curves) in terms of final return and learning efficiency. Solid lines and shaded regions represent the mean and standard error across 14 runs.

## 6 DMCONTROL RESULTS

To address RQ 1 and 2, we evaluate R2N in three DMControl environments with noise fraction  $n_f = 0.90$ : Cartpole-swingup, Cheetah-run, and Walker-walk. Recall that this results in a  $10\times$  expansion in the state space size from the original tasks. This significantly increases the size of the state space for the original tasks, from 5 (Cartpole-swingup), 17 (Cheetah-run), and 24 (Walker-walk) to 51, 171, and 241 respectively. As a result, we use larger preference budgets of 400 (Cartpole-swingup), 1000 (Cheetah-run), and 4000 (Walker-walk) to compensate for the increased task difficulty. Next, we evaluate R2N in two DMControl environments with noise fraction  $n_f = 0.70$ : Quadruped-walk and Humanoid-stand. To keep the state space size for all ENE variants comparable, we use a smaller noise fraction for Quadruped-walk and Humanoid-stand. This is necessary as the original state spaces of these environments are three times larger than those of the DMControl environments previously tested. This results in the state space size increasing from 68 to 260 and 67 to 224 for the Quadruped-walk and Humanoid-stand environments respectively. In this setting, we used preference budgets of 4000 (Quadruped-walk) and 10000 (Humanoid-stand).

*R2N versus Sparse Training Baselines.* In Figure 2 and Appendix B.8, we evaluate RQ 1, examining the effectiveness of R2N compared to four sparse training baselines. We find that in all five environments, R2N-PEBBLE (green curves) is the only algorithm that *consistently* achieves superior performance. In particular, R2N-PEBBLE significantly outperforms L1-Regularization (black curves) and DropConnect (brown curves) in terms of learning efficiency in four out of five environments ( $p \leq 0.034$ ). While Static-PEBBLE (orange curves) and SET-PEBBLE (pink curves) prove to be more competitive, R2N maintains significant performance improvement both in Humanoid-stand (Static and SET-PEBBLE; AUC and final return,  $p \leq 0.024$ ) and Cheetah-run (Static-PEBBLE; AUC,  $p \leq 0.018$ ).

*Effectiveness of R2N across PbRL Algorithms.* Next, we focus on RQ 2, to understand whether R2N can boost performance across a diverse set of PbRL algorithms. In Figure 3 and Appendix B.9, we find that the addition of R2N (solid lines) significantly improved both the learning efficiency ( $p \leq 0.021$ ) and final return ( $p \leq 0.006$ ) of the base PbRL algorithm (dotted lines) across all 15 tested baseline-environment combinations.<sup>1</sup> Moreover, in Table 7, we find that R2N resulted in a substantial increase in average final return over the base PbRL algorithms, with almost half of the cases achieving a 50% or greater performance boost. We argue that R2N can outperform the current PbRL algorithms because it is designed to disregard irrelevant features, an important trait for extremely noisy environments. In Figure 4c, we find that R2N adapts its reward model to focus more on task-relevant features. This results in a significantly greater number of connections to the task-relevant features (pink curve) as compared with the noise features (orange curve). We observe a similar pattern for the actor and critic networks (see Appendix B.1). Existing PbRL algorithms primarily focus on improving feedback efficiency in conventional RL environments that contain only task-relevant features — these results show that such improvements alone may be insufficient for more noisy DMControl

environments. However, we also find that R2N can achieve comparable performance in noise-free environments (see Figure 18), indicating its broader applicability.

## 7 SENSITIVITY AND ABLATION STUDIES

Research questions 3-5 pertain to the utility of R2N across the following dimensions: (1) the number of noise features, (2) the number of feedback queries, (3) the effect of DST on individual learning modules in R2N, and (4) the type of noise features. We perform this analysis in the Cheetah-run environment. In these studies, we restrict our comparison of R2N-PEBBLE with only PEBBLE, as PEBBLE performed comparably to the other tested PbRL algorithms in Section 6. We include additional results for the Walker-walk environment in Appendix B. These results demonstrate similar performance trends.

*Noise Study.* First, we aim to understand how effective R2N is at various noise fractions, RQ 3. Therefore, we fix the preference budget to 1000 and vary the noise fraction  $n_f \in \{0, 0.2, 0.5, 0.7, 0.9, 0.95\}$ . In Figure 4a and Table 10, we find that for higher noise fractions in Cheetah-run, R2N-PEBBLE maintains significant improvement over PEBBLE in learning efficiency with  $p \leq 0.014$  for  $n_f \geq 50\%$ , and in final return with  $p \leq 0.007$  for  $n_f \geq 70\%$ . Unsurprisingly, given the small feedback budget, both methods perform worse as the noise fraction increases.

*Feedback Study.* We continue our analysis of RQ 3 and evaluate both algorithms under multiple preference budgets. We set the noise fraction  $n_f = 0.90$  and vary the preference budget  $b \in \{100, 200, 400, 1000, 2000, 4000, 10000\}$ . In Figure 4b and Table 11, we find that in Cheetah-run, R2N-PEBBLE significantly outperforms PEBBLE in terms of learning efficiency (AUC) and final return with  $p \leq 0.009$  for all tested feedback budgets.

*DST Component Ablation.* In R2N, we apply DST to both the reward model and actor/critic networks. In RQ 4, our goal is to understand the importance of DST for both learning modules. In this ablation, we consider a noise fraction  $n_f = 0.90$  and a preference budget of 1000. In Figure 4d and Table 12, we find that in Cheetah-run, full R2N (green curve) significantly outperforms the R2N variants that apply DST only to the RL module (purple curve;  $p \leq 0.022$  for AUC and final return) or only to the reward learning module (black curve;  $p \leq 0.002$  for AUC and final return). This demonstrates that it is important for the reward model and RL agent to learn to avoid irrelevant features in R2N-PEBBLE.

*Imitating Real Features Study.* Experiments thus far consider extremely noisy environments where noise features were sampled from  $\mathcal{N}(0, 1)$ . For RQ 5, we increase the difficulty by using noise features that mimic task-relevant features. We find the distribution of each task-relevant feature as follows. First, we train a SAC agent in a noise-free environment for 1 million timesteps. Second, we perform policy rollouts and store the state transitions. Third, with the state transition data, we can create a histogram for each feature. Fourth, we sample from each feature distribution to create a noise feature. Due to increased difficulty, we set the noise fraction  $n_f = 0.90$  but use a higher preference budget of 4000. In Figure 4e and Table 13, we find that in this setting R2N-PEBBLE (dotted green

<sup>1</sup>We perform three separate Welch’s  $t$ -tests comparing the base PbRL algorithm to its R2N variant.

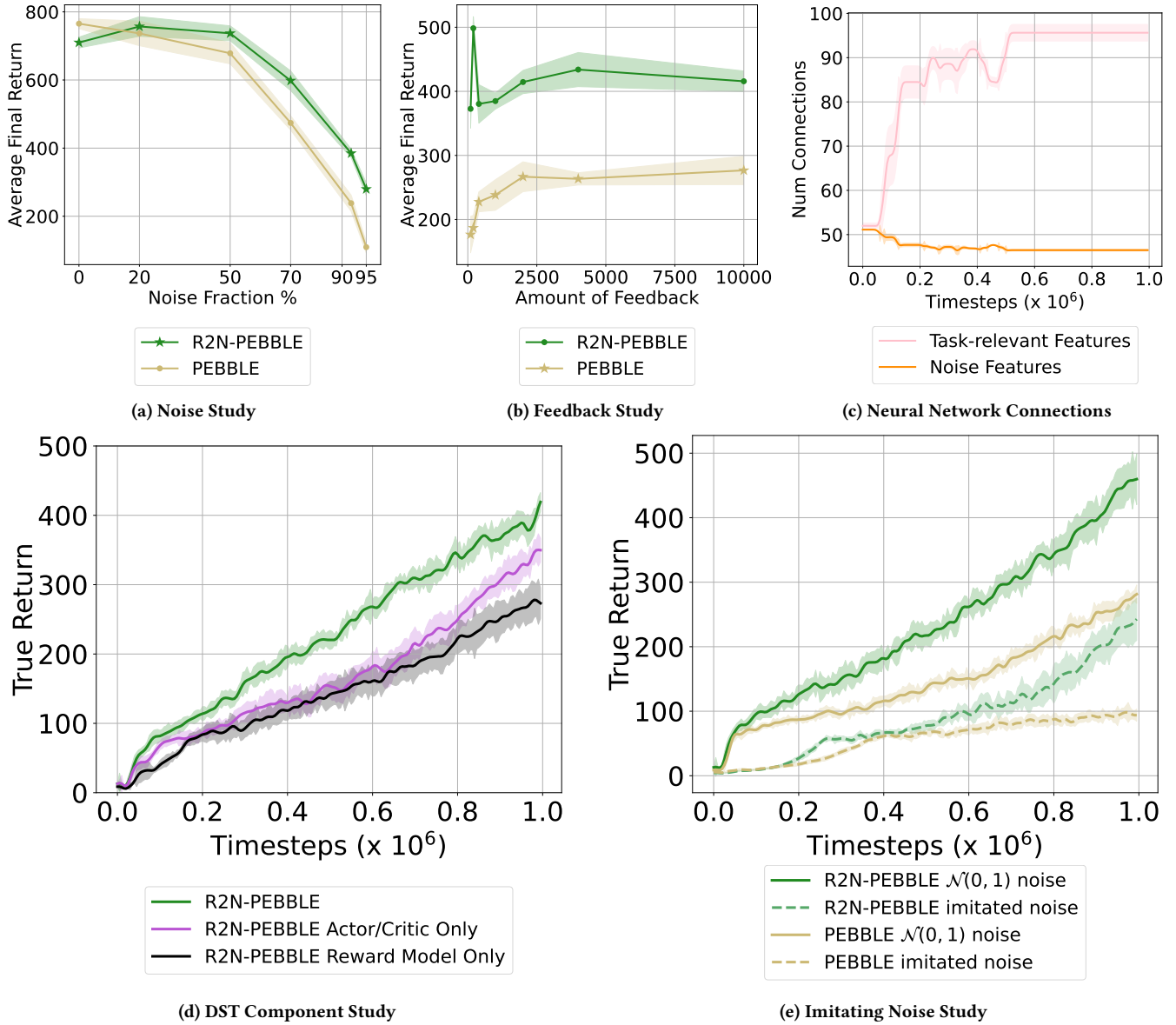


Figure 4: Further studies in Cheetah-run: (a) effect of noise fraction, (b) effect of feedback budget, (c) average number of neural network connections to task-relevant versus noise features in a reward model with R2N, (d) DST component ablation, and (e) effect of noise feature distributions. Solid lines and shaded regions represent the mean and standard error across five runs.

curve) maintains significantly greater performance than PEBBLE (dotted yellow curve) with  $p \leq 0.004$  for both AUC and final return. However, for both R2N-PEBBLE and PEBBLE, we observe performance degradation compared to the setting in which the noise is sampled from the standard normal distribution (solid curves).

## 8 CONCLUSION

For RL agents to become commonplace, agents should be able to learn from people in human-centered environments (e.g., home, school, office). However, humans live in a world full of information,

most of which is not necessary for completing individual tasks. Current state-of-the-state PbRL algorithms do not consider the impact of irrelevant environment features and are, consequently, unable to adequately learn in this setting. To that end, we present R2N, a novel *robust-to-noise* PbRL algorithm that leverages dynamic sparse training techniques to learn robust reward models in *extremely noisy environments*. R2N continually adjusts the network topology of both the reward model and RL agent networks to focus on task-relevant features. This enables R2N to successfully learn in environments where up to 95% of the features are noise.



