# Flipping Coins to Estimate Pseudocounts for Exploration in Reinforcement Learning

**Sam Lobel** [*1] **Akhil Bagaria** [*1] **George Konidaris** [1]

## Abstract

We propose a new method for count-based exploration in high-dimensional state spaces. Unlike previous work which relies on density models, we show that counts can be derived by averaging samples from the Rademacher distribution (or *coin flips*). This insight is used to set up a simple supervised learning objective which, when optimized, yields a state's visitation count. We show that our method is significantly more effective at deducing ground-truth visitation counts than previous work; when used as an exploration bonus for a model-free reinforcement learning algorithm, it outperforms existing approaches on most of 9 challenging exploration tasks, including the Atari game MONTEZUMA'S REVENGE.

## 1. Introduction

Deep exploration is crucial to solving long-horizon problems using reinforcement learning (RL) (Osband et al., 2016). When the number of states is small, an agent can simply keep track of how many times it has visited each state. This count can then be used as an exploration bonus to train a near-optimal policy (Strehl & Littman, 2008). When the world is much bigger than the agent, it may never revisit the same state (Sutton et al., 2022). To facilitate count-based exploration in such domains, the notion of visitation counts has been generalized to that of "pseudocounts" (Bellemare et al., 2016) which behave similarly to counts but can be meaningfully applied in large or infinite state-spaces. Previous methods have equated the problem of estimating pseudocounts to the canonical machine learning problem of *density estimation*: the more informative a given state is to the model while learning, the higher the reward for reaching it (Bellemare et al., 2016; Ostrovski et al., 2017).

While providing the first way to estimate pseudocounts, Bellemare et al. (2016)'s relationship between counts and probability densities exists only when the density model meets the following restrictions (Ostrovski et al., 2017):

- It must output normalized probability densities, which precludes many powerful density models.
- It must be *learning-positive*, which means that the probability density of a state must increase when it is encountered by the density model again.
- It must be updated exactly once per state visitation, precluding common techniques such as batching.

These requirements make density-based pseudocounts challenging to implement and sensitive to network architecture and hyperparameters such as learning rate. In light of these restrictions, it is tempting to forego count-based exploration in favor of other novelty estimates based on dynamics (Pathak et al., 2017) or observation (Burda et al., 2019) prediction errors. But prediction-error based methods do not tell us how an exploration bonus should decay with repeated visits. Furthermore, they do not enjoy the same theoretical foundations afforded by count-based methods (Strehl & Littman, 2008; Azar et al., 2017; Jin et al., 2018). For instance, count-based bonuses lead to near-optimal policies even when environments are highly stochastic; no such guarantees exist for prediction-error based methods.

We hypothesize that count-based exploration can be more effective than prediction-error based methods if we can compute pseudocounts under a less restrictive setting. Our core insight is that a state's visitation count can be derived from the sampling distribution of Rademacher trials made every time a state is encountered. We train a neural network, the *Coin Flip Network* (CFN), to predict the average of this sampling distribution; by solving this supervised learning problem, we output the inverse of the state's visitation count. Unlike other pseudocount methods (Ostrovski et al., 2017), we do not place any restrictions on the type of function approximator or the procedure used to train it, thereby allowing a practitioner to select the model architecture best suited to their input modality.

[*]Equal contribution [1]Department of Computer Science, Brown University, Providence, RI, USA. Correspondence to: Sam Lobel <samuel_lobel@brown.edu>, Akhil Bagaria <akhil_bagaria@brown.edu>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

We show that in visual versions of Gridworld (Allen et al., 2021) and TAXI (Dietterich, 1998), our method can recover the ground-truth counts while other pseudocount methods cannot. We then evaluate our algorithm on a variety of challenging sparse-reward continuous control problems; in these environments, we outperform baseline actor-critic (Haarnoja et al., 2018) and random network distillation (Burda et al., 2019), with the largest gains on the most challenging exploration domains. On the image-based exploration benchmark problem MONTEZUMA'S REVENGE (Bellemare et al., 2013), we outperform baseline Rainbow (Hessel et al., 2018) and are competitive with state-of-the-art exploration algorithms (Ostrovski et al., 2017; Burda et al., 2019), which arguably have been overfit to this domain (Taiga et al., 2020). Finally, we show that increasing transition noise in Gridworld and MONTEZUMA'S REVENGE causes RND's performance to degrade more rapidly than CFN's, as predicted by the theoretical properties of count-based exploration.