This work represents the first PbRL algorithm specifically designed to learn in extremely noisy environments. As such, several promising research directions remain to be explored. For example, to provide a proof-of-concept of R2N, we use a simulated teacher to obtain preferences. To further validate our obtained results and confirm their generalizability to human preferences, future work must perform a human-subjects study. In addition, we limit our investigation of R2N to the Extremely Noisy Environment setting, where noise features are added to existing RL environments. To consider more real-world settings, R2N should be studied with real robots that may be receiving pixel input. Lastly, this work assumes that a feature’s relevance is constant (i.e., always useful or always noise). However, an interesting extension would be learning to filter irrelevant features in the continual learning setting, in which features can be relevant to one task and irrelevant to the next.

## REFERENCES

- [1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship Learning via Inverse Reinforcement Learning. In *The 21st International Conference on Machine Learning*. URL: <https://ai.stanford.edu/~ang/papers/icml04-apprentice.pdf>.
- [2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* (2009). URL: <https://doi.org/10.1016/j.robot.2008.10.024>.
- [3] Zahra Atashgahi, Joost Pieterse, Shiwei Liu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. 2022. A Brain-inspired Algorithm for Training Highly Sparse Neural Networks. *Machine Learning* (2022), 1–42. URL: <https://arxiv.org/abs/1903.07138>.
- [4] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. 2018. Deep Rewiring: Training very sparse deep networks. *International Conference on Learning Representations* (2018). URL: <https://arxiv.org/abs/1711.05136>.
- [5] Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika* (1952). URL: <https://doi.org/10.2307/2334029>.
- [6] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. 2015. Reinforcement Learning from Demonstration through Shaping. In *The 24th International Joint Conference on Artificial Intelligence*. URL: <https://www.ijcai.org/Proceedings/15/Papers/472.pdf>.
- [7] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *The 31st Conference on Neural Information Processing Systems*. URL: <https://arxiv.org/abs/1706.03741>.
- [8] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2022. A Hitchhiker’s Guide to Statistical Comparisons of Reinforcement Learning Algorithms. URL: <https://arxiv.org/abs/1904.06979>.
- [9] Tim Dettmers and Luke Zettlemoyer. 2019. Sparse Networks from Scratch: Faster Training without Losing Performance. *arXiv preprint arXiv:1907.04840* (2019). URL: <https://arxiv.org/abs/1907.04840>.
- [10] Yogesh K. Dwivedi, Laurie Hughes, Elvira Ismagilova, Gert Aarts, Crispin Coombs, Tom Crick, Yanqing Duan, Rohita Dwivedi, John Edwards, Aled Eirug, Vasilis Galanos, P. Vigneswara Ilavarasan, Marijn Janssen, Paul Jones, Arpan Kumar Kar, Hatice Kizgin, Bianca Kronemann, Banita Lal, Biagio Lucini, Rony Medaglia, Kenneth Le Meunier-FitzHugh, Leslie Caroline Le Meunier-FitzHugh, Santosh Misra, Emmanuel Mogaji, Sujeet Kumar Sharma, Jang Bahadur Singh, Vishnupriya Raghavan, Ramakrishnan Raman, Nripendra P. Rana, Spyridon Samothrakakis, Jak Spencer, Kuttimani Tamilmani, Annie Tubadji, Paul Walton, and Michael D. Williams. 2021. Artificial Intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy. *International Journal of Information Management* (2021). URL: <https://www.sciencedirect.com/science/article/pii/S026840121930917X>.
- [11] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the Lottery: Making All Tickets Winners. In *The 37th International Conference on Machine Learning*. URL: <https://arxiv.org/abs/1911.11134>.
- [12] Spencer Frazier and Mark Riedl. 2019. Improving deep reinforcement learning in Minecraft with action advice. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. URL: <https://arxiv.org/abs/1908.01007>.
- [13] Google Gemini Team. 2024. Gemini: A Family of Highly Capable Multimodal Models. *arXiv:2312.11805 [cs.CL]* URL: <https://arxiv.org/abs/2312.11805>.
- [14] Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. 2022. The State of Sparse Training in Deep Reinforcement Learning. In *The 39th International Conference on Machine Learning*. URL: <https://arxiv.org/abs/2206.10369>.
- [15] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. 2013. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. In *The 26th Conference on Neural Information Processing Systems*. URL: <https://proceedings.neurips.cc/paper/2013/file/e034fb6b66aacc1d48f445ddfb08da98-Paper.pdf>.
- [16] Bram Grooten, Ghada Sokar, Shibhansh Dohare, Elena Mocanu, Matthew E. Taylor, Mykola Pechenizkiy, and Decebal Constantin Mocanu. 2023. Automatic Noise Filtering with Dynamic Sparse Training in Deep Reinforcement Learning. In *The 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. URL: <https://arxiv.org/abs/2302.06548>.
- [17] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *The 35th International Conference on Machine Learning*. URL: <https://arxiv.org/abs/1801.01290>.
- [18] Joey Hejna and Dorsa Sadigh. 2023. Inverse Preference Learning: Preference-based RL without a Reward Function. In *The 37th Conference on Neural Information Processing Systems*. URL: <https://arxiv.org/abs/2305.15363>.
- [19] Donald Joseph Hejna III and Dorsa Sadigh. 2023. Few-Shot Preference Learning for Human-in-the-Loop RL. In *The 6th Conference on Robot Learning*. URL: <https://proceedings.mlr.press/v205/iii23a.html>.
- [20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *The 3rd International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=8gmWwJfYlj>.
- [21] W Bradley Knox and Peter Stone. 2009. Interactively Shaping Agents via Human Reinforcement. In *The 5th International Conference on Knowledge Capture*. URL: <https://bradknox.net/public/papers/kcap09-knox.pdf>.
- [22] Kimin Lee, Laura Smith, Anca Dragan, and Pieter Abbeel. 2021. B-Pref: Benchmarking Preference-Based Reinforcement Learning. In *Conference on Neural Information Processing Systems: Track on Datasets and Benchmarks*. URL: <https://arxiv.org/abs/2111.03026>.
- [23] Kimin Lee, Laura M Smith, and Pieter Abbeel. 2021. PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training. In *The 38th International Conference on Machine Learning*. URL: <https://arxiv.org/abs/2106.05091>.
- [24] Xinran Liang, Katherine Shu, Kimin Lee, and Pieter Abbeel. 2022. Reward Uncertainty for Exploration in Preference-based Reinforcement Learning. In *The 10th International Conference on Learning Representations*. URL: <https://arxiv.org/abs/2205.12401>.
- [25] Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jeff Huang, and David L Roberts. 2016. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Autonomous Agents and Multi-Agent Systems* (2016). URL: <https://link.springer.com/article/10.1007/s10458-015-9283-7>.
- [26] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, David Roberts, Matthew E Taylor, and Michael L Littman. 2017. Interactive Learning from Policy-Dependent Human Feedback. In *The 34th International Conference on Machine Learning*. URL: <https://arxiv.org/abs/1701.06049>.
- [27] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2018. Scalable Training of Artificial Neural Networks with Adaptive Sparse Connectivity inspired by Network Science. *Nature communications* (2018). URL: <https://arxiv.org/abs/1707.04780>.
- [28] Calarina Muslimani and Matthew E Taylor. 2024. Leveraging Sub-Optimal Data for Human-in-the-Loop Reinforcement Learning (extended abstract). In *The 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.
- [29] Andrew Y. Ng. 2004. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning* (Banff, Alberta, Canada) (ICML '04). Association for Computing Machinery, New York, NY, USA, 78. <https://doi.org/10.1145/1015330.1015435>.
- [30] OpenAI. 2023. GPT-4 Technical Report. URL: <https://arxiv.org/abs/2303.08774>.
- [31] Jongjin Park, Younggyo Seo, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. 2022. SURF: Semi-supervised Reward Learning with Data Augmentation for Feedback-efficient Preference-based Reinforcement Learning. In *The 10th International Conference on Learning Representations*. URL: <https://arxiv.org/abs/2203.10050>.
- [32] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *The 37th Conference on Neural Information Processing Systems*. URL: <https://arxiv.org/abs/2305.18290>.
- [33] Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. 2022. Dynamic Sparse Training for Deep Reinforcement Learning. In *The 31st International Joint Conference on Artificial Intelligence*. URL: <https://arxiv.org/abs/2106.04217>.
- [34] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew LeFrancq, et al. 2018. Deepmind Control Suite. URL: <https://arxiv.org/abs/1801.00690>.
- [35] Lisa Torrey and Matthew Taylor. 2013. Teaching on a budget: Agents advising agents in reinforcement learning. In *The 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. URL: <https://www.ifaamas.org>.

org/Proceedings/aamas2013/docs/p1053.pdf.

- [36] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of Neural Networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 1058–1066. <https://proceedings.mlr.press/v28/wan13.html>
- [37] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. 2018. Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces. In *The 32nd AAAI Conference on Artificial Intelligence*. URL: <https://arxiv.org/abs/1709.10163>.
- [38] Devin White, Mingkan Wu, Ellen Novoseller, Vernon Lawhern, Nick Waytowich, and Yongcan Cao. 2023. Rating-based Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*. URL: <https://arxiv.org/abs/2307.16348>.
- [39] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. 2019. Discovering Neural Wirings. *Advances in Neural Information Processing Systems* 32 (2019). URL: <https://arxiv.org/abs/1906.00586>.
- [40] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. 2021. MEST: Accurate and Fast Memory-Economic Sparse Training Framework on the Edge. *Advances in Neural Information Processing Systems* 34 (2021), 20838–20850. URL: <https://arxiv.org/abs/2110.14032>.
- [41] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2020. Fine-Tuning Language Models from Human Preferences. URL: <https://arxiv.org/abs/1909.08593>.

## APPENDIX

### A IMPLEMENTATION DETAILS

In this section, we present further implementation details on our method R2N, as well as the other baselines compared in our study. Our results in Section 6 show that R2N can boost the performance of multiple preference-based reinforcement learning algorithms, especially in extremely noisy environments. For an overview of the dimensions of the state space in each ENE studied, see Table 1.

The primary R2N-specific hyperparameters are the (1) reward model input layer sparsity level  $s$ , (2) the reward model topology update period  $\Delta T$ , and (3) the reward model drop fraction  $d_f$ . To set these values, we performed a grid search. For the reward model input layer sparsity level  $s$ , we used the default sparsity level as in ANF [16]. Next, for the reward model topology update period  $\Delta T$ , we tested values for  $\Delta T \in \{20, 100, 250, 500, 1000\}$ . Lastly, for the reward model drop fraction  $d_f$ , we tested values for  $d_f \in \{0.05, 0.1, 0.2\}$ .

To train a single R2N model (i.e., one seed), we used 1 GPU, 20-24 hours of run-time and 25-50G of memory.

*Hyperparameters.* An overview of the hyperparameters of the standard RL algorithms (without preference learning) is provided in Table 2. The specific settings of R2N and our PbRL baselines are given in Table 3. The hyperparameters for the sparse training baselines are outlined in Table 4.

### B ADDITIONAL RESULTS

#### B.1 Neural Network Connections to Relevant versus Noise Features

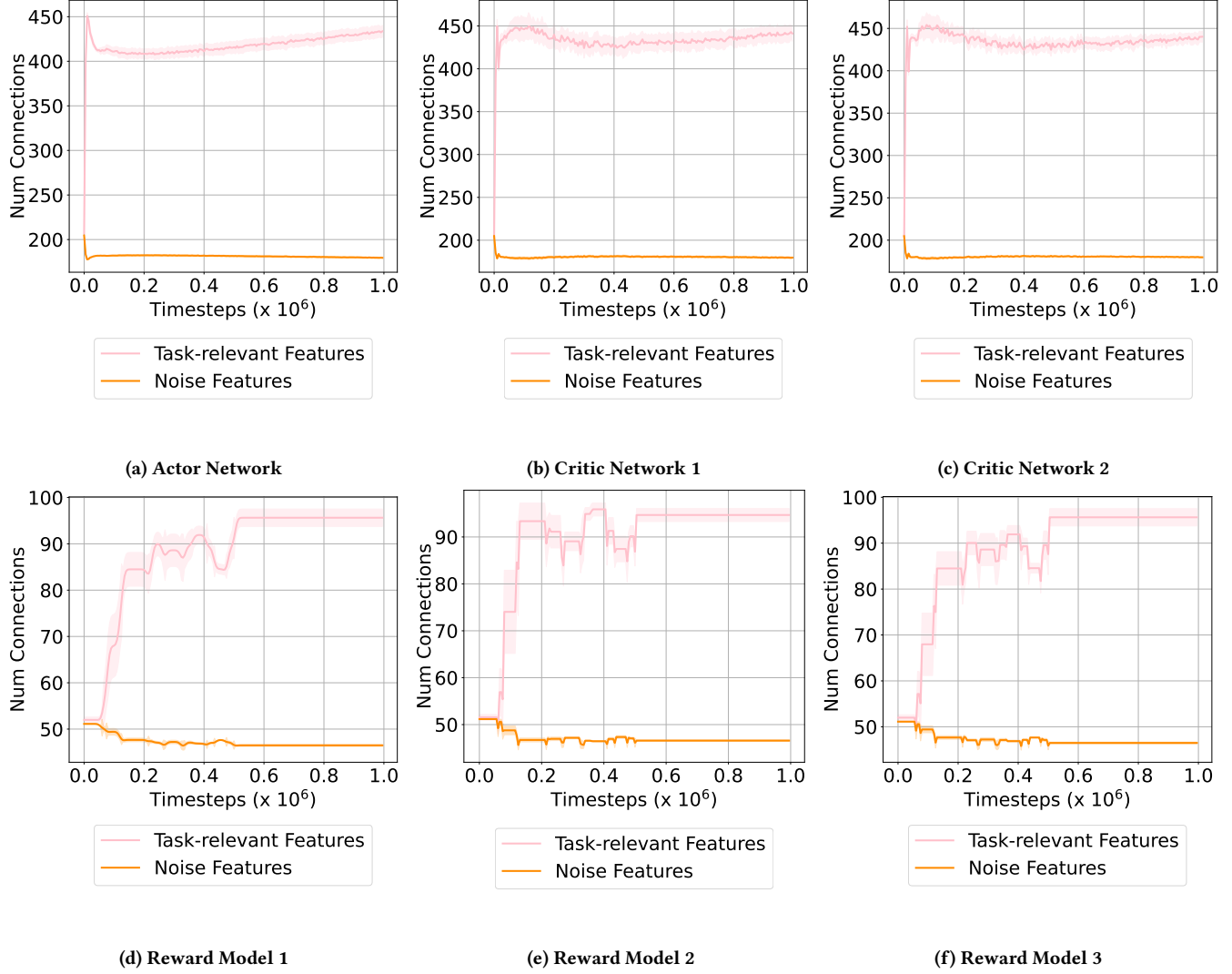
In this section, we show further analysis of the neural network connections in R2N-PEBBLE for the Cheetah-run experiment (90% noise features and a feedback budget of 1000) and Walker-walk experiment (90% noise features and a feedback budget of 4000). More specifically, Figures 5 and 6 highlight that with R2N, the RL agent and the reward models can significantly increase the number of connections to the task-relevant features (pink curves) compared with the number of noise features (orange curves). However, note that the reward models are trained until timestep 500000, so the connectivity does not alter after that point.

#### B.2 Feedback and Noise Ablations in Cheetah-run

In this section, we present additional experimental results ablating over the noise fractions and preference feedback budgets for the Cheetah-run environment. Overall, in Figures 7 through 10, we find that R2N-PEBBLE (green curves) is generally more robust than PEBBLE (yellow curves) in varying noise levels and feedback amounts. Especially in environments with high noise levels, such as 90-95%, shown in Figures 9 and 10, R2N significantly improves the performance of PEBBLE.

**Table 1: This table shows the size of the state space for the original environments and the corresponding extremely noisy environment (ENE) variants.**

Task	Original State Space	Noise	State Space of ENE
Cartpole-swingup	5	0.9	51
Walker-walk	24	0.9	241
Cheetah-run	17	0.9	171
Quadruped-walk	68	0.7	260
Humanoid-stand	67	0.7	224



**Figure 5: Cheetah-run.** These plots show the average number of neural network connections to the relevant versus noise features in R2N’s RL networks and reward models.

Table 2: Hyperparameters for SAC and ANF-SAC, the standard RL algorithms that learn from the environment’s true reward signal, not via preference learning. Note, however, that we present R2N’s hyperparameters for the actor and critic networks here as well.

Hyperparameter	Value
<i>SAC (Shared by all algorithms)</i>	
optimizer	Adam [20]
discount	0.99
actor learning rate	$10^{-4}$
critic learning rate	$10^{-4}$
alpha learning rate	$10^{-4}$
actor betas	0.9, 0.999
critic betas	0.9, 0.999
alpha betas	0.9, 0.999
target smoothing coefficient	0.005
actor update frequency	1
critic target update frequency	2
init temperature	0.1
batch size	1024
replay buffer size	$10^6$
initial collect steps	1000
network type	MLP
number of hidden layers	2
number of neurons per hidden layer	1024
nonlinearity	ReLU
number of training steps	$10^6$
<i>ANF-SAC and R2N (actor &amp; critics)</i>	
sparsity level input layer	0.8
drop fraction $d_f$	0.05
topology-change period $\Delta T$	1000
new weights init value	0
DST method	RigL [11]
sparsify target networks	false

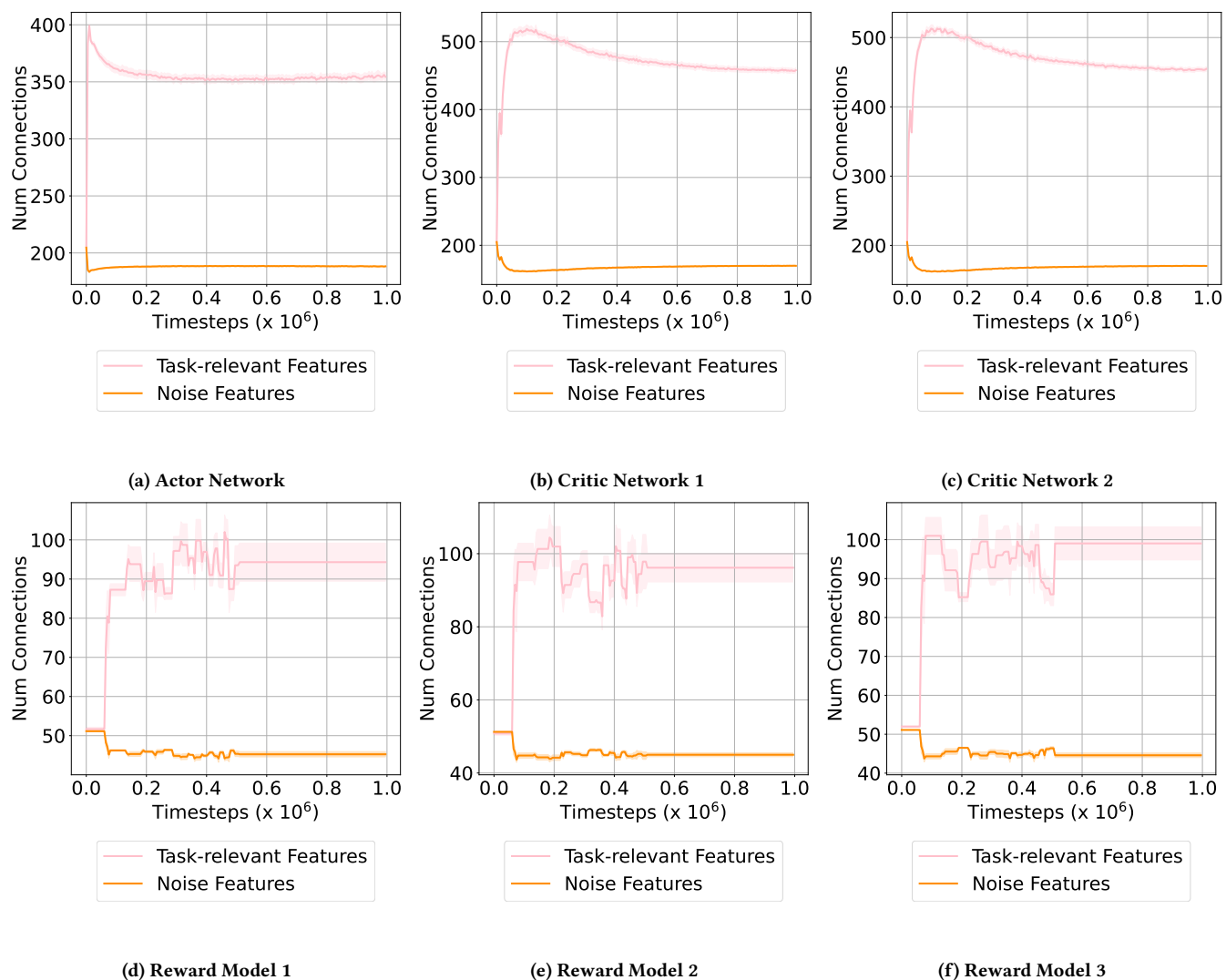


Table 3: Hyperparameters for the PbRL algorithms used, including our method R2N.

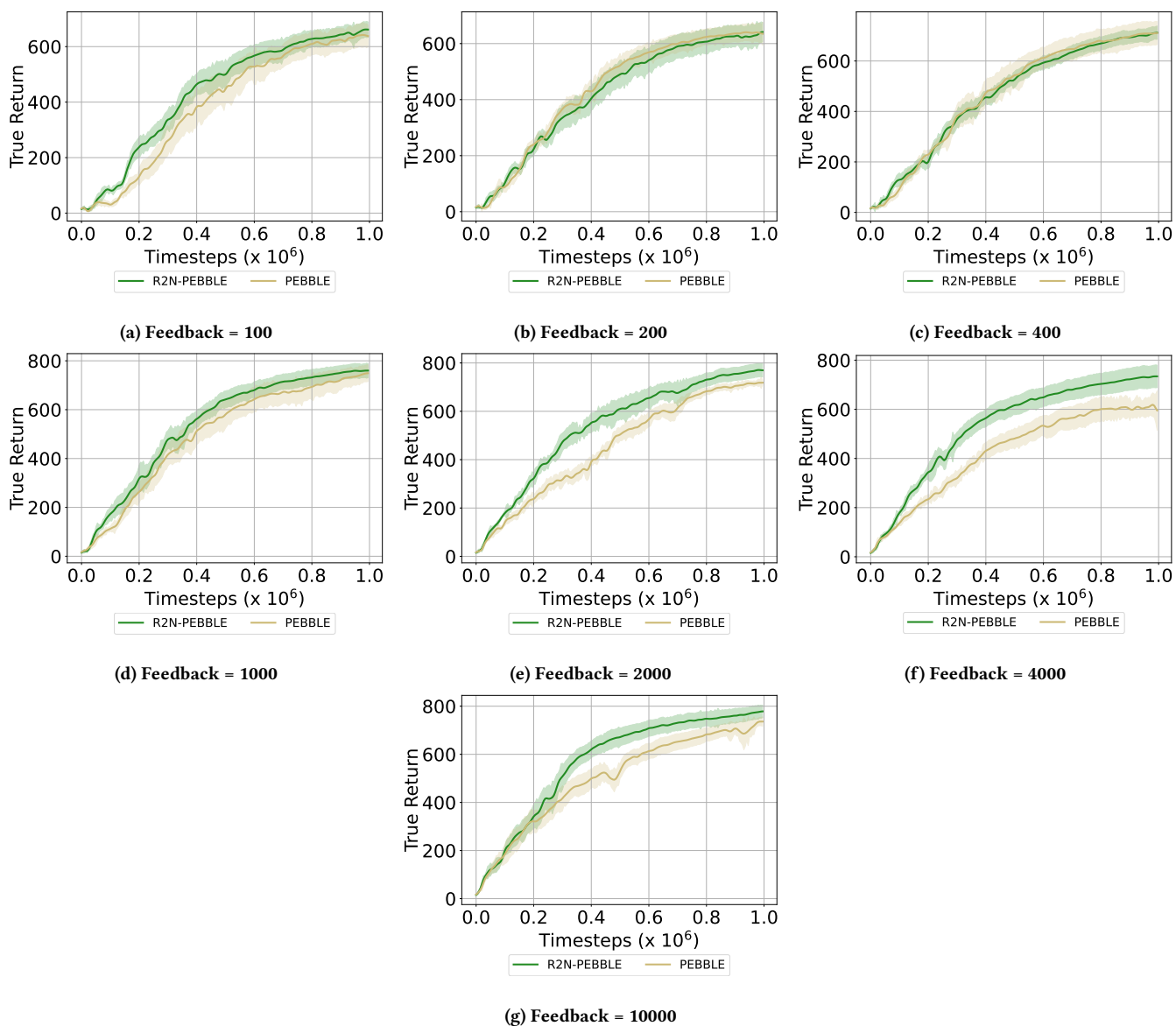
Hyperparameter	Value
<i>PEBBLE (reward model hyperparameters shared by all PbRL algorithms)</i>	
trajectory segment size	50
number of unsupervised exploration steps	9000
learning rate	0.003
batch size	128
ensemble size	3
network type	MLP
number of hidden layers	4
number of neurons per hidden layer	128
nonlinearity	LeakyReLU
optimizer	Adam [20]
replay buffer size	$10^5$
feedback frequency	5000
reward batch size	feedback budget / 100
trajectory sampling scheme	DMC – Uniform
training epochs	50
<i>SURF</i>	
confidence threshold $\tau$	0.99
loss weight $\lambda$	1
inverse label ratio	10
data augmentation window	5
crop range	5
<i>RUNE</i>	
beta schedule	linear decay
beta init	0.05
beta decay	0.00001
<i>R2N (reward models)</i>	
sparsity level input layer $s_i$	0.8
drop fraction $d_f$	0.20
topology-change period $\Delta T$	100