## 2. Background and Related Work

We consider problems modeled as Markov Decision Processes (MDPs) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$ where $\mathcal{S}$ is the state-space, $\mathcal{A}$ is the action-space, $\mathcal{R}$ is a reward function, $\mathcal{T}$ is a transition function and $\gamma$ is the discount factor. The aim of the agent is to learn a policy that maximizes the expected discounted sum of rewards (Sutton & Barto, 2018).

### 2.1. Deep reinforcement learning

Model-free RL algorithms often use variants of Q-learning (Watkins & Dayan, 1992) to learn an action-value function $Q_\theta(s, a)$ and then act greedily with respect to it. This Q-function can be learned in high-dimensional spaces using non-linear function approximators (parameterized by $\theta$) by minimizing the loss $L(\theta) = \mathbb{E}[(Q_\theta(s, a) - y_t)^2]$, where the Q-learning target $y_t$ is given by the following equation (Mnih et al., 2015):

$$y_t = \mathcal{R}_t + \gamma \max_{a_{t+1}} Q_{\theta'}(s_{t+1}, a_{t+1}),$$

and $\theta'$ are the parameters of a slowly changing target network (Mnih et al., 2015). This principle has since been extended with various algorithmic improvements; for example, Rainbow (Hessel et al., 2018) for discrete action-spaces and Soft Actor Critic (SAC) (Haarnoja et al., 2018) for continuous action-spaces. The majority RL systems use myopic strategies for exploration (e.g, $\epsilon$-greedy, action noise), which do not scale to long-horizon problems.

### 2.2. Bonus-based exploration

A promising strategy for exploration is to incorporate an *intrinsic reward* that encourages the agent to gather informative data. This intrinsic reward $\mathcal{B}(s_t, a_t)$ is added to the extrinsic reward to create an augmented Q-learning target:

$$y_t = \mathcal{R}_t + \lambda \mathcal{B}(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a'), \quad (1)$$

where $\lambda \in \mathbb{R}^+$ modulates the scale of the exploration bonus. Acting greedily with respect to this Q-function balances exploration and exploitation, and is the basis for many provably efficient (Strehl & Littman, 2008; Jin et al., 2018) and practically successful (Taiga et al., 2020) algorithms. As is typical in the literature, we consider bonuses $\mathcal{B}(s)$ that are only dependent on state (Burda et al., 2018).

**Count-based exploration.** In tabular domains, a count-based exploration bonus of $1/\sqrt{N(s)}$ (near) optimally trades-off exploration and exploitation, even in highly stochastic MDPs (Auer, 2002; Strehl & Littman, 2008). This approach was extended to function approximation by using density models to calculate pseudocounts: first with the CTS model (Bellemare et al., 2016) and then with PixelCNN (Ostrovski et al., 2017). However, it is challenging to learn density models in high-dimensional observation spaces—especially given the restrictions discussed in the Introduction. Successor Counts (Machado et al., 2020) also bypasses density modeling (by relating pseudocounts to the norm of successor representations (Dayan, 1993)). But in problems that require deep exploration (Taiga et al., 2020), they are outperformed by PixelCNN (Machado et al., 2020), which we compare our method to in Section 4.

Many methods bypass learning and resort to workarounds that heavily incorporate domain knowledge—for example, #-Counts (Tang et al., 2017) and OPIQ (Rashid et al., 2020) use locality sensitive hashing, Go-explore (Ecoffet et al., 2021) severely downsamples input images before binning them, and MEGA assumes knowledge of which dimensions of state are useful for the task (Pitis et al., 2020). By contrast, CFN takes raw observations as input and flexibly learns representations optimized for predicting exploration bonuses.