Table 4: Hyperparameters for Sparse Training Baselines

Hyperparameter	Value
<i>SET-PEBBLE</i>	
sparsity level input layer $s_i$	0.8
drop fraction $d_f$	0.20
<i>Static-PEBBLE</i>	
sparsity level input layer $s_i$	0.8
<i>DropConnect</i>	
drop fraction $d_f$	0.20
<i>L1 Regularization</i>	
$\lambda$	0.01



**Figure 6: Walker-walk.** These plots show the average number of neural network connections to the relevant versus noise features in R2N’s RL networks and reward models.



**Figure 7: Cheetah-run, Feedback Ablation, Noise = 20%**

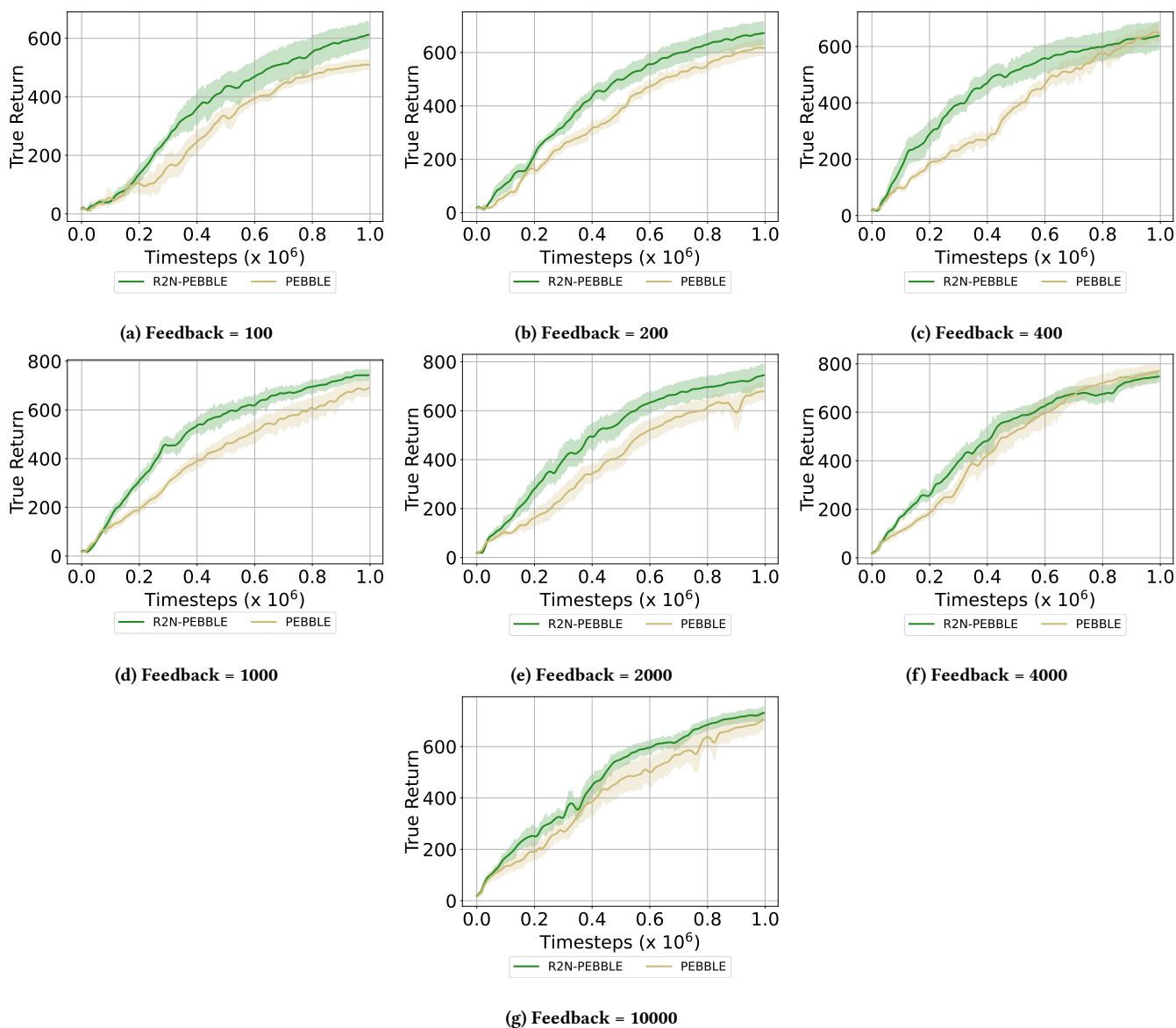


Figure 8: Cheetah-run, Feedback Ablation, Noise = 50%



### B.3 Feedback and Noise Ablations in Walker-walk

In this section, we present additional experimental results ablating the noise levels and preference budgets for the Walker-walk environment. Overall, in Figures 11 through 14, we find that R2N-PEBBLE (green curves) is generally more robust than PEBBLE (yellow curves) in varying noise levels and feedback amounts.

### B.4 Comparison of R2N with Traditional RL Algorithms

In this section, we show additional results comparing R2N-PEBBLE (green curve) with two RL algorithms, SAC (brown curve) and ANF-SAC (black curve). We note that both these algorithms learn while accessing the true environmental reward which is unavailable to R2N. Impressively, in Figure 15, we find that in three out of the five tested DMControl environments, R2N can achieve comparable results with SAC and ANF-SAC.

### B.5 Imitating Noise Analysis

In this section, we show additional results of the experiment where noise features imitate the distribution of real features, as described in Section 7. We find that in the Cheetah-run environment, R2N-PEBBLE (green dotted curve) maintains performance gains over PEBBLE (yellow dotted curve) when we increase the preference budget to 10000 (see Figure 16a). However, we find that for Walker-walk, R2N-PEBBLE performs comparably to PEBBLE for preference budgets of 4000 and 10000 (see Figures 16b and 16c).

### B.6 DST Component Analysis

In this section, we aim to understand the importance of dynamic sparse training on both reward learning and RL modules. In Figure 17, we find that in Walker-walk, full R2N which applies DST to both learning modules is superior to R2N variants that only apply DST to one learning module.

### B.7 R2N on Zero Noise Environments

In this section, we show additional results comparing R2N-PEBBLE (green curve) with PEBBLE (yellow curve) in traditional RL environments without any added noise features. Overall in Figure 18, we find that R2N-PEBBLE can achieve performance gains in some environments (Humanoid-stand), however, in others, it performs comparable or slightly worse. In this set of experiments, we kept the feedback budgets the same as the experiments with high added noise. Therefore, it might be the case that in 0% noise environments, R2N-PEBBLE can achieve greater performance gains in lower feedback regimes.

### B.8 Tests for Statistical Significance: R2N versus Sparse Training Baselines

This section provides a full summary of all experiments under RQ 1. Tables 5 and 6 show the average AUC and final return for R2N and the four sparse training baselines in the tested DMControl experiments. We also include the outcome of a Welch’s  $t$ -test (equal

variances not assumed). We use a p-value significance threshold of 0.05.

### B.9 Tests for Statistical Significance for: Effectiveness of R2N across PbRL Algorithms

This section provides a full summary of all experiments under RQ 2. Table 8 and Table 9 show the average final return and average area under the curve respectively over 14 seeds. We perform statistical tests between R2N and each baseline it is paired with, showing the outcome of Welch’s  $t$ -test (equal variances not assumed). We use a p-value significance threshold of 0.05. Note that in all 15 cases, R2N significantly increases both the final performance and the area under the curve (AUC) of its baseline.

### B.10 Tests for Statistical Significance for Sensitivity and Ablation Studies

This section provides a full summary of all experiments presented in Section 7. Tables 10–13 show the average final return and average area under the curve respectively over five seeds for the various sensitivity and ablation studies. We perform statistical tests between R2N and each baseline it is paired with, showing the outcome of Welch’s  $t$ -test (equal variances not assumed). We use a p-value significance threshold of 0.05.

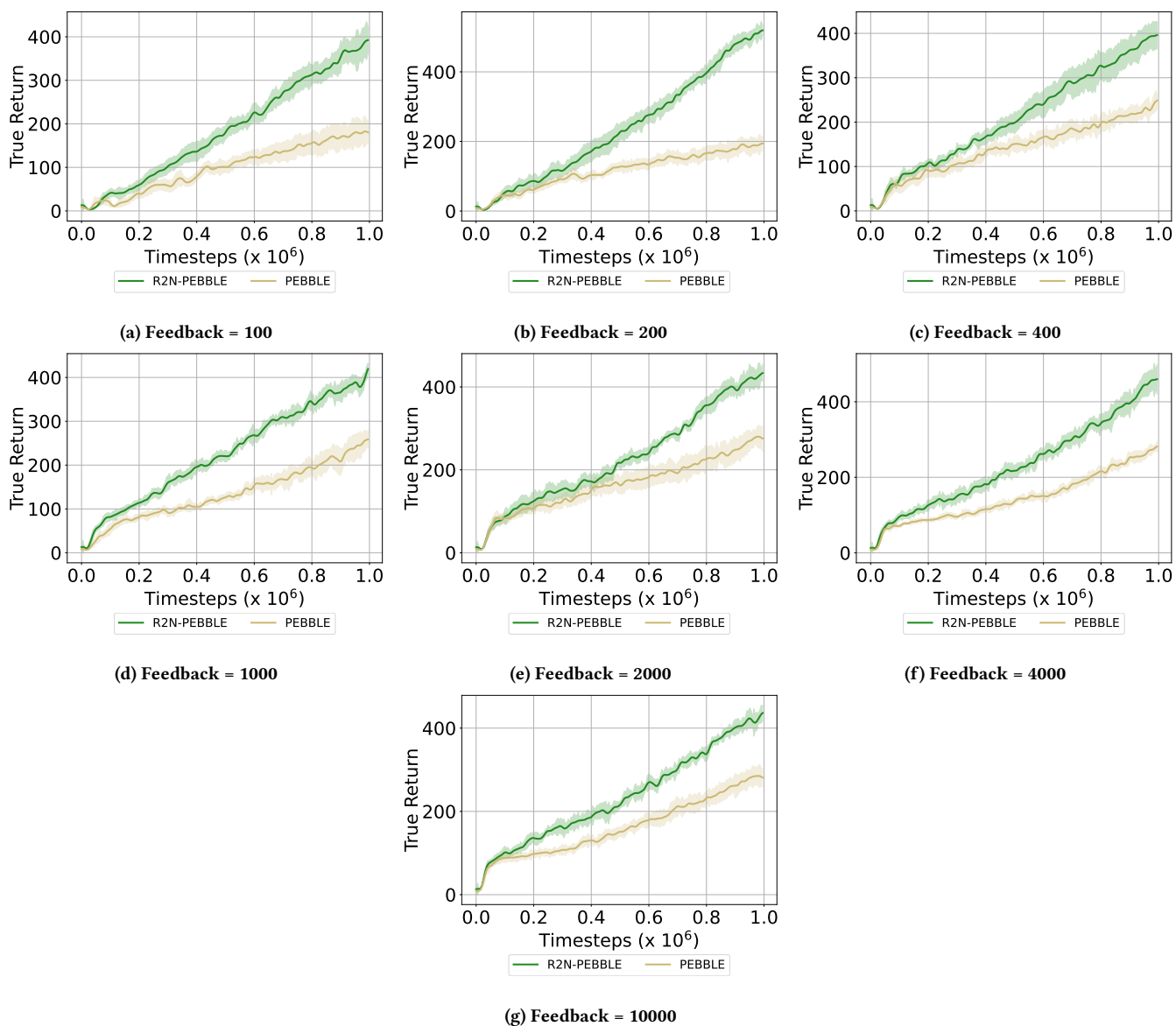


Figure 9: Cheetah-run, Feedback Ablation, Noise = 90%

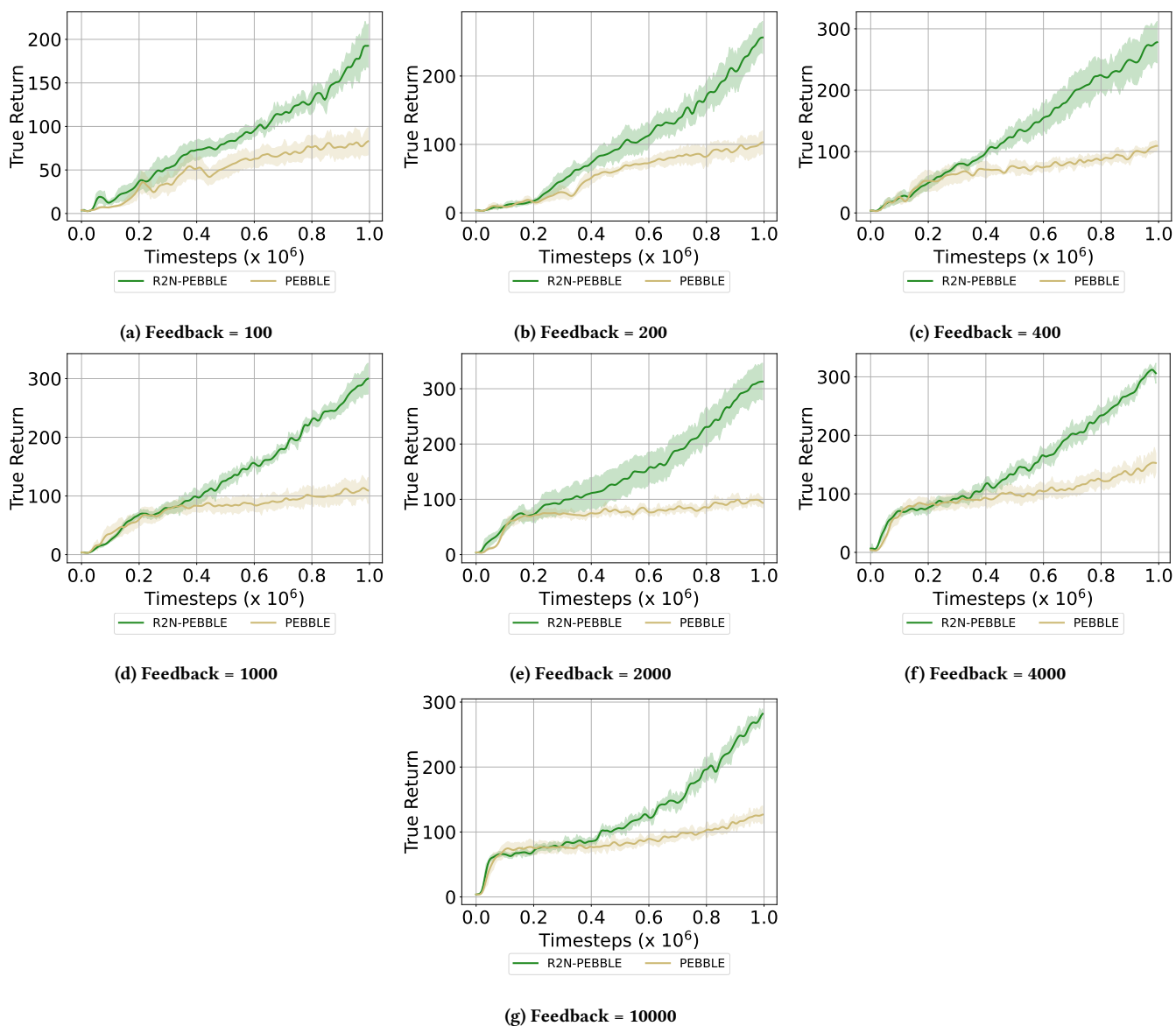


Figure 10: Cheetah-run, Feedback Ablation, Noise = 95%

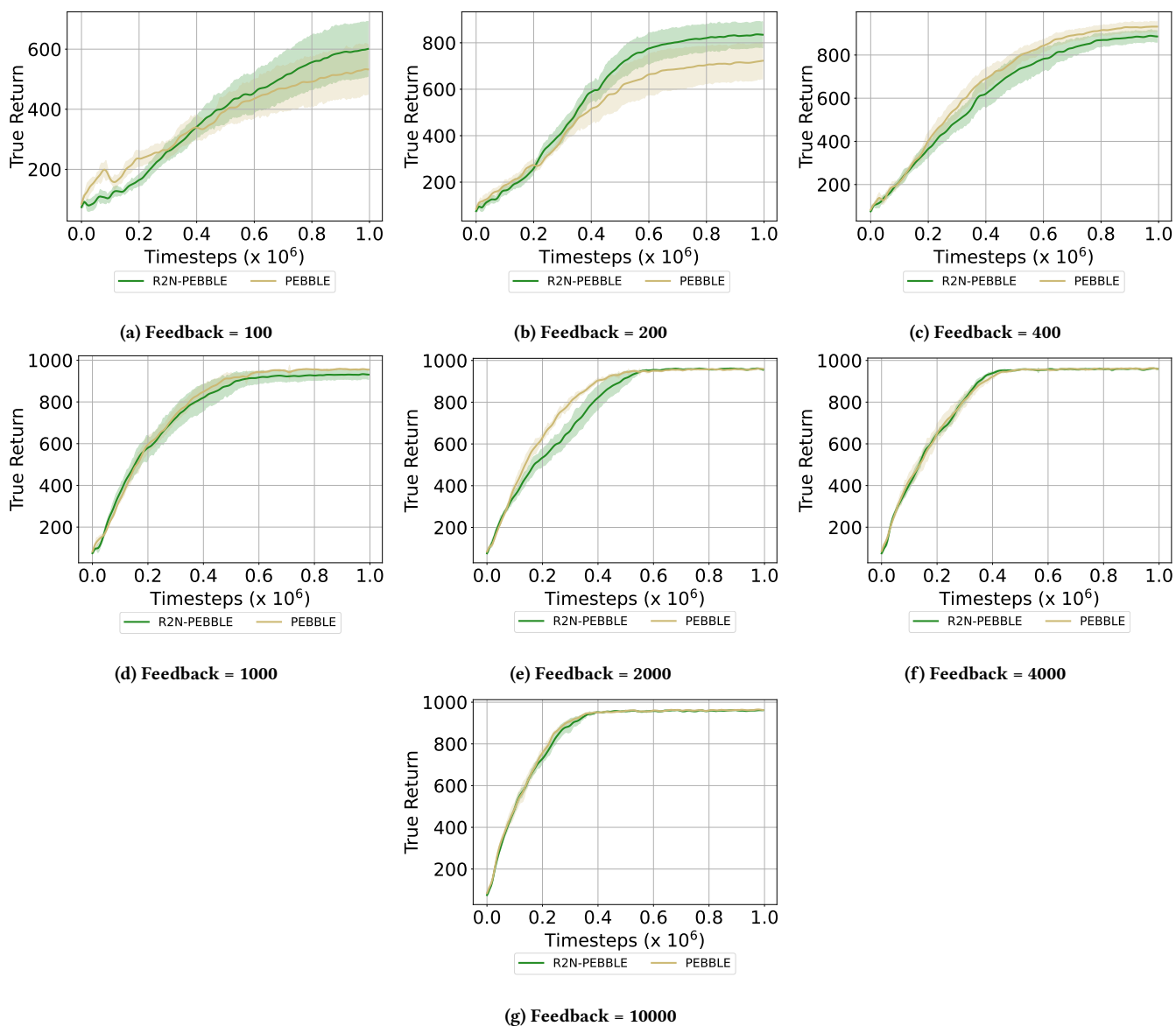


Figure 11: Walker-walk, Feedback Ablation, Noise = 20%

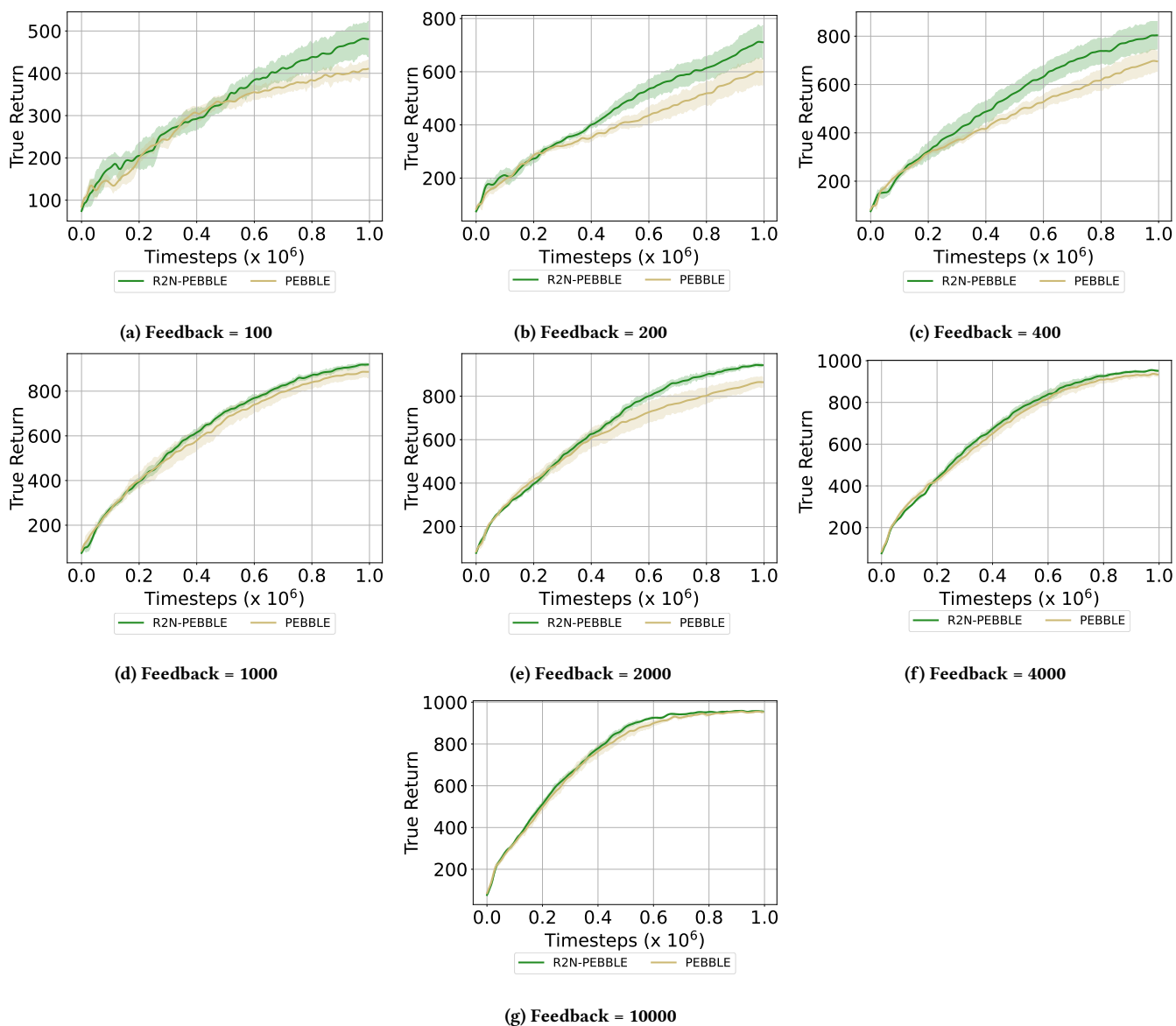