**Model-prediction error.** Many methods learn a transition model and use the error in the predicted next state as an exploration bonus (Stadie et al., 2015; Houthooft et al., 2016; Pathak et al., 2017; Ermolov & Sebe, 2020; Guo et al., 2022). This causes the agent to collect more data where the transition model is least accurate (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002; Kakade et al., 2003). However, most methods learn deterministic models (Pathak et al., 2017; Guo et al., 2022), making them difficult to scale to stochastic environments.

**Random Network Distillation (RND).** Due to its simplicity and empirical strength, RND (Burda et al., 2019) has emerged as the most popular exploration algorithm. RND assigns an exploration bonus to a state using a simple,

elegant heuristic: the novelty of a state is directly proportional to how the accurately a trainable network can mimic a randomly-initialized network's projection of it. Compared to count-based methods, RND's exploration bonus is difficult to interpret—it is an unnormalized distance in a neural network latent space. Furthermore, it is unclear at what rate the RND bonus falls with visitation count: this depends on the learning dynamics of neural networks and stochastic gradient descent. In Section 4 we investigate the shape of RND's bonus and compare it to CFN.

### 2.3. Uncertainty Estimation

Some methods explicitly estimate epistemic uncertainty in the value function and use that to drive exploration (Osband et al., 2016; 2018; O'Donoghue et al., 2018). These are promising realizations of Thompson sampling (Thompson, 1933) in high-dimensional domains, but empirically underperform optimism driven approaches.

### 2.4. Integrated exploratory RL agents

Novelty estimates are often integrated into larger RL agents that use more sophisticated techniques such as episodic memory (Badia et al., 2020b), adaptive horizons (Badia et al., 2020a; Kapturowski et al., 2022), transferable representation learning (Zhang et al., 2021; Raileanu & Rocktäschel, 2020), goal-conditioned policies (Pong et al., 2019) or planning (Bagaria et al., 2021). Our method improves how the novelty bonus itself is computed and can be included in any of these agents without much modification.

## 3. Coin Flip Network (CFN)

A pseudocount $\mathcal{N} : \mathcal{S} \to \mathbb{R}^+$ generalizes the notion of counts to large state-spaces and can be used to quantify the novelty of a state (Bellemare et al., 2016). Specifically, visiting a state $s$ affords the agent a novelty bonus of $\mathcal{B}(s) = \frac{1}{\sqrt{\mathcal{N}(s)}}$; we will use a neural network, the *Coin Flip Network* (CFN) $f_\phi$, to directly predict this count-based exploration bonus.

To learn $f_\phi$, we set up a simple regression problem:

$$f_\phi = \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}_{(s_i,y_i)\sim\mathcal{D}}\Big[\mathcal{L}(s_i, y_i)\Big], \tag{2}$$

where $\mathcal{L}$ is the mean-square error loss function and $\mathcal{D}$ is a dataset of state-label pairs. Our main insights relate to the design of the labels $y_i$ in such a way that the resulting function $f_\phi$ will map each state to its count-based exploration bonus $\frac{1}{\sqrt{\mathcal{N}(s)}}$.

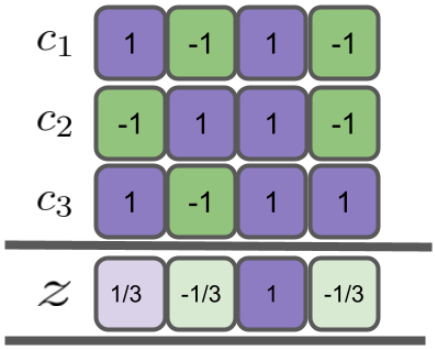


*Figure 1.* Illustration of our counting method for a state $s$ with true count 3. Each occurrence of $s$ creates new coin-flip vectors $c_1, c_2, c_3$. We average these vectors into $z$ and compute the squared magnitude. Dividing this by the number of coin flips $d = 4$ yields the inverse count $1/\mathcal{N}(s)$.