Figure 12: Walker-walk, Feedback Ablation, Noise = 50%



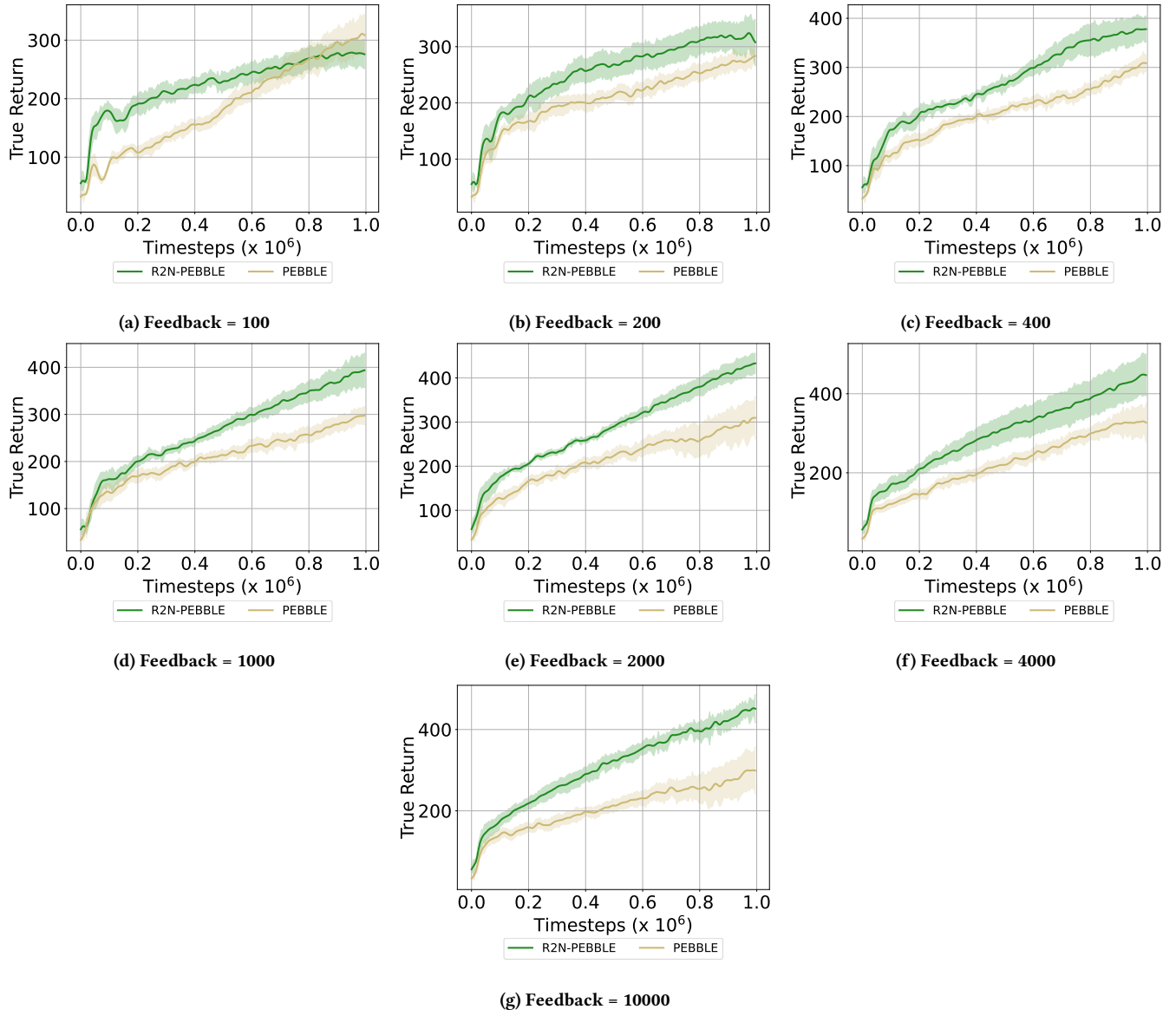


Figure 13: Walker-walk, Feedback Ablation, Noise = 90%

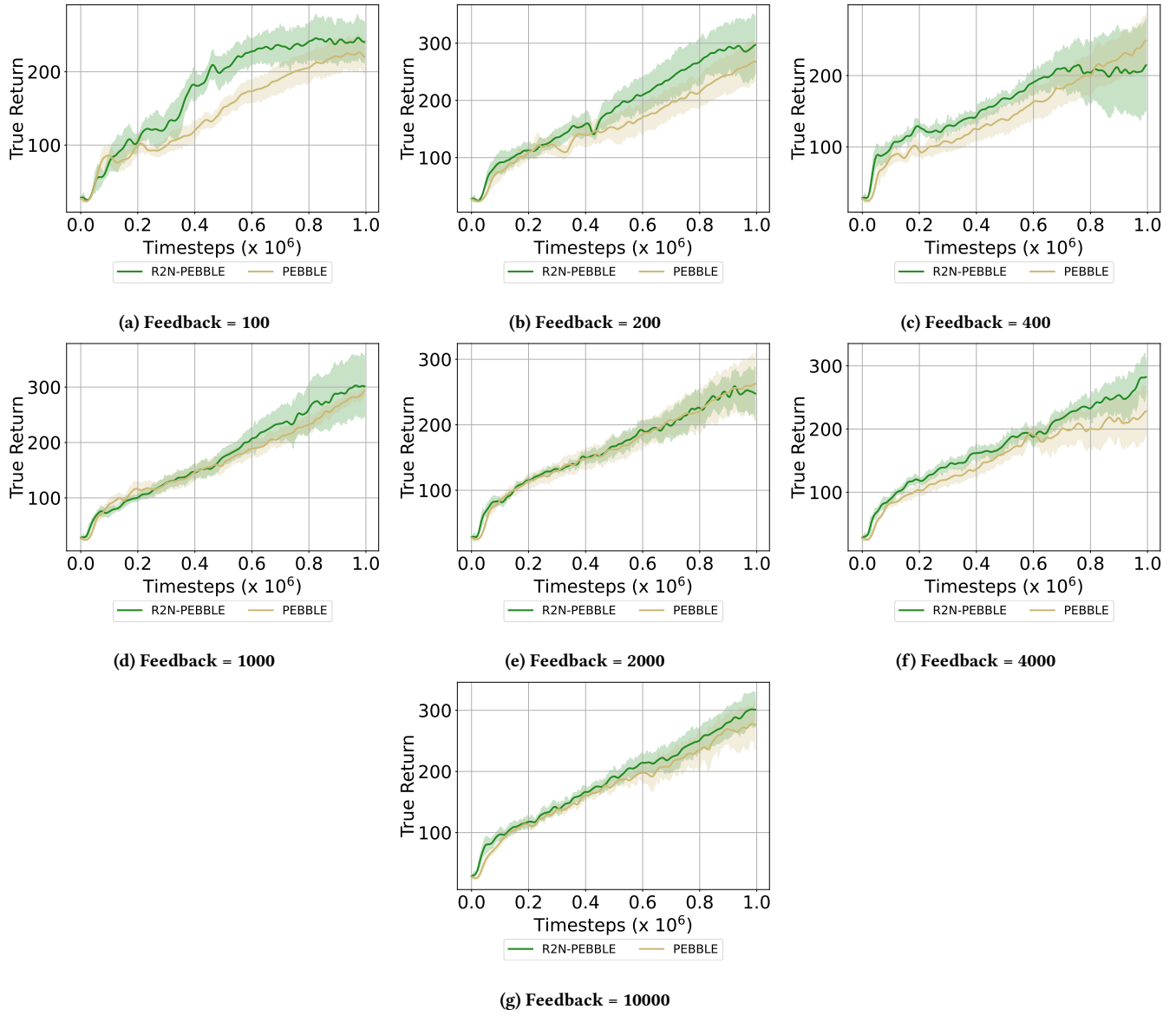


Figure 14: Walker-walk, Feedback Ablation, Noise = 95%

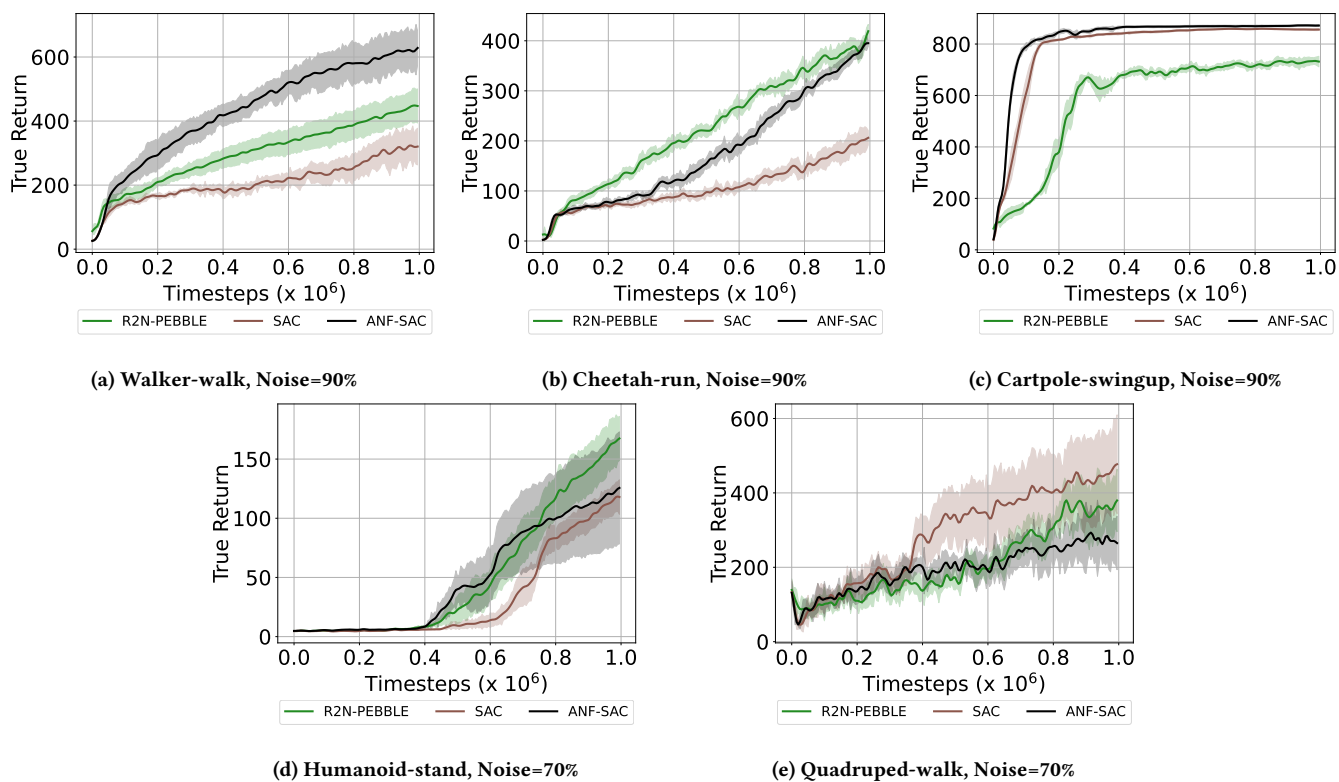
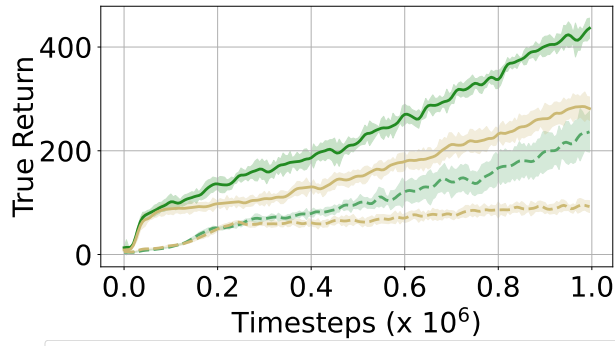
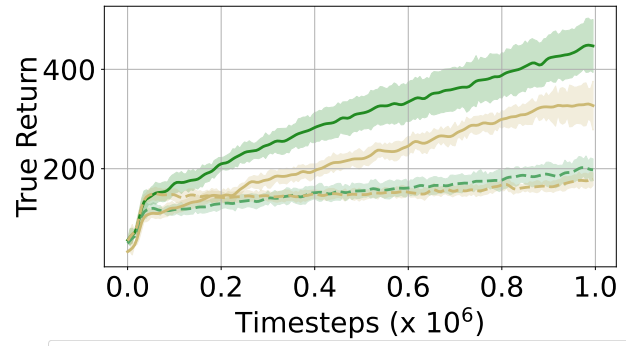


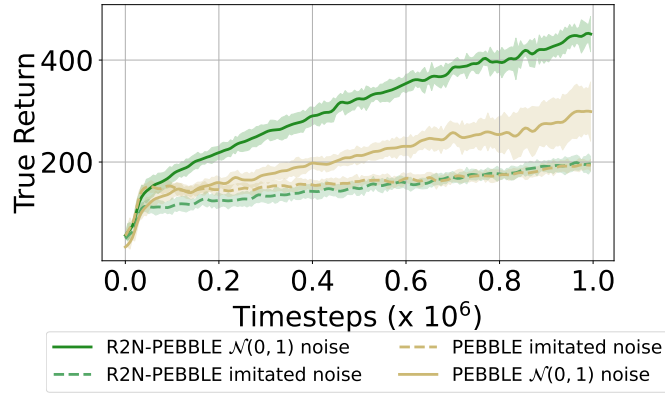
Figure 15: These plots compare R2N-PEBBLE with SAC and ANF-SAC, algorithms that have access to the ground truth reward function.



(a) Cheetah-run, Noise=90%, Feedback=10000

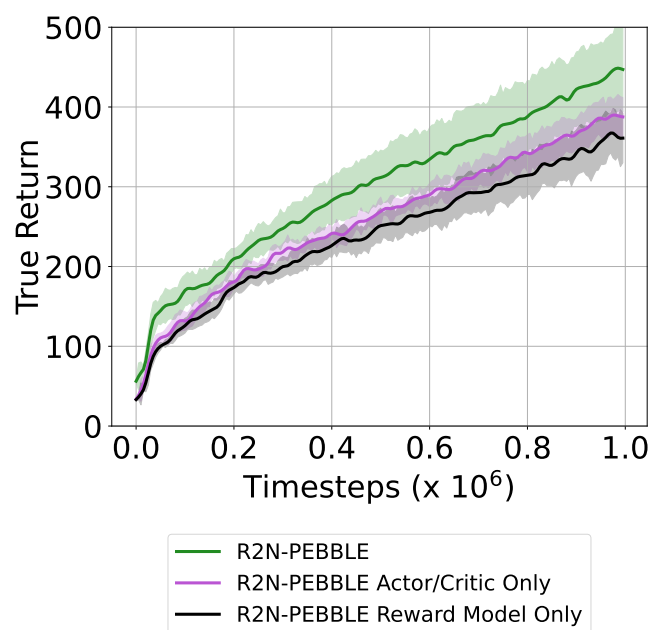


(b) Walker-walk, Noise=90%, Feedback=4000



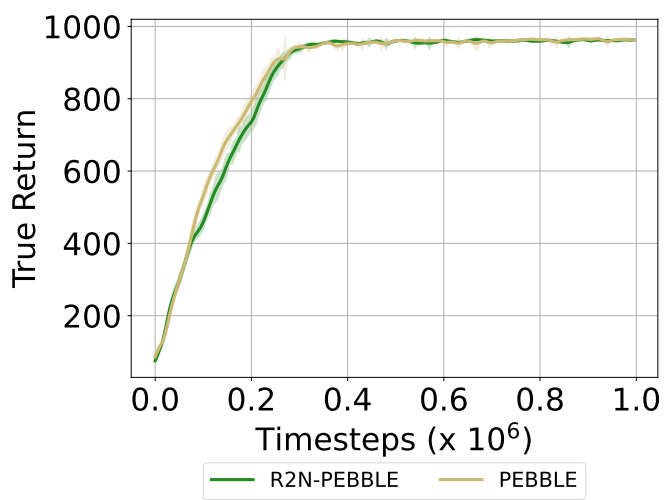
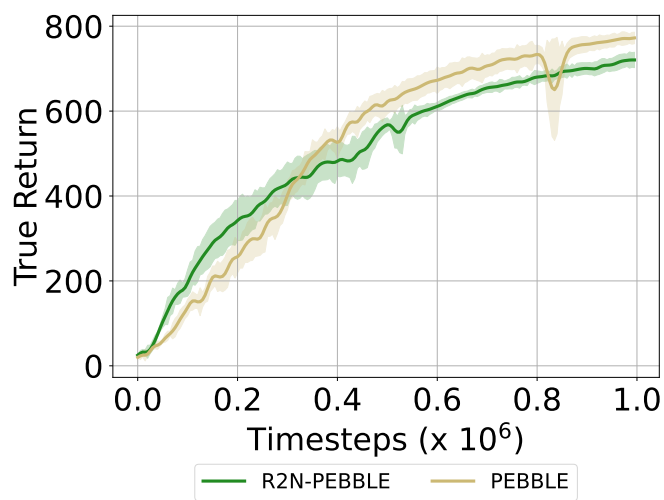
(c) Walker-walk, Noise=90%, Feedback=10000

Figure 16: These plots compare the performance of R2N-PEBBLE and PEBBLE in the ENE setting where the noise features imitate the task-relevant features.



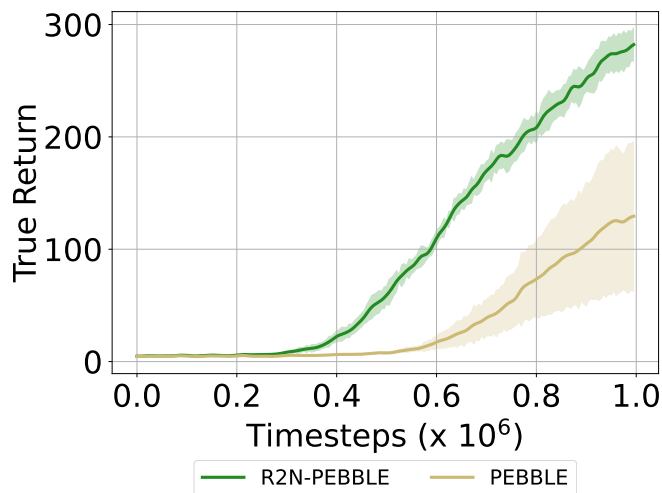
**Figure 17:** This plot shows the performance of R2N-PEBBLE in Walker-walk (noise = 90%, feedback = 4000) with R2N variants that only apply DST to either the RL modules (actor/critic) or the reward learning module.





(a) Cheetah-run, Feedback=1000

(b) Walker-walk, Feedback=4000



(c) Humanoid-stand, Feedback=10000

Figure 18: These plots compare the performance of R2N-PEBBLE and PEBBLE in traditional RL environments (i.e., 0% noise setting).

**Table 5: This table shows the average area under the curve (AUC) +/- standard error across all methods and environments. \* indicates a significant difference between R2N and the other sparse training baselines.**

Task	Feedback / Noise	Method	AUC	P Value
Cartpole-swingup	400 / 0.9	R2N-PEBBLE	117315.62 +/- 2335.79	
		Static-R2N-PEBBLE	116461.45 +/- 2334.99	0.411
		SET-R2N-PEBBLE	112016.81 +/- 6987.0	0.269
		DropConnect-PEBBLE	117025.07 +/- 2073.38	0.4678
		L1Reg-PEBBLE	110815.92 +/- 4545.69	0.144
Walker-walk	4000 / 0.9	R2N-PEBBLE	59633.26 +/- 6314.67	
		Static-R2N-PEBBLE	54004.32 +/- 4094.93	0.261
		SET-R2N-PEBBLE	47873.41 +/- 1013.76	0.069
		DropConnect-PEBBLE	38794.09 +/- 3707.38	0.017*
		L1Reg-PEBBLE	43978.68 +/- 2085.58	0.034*
Cheetah-run	1000 / 0.9	R2N-PEBBLE	45055.91 +/- 1383.87	
		Static-R2N-PEBBLE	39629.12 +/- 1361.03	0.018*
		SET-R2N-PEBBLE	41051.56 +/- 2109.16	0.096
		DropConnect-PEBBLE	24896.96 +/- 2316.76	0.000*
		L1Reg-PEBBLE	26874.27 +/- 1625.5	0.000*
Quadruped-walk	4000 / 0.7	R2N-PEBBLE	41388.91 +/- 3133.91	
		Static-R2N-PEBBLE	41030.21 +/- 1388.97	0.463
		SET-R2N-PEBBLE	37897.23 +/- 2112.66	0.216
		DropConnect-PEBBLE	29980.62 +/- 1000.13	0.007*
		L1Reg-PEBBLE	28905.57 +/- 2408.5	0.011*
Humanoid-stand	10000 / 0.7	R2N-PEBBLE	10231.46 +/- 1680.98	
		Static-R2N-PEBBLE	4323.92 +/- 997.25	0.013*
		SET-R2N-PEBBLE	2310.3 +/- 668.14	0.002*
		DropConnect-PEBBLE	1109.44 +/- 69.78	0.000*
		L1Reg-PEBBLE	1170.72 +/- 56.38	0.000*

Task	Feedback / Noise	Method	Final Performance	P Value
Cartpole-swingup	400 / 0.9	R2N-PEBBLE	729.95 +/- 14.26	
		Static-R2N-PEBBLE	737.59 +/- 16.23	0.620
		SET-PEBBLE	728.63 +/- 20.24	0.481
		DropConnect-PEBBLE	719.91 +/- 7.62	0.296
		L1Reg-PEBBLE	690.3 +/- 17.57	0.077
Walker-walk	4000 / 0.9	R2N-PEBBLE	434.85 +/- 48.74	
		Static-R2N-PEBBLE	367.56 +/- 30.18	0.162
		SET-PEBBLE	329.17 +/- 11.53	0.047*
		DropConnect-PEBBLE	252.73 +/- 34.83	0.013*
		L1Reg-PEBBLE	340.34 +/- 33.39	0.095
Cheetah-run	1000 / 0.9	R2N-PEBBLE	384.8 +/- 12.53	
		Static-R2N-PEBBLE	377.23 +/- 13.53	0.361
		SET-PEBBLE	404.03 +/- 13.36	0.812
		DropConnect-PEBBLE	219.24 +/- 19.39	0.000*
		L1Reg-PEBBLE	239.53 +/- 14.52	0.000*
Quadruped-walk	4000 / 0.7	R2N-PEBBLE	355.62 +/- 73.09	
		Static-R2N-PEBBLE	283.55 +/- 20.56	0.210
		SET-PEBBLE	245.51 +/- 12.33	0.110
		DropConnect-PEBBLE	207.51 +/- 14.16	0.056
		L1Reg-PEBBLE	218.3 +/- 34.69	0.083
Humanoid-stand	10000 / 0.7	R2N-PEBBLE	154.63 +/- 20.46	
		Static-R2N-PEBBLE	81.42 +/- 19.54	0.024*
		SET-PEBBLE	43.65 +/- 19.9	0.004*
		DropConnect-PEBBLE	7.93 +/- 1.64	0.000*
		L1Reg-PEBBLE	8.34 +/- 1.22	0.000*

Table 6: This table shows the average area under the curve (AUC) +/- standard error across all methods and environments. \* indicates a significant difference between R2N and the other sparse training baselines.