### 3.1. Counts from the Rademacher distribution

The first step in designing the labels $y_i$ in Eq 2 is to notice that moments of the Rademacher/coin-flip distribution directly encode counts.

Consider the fair coin-flip distribution $\mathcal{C}$ over outcomes $\{-1, 1\}$. Imagine flipping this coin $n$ times, and averaging the results into $z_n$. In expectation, the average is 0, but any given trial likely results in a non-zero value for $z_n$. Generally, for all $n$, the second moment of $z_n$ is related to the inverse-count:

$$\mathcal{M}_2(z_n) = \mathbb{E}[z_n^2] = \sum_i Pr(z_n = i) * i^2 = 1/n. \tag{3}$$

This property is a simple restatement of the fact that $\mathbb{E}[z_n^2]$ is the *variance of the sample mean* of the coin-flip distribution, which is well-known to scale inversely with sample size. In fact, this scaling is shared between all zero-mean unit-variance distributions, not just the coin-flip distribution. However, using this distribution leads to the *lowest variance estimates of inverse-counts* out of the entire class of matching distributions. We prove these two facts in Appendix A.1 and A.2 respectively.

### 3.2. Estimating counts for a state via multiple coin flips

An additional way to lower the variance of this estimator is to average together multiple estimates of $z_n^2$: by flipping $d$ coins each time, we get $d$ independent estimates of $\frac{1}{n}$, which reduces variance by a factor of $\frac{1}{d}$ (see Appendix A.4).

Consider the contrived case of an MDP with a single state and imagine that we draw a random sample from $\mathcal{C}^d$ each time that state is visited. Equation 3 implies that the squared magnitude of the averaged vectors is an unbiased estimator of $d/n$; this is also illustrated in Figure 1. Of course, we do

not need a novel method to count the number of elements in a list; but this informs our eventual method for producing bonuses in general MDPs.

### 3.3. Predicting counts for multiple states

Having solved the uninteresting problem of extracting counts for a single state MDP, we will now generalize to datasets with multiple occurrences of multiple states. As label $y_i$ for state $s_i$ in Eq 2, we generate a $d$-dimensional random vector $\mathbf{c_i} \sim \{-1, 1\}^d$; this leads to the following simplification of Equation 2:

$$f_\phi^*(s) = \operatorname*{argmin}_\phi \sum_{i=1}^{|\mathcal{D}|} \|\mathbf{c_i} - f_\phi(s_i)\|^2$$
$$= \operatorname*{argmin}_\phi \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{d} (c_{ij} - f_\phi(s_i)_j)^2. \quad (4)$$

When there are multiple instances of the same state $s$ in $\mathcal{D}$, each occurrence will be paired with a different random vector. In that case, $f_\phi^*$ cannot learn a perfect mapping from states to labels, and instead minimizes $\mathcal{L}$ by outputting the *mean* random vector for all instances of a given state:

$$f_\phi^*(s) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{c_i}.$$

Combining Equation 3 and 4 relates the solution $f_\phi^*$ to the inverse count:

$$f_\phi^*(s) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{c_i}$$
$$\implies \mathbb{E}\Big[\frac{1}{d}\|f_\phi^*(s)\|^2\Big] = \frac{1}{d} \sum_{j=1}^{d} \mathbb{E}\Big[\Big(\sum_{i=1}^{n} \frac{c_{ij}}{n}\Big)^2\Big]$$
$$= \frac{1}{d} \sum_{j=1}^{d} \mathbb{E}\Big[z_n^2\Big]$$
$$= \frac{1}{d} \sum_{j=1}^{d} \frac{1}{n} = \frac{1}{n}.$$

Thus, by training $f_\phi$ on the objective described in Equation 4 we can map states to approximate count-based bonuses:

$$\boxed{\mathcal{B}(s) := \sqrt{\frac{1}{d}\|f_\phi(s)\|^2} \approx \frac{1}{\sqrt{\mathcal{N}(s)}}} \quad (5)$$

### 3.4. Generalizing outside the training data

The optimization procedure described so far will eventually derive the correct visitation counts for states in the training data (given a powerful function approximator and sufficient training iterations). But as the agent interacts with the environment, how will the CFN bonus generalize to states absent from the training data? Although in practice we represent $f_\phi$ as a neural network, to gain intuition on generalization, we mathematically examine the case when $f_\phi$ is linear. In this case, the bonus for a state $s$ is a linear combination of the bonuses assigned to the right singular vectors of the training data; more discussion and proof is in Appendix B. Intuitively, the singular vectors of the training data take on the role of unique states: instead of tracking how many of each state visitation there are, a linear $f_\phi$ records how much of each singular vector is present in total in the dataset. Interestingly, when states are represented using one-hot vectors, the resulting solution to Equation 4 recovers tabular counts.

### 3.5. Improving predictions for novel states

As stated above, a learning architecture with infinite capacity would learn the exact inverse count for each unique state in the training data. But finite capacity and training time imply that the network will not learn this mapping exactly. Next, we will propose two ways to guide our network to favorably trade-off prediction errors among states in the dataset: first, we will use *prioritized sampling* to preferentially learn the novelty of rare states; second, we will use *optimistic initialization* to assign a pseudocount of 1 to novel states newly added to the replay buffer.

#### 3.5.1. Prioritizing Novel States

Since training to convergence at every time step is not feasible, we update $f_\phi$ once every time step on a mini-batch of states drawn from a replay buffer. Revisiting the optimization problem from Equation 4, we note that a state $s$ with count $n$ will appear in uniform sampling $n$ times more often than a state visited only once. This would make $f_\phi$ focus too much on learning the bonus of high count states, which are uninteresting from an exploration perspective. To remedy this problem, we would like to assign more weight to low count states by sampling them with greater probability (Schaul et al., 2015).

Of course, we do not have access to the true count during training; so, we approximate this procedure by prioritizing by our current *estimate* of inverse-count:

$$\text{priority}(s) \leftarrow \frac{1}{d}\|f_\phi(s)\|^2 \approx \frac{1}{\mathcal{N}(s)}.$$

Though this prioritization changes the importance of different states relative to each other, all instances of the same state will be sampled in equal proportion. Therefore, solving the prioritized version of Equation 4 still outputs the unbiased average of a state's coin-flip vectors (and therefore

the correct pseudocounts).

Prioritizing in this way introduces another difficulty: if a state has been recently added to the replay buffer, it has not appeared in many gradient updates and thus we cannot trust our estimate of its count. To combat this, we also prioritize sampling by the number of times, $n_{\text{updates}}(s)$, we have sampled $s$ in the past. We combine both these prioritization schemes using an $\alpha$-weighted sum (we use $\alpha = 0.5$):

$$\text{priority}(s) = \alpha \left( \frac{1}{n_{\text{updates}}(s)} \right) + (1-\alpha)\frac{1}{d}\|f_\phi(s)\|^2. \quad (6)$$

The $n_{\text{updates}}$ term weighs different instances of the same state differently, but its effect on prioritization disappears quickly during training, so it does not influence the fixed point either.

#### 3.5.2. Optimistic Initialization of Bonus

Consider a state $s$ that CFN has not been trained on yet. The exploration bonus $\mathcal{B}(s)$ will be determined by CFN's generalization properties. If $s$ is very different than the other states that CFN has already been trained on, then $\mathcal{B}(s)$ tends to be close to 0, when in fact we would like the pseudocount of novel states to be initialized to 1.

We achieve this optimistic initialization using a *random prior* (Osband et al., 2018):

$$f_\phi(s) = \hat{f}_\phi(s) + f_{\text{prior}}(s),$$

where $f_{\text{prior}}$ is the output of a frozen and randomly initialized neural network, and