PERCENT IMPROVEMENT OF R2N			
TASK / FEEDBACK / NOISE FRACTION	R2N-PEBBLE v. PEBBLE	R2N-RUNE v. RUNE	R2N-SURF v. SURF
CARTPOLE-SWINGUP / 400 / 0.90	8.540%	8.21%	7.90%
WALKER-WALK / 4000 / 0.90	33.60%	36.59%	44.98%
CHEETAH-RUN / 1000 / 0.90	70.26%	103.06%	55.74%
QUADRUPED-WALK / 4000 / 0.70	66.83%	39.67%	39.55%
HUMANOID-STAND / 10000 / 0.70	1942.97%	2165.27%	1266.18%

Table 7: This table shows the percent improvement in the average final return of the original PbRL algorithms once R2N is applied to it. This highlights that in all tested cases, R2N can boost the performance of the base PbRL algorithm.

**Table 8: This table shows the average final return +/- standard error across all methods and environments. \* indicates a significant difference between R2N and the original PbRL algorithm.**

Task	Feedback / Noise	Method	Final Return	P Value
Cartpole-swingup	400 / 0.9	<b>R2N-PEBBLE</b>	<b>716.89 +/- 9.75*</b>	0.001
		PEBBLE	660.48 +/- 11.47	
		<b>R2N-RUNE</b>	<b>731.78 +/- 8.69*</b>	0.000
		RUNE	674.94 +/- 5.52	
		<b>R2N-SURF</b>	<b>734.73 +/- 10.83*</b>	0.003
		SURF	680.90 +/- 13.79	
Walker-walk	4000 / 0.9	<b>R2N-PEBBLE</b>	<b>455.22 +/- 21.62*</b>	0.000
		PEBBLE	340.72 +/- 16.29	
		<b>R2N-RUNE</b>	<b>462.98 +/- 20.32*</b>	0.000
		RUNE	338.94 +/- 24.78	
		<b>R2N-SURF</b>	<b>411.19 +/- 23.52*</b>	0.000
		SURF	283.60 +/- 11.33	
Cheetah-run	1000 / 0.9	<b>R2N-PEBBLE</b>	<b>404.59 +/- 17.12*</b>	0.000
		PEBBLE	237.62 +/- 14.43	
		<b>R2N-RUNE</b>	<b>410.94 +/- 19.18*</b>	0.000
		RUNE	202.37 +/- 15.26	
		<b>R2N-SURF</b>	<b>405.68 +/- 18.38*</b>	0.000
		SURF	260.47 +/- 10.74	
Quadruped-walk	4000 / 0.7	<b>R2N-PEBBLE</b>	<b>309.33 +/- 29.36*</b>	0.000
		PEBBLE	185.41 +/- 8.88	
		<b>R2N-RUNE</b>	<b>266.06 +/- 24.87*</b>	0.005
		RUNE	190.48 +/- 9.30	
		<b>R2N-SURF</b>	<b>274.78 +/- 13.77*</b>	0.006
		SURF	196.90 +/- 24.05	
Humanoid-stand	10000 / 0.7	<b>R2N-PEBBLE</b>	<b>127.89 +/- 20.31*</b>	0.000
		PEBBLE	6.26 +/- 0.18	
		<b>R2N-RUNE</b>	<b>140.90 +/- 21.94*</b>	0.000
		RUNE	6.22 +/- 0.19	
		<b>R2N-SURF</b>	<b>74.73 +/- 18.66*</b>	0.000
		SURF	5.47 +/- 0.19	

Table 9: This table shows the average area under the curve (AUC) +/- standard error across all methods and environments. \* indicates a significant difference between R2N and the original PbRL algorithm.

Task	Feedback / Noise	Method	AUC	P Value
Cartpole-swingup	400 / 0.9	<b>R2N-PEBBLE</b>	<b>114986.66 +/- 2597.90*</b>	0.004
		PEBBLE	104113.11 +/- 2612.11	
		<b>R2N-RUNE</b>	<b>114836.05 +/- 2165.07*</b>	0.001
		RUNE	106055.52 +/- 1370.12	
		<b>R2N-SURF</b>	<b>117204.59 +/- 3351.06*</b>	0.021
		SURF	107394.51 +/- 2908.52	
Walker-walk	4000 / 0.9	<b>R2N-PEBBLE</b>	<b>61814.89 +/- 2902.65*</b>	0.000
		PEBBLE	45987.92 +/- 1716.72	
		<b>R2N-RUNE</b>	<b>64458.89 +/- 2444.11*</b>	0.000
		RUNE	45895.60 +/- 2579.91	
		<b>R2N-SURF</b>	<b>51750.05 +/- 2031.67*</b>	0.000
		SURF	41111.81 +/- 1387.92	
Cheetah-run	1000 / 0.9	<b>R2N-PEBBLE</b>	<b>43970.86 +/- 1898.99*</b>	0.000
		PEBBLE	27585.24 +/- 1743.22	
		<b>R2N-RUNE</b>	<b>42316.32 +/- 1626.45*</b>	0.000
		RUNE	22849.97 +/- 894.95	
		<b>R2N-SURF</b>	<b>45172.59 +/- 2385.34*</b>	0.000
		SURF	28496.79 +/- 1338.12	
Quadruped-walk	4000 / 0.7	<b>R2N-PEBBLE</b>	<b>39639.90 +/- 1498.20*</b>	0.000
		PEBBLE	29227.14 +/- 1320.89	
		<b>R2N-RUNE</b>	<b>36803.72 +/- 1287.16*</b>	0.001
		RUNE	29744.02 +/- 1331.92	
		<b>R2N-SURF</b>	<b>37356.69 +/- 1390.05*</b>	0.000
		SURF	28689.95 +/- 1460.15	
Humanoid-stand	10000 / 0.7	<b>R2N-PEBBLE</b>	<b>8508.52 +/- 1357.08*</b>	0.000
		PEBBLE	1058.61 +/- 13.09	
		<b>R2N-RUNE</b>	<b>9112.00 +/- 1421.13*</b>	0.000
		RUNE	1059.46 +/- 13.33	
		<b>R2N-SURF</b>	<b>4987.07 +/- 1251.07*</b>	0.002
		SURF	985.91 +/- 17.57	

Table 10: This plot shows the average area under the curve (AUC) and final return +/- standard error for the noise ablation in Cheetah-run. \* indicates a significant difference between R2N-PEBBLE and PEBBLE.

Noise Level	Method	AUC	P Value	Final Return	P Value
0	R2N-PEBBLE	100172.87 +/- 3499.24	0.710	709.93 +/- 15.80	0.975
	PEBBLE	103557.90 +/- 3922.01		765.43 +/- 14.62	
20	R2N-PEBBLE	107612.78 +/- 5146.40	0.216	757.15 +/- 28.88	0.352
	PEBBLE	99416.24 +/- 7195.32		736.78 +/- 36.13	
50	<b>R2N-PEBBLE</b>	<b>102043.73 +/- 4298.56*</b>	0.014	737.09 +/- 21.96	0.103
	PEBBLE	82705.98 +/- 4826.48		678.45 +/- 31.12	
70	<b>R2N-PEBBLE</b>	<b>69993.54 +/- 2737.56*</b>	0.009	<b>598.03 +/- 28.87*</b>	0.007
	PEBBLE	55111.51 +/- 3559.99		473.89 +/- 20.99	
90	<b>R2N-PEBBLE</b>	<b>45055.91 +/- 1383.87*</b>	0.000	<b>384.80 +/- 12.53*</b>	0.001
	PEBBLE	26506.93 +/- 1444.38		238.24 +/- 23.91	
95	<b>R2N-PEBBLE</b>	<b>27014.93 +/- 491.60</b>	0.001	<b>279.80 +/- 18.30*</b>	0.000
	PEBBLE	15493.66 +/- 2258.69		109.26 +/- 16.56	

Table 11: This table shows the average area under the curve (AUC) and final return +/- standard error for the feedback study in Cheetah-run. \* indicates a significant difference between R2N-PEBBLE and PEBBLE.

Feedback Amount	Method	AUC	P Value	Final Return	P Value
100	<b>R2N-PEBBLE</b>	<b>37114.41 +/- 3084.44*</b>	0.002	<b>372.74 +/- 30.01*</b>	0.001
	PEBBLE	19610.52 +/- 2607.00		176.43 +/- 28.41	
200	<b>R2N-PEBBLE</b>	<b>47562.73 +/- 2197.19*</b>	0.000	<b>498.50 +/- 18.10*</b>	0.000
	PEBBLE	23088.88 +/- 2265.25		186.83 +/- 19.61	
400	<b>R2N-PEBBLE</b>	<b>41644.86 +/- 3360.05*</b>	0.008	<b>380.38 +/- 29.99*</b>	0.001
	PEBBLE	27966.80 +/- 2221.47		227.58 +/- 15.53	
1000	<b>R2N-PEBBLE</b>	<b>45055.91 +/- 1383.87*</b>	0.000	<b>384.80 +/- 12.53*</b>	0.001
	PEBBLE	26506.94 +/- 1444.38		238.24 +/- 23.91	
2000	<b>R2N-PEBBLE</b>	<b>45139.38 +/- 1645.85*</b>	0.009	<b>414.54 +/- 18.11*</b>	0.001
	PEBBLE	32659.90 +/- 3447.91		266.72 +/- 23.28	
4000	<b>R2N-PEBBLE</b>	<b>46260.24 +/- 2147.41*</b>	0.000	<b>433.98 +/- 26.61*</b>	0.000
	PEBBLE	28900.51 +/- 1154.58		263.39 +/- 9.58	
10000	<b>R2N-PEBBLE</b>	<b>46704.99 +/- 1681.03*</b>	0.001	<b>415.72 +/- 16.04*</b>	0.001
	PEBBLE	31946.62 +/- 2314.48		276.67 +/- 22.17	

Table 12: This table shows the average area under the curve (AUC) and final return +/- standard error for the DST ablation in Cheetah-run. \*\* indicates a significant difference between R2N-PEBBLE and both R2N-Actor/Critic Only and R2N-Reward Model Only .

Method	AUC	P Value	Final Return	P Value
<b>R2N-PEBBLE</b>	<b>45055.91 +/- 1383.87**</b>		<b>384.80 +/- 12.53**</b>	
R2N-Actor/Critic Only	33159.39 +/- 1813.11	0.001	327.33 +/- 17.82	0.022
R2N-Reward Model Only	28643.04 +/- 2679.96	0.001	265.27 +/- 22.51	0.002

Table 13: This table shows the average area under the curve (AUC) and final return +/- standard error for the imitating noise study in Cheetah-run. \* indicates a significant difference between R2N-PEBBLE and PEBBLE .

Method	AUC	P Value	Final Return	P Value
<b>R2N-PEBBLE imitated noise</b>	<b>18081.23 +/- 1611.21*</b>		<b>220.19 +/- 26.47*</b>	
PEBBLE imitated noise	11356.92 +/- 660.51	0.004	93.92 +/- 50.70	0.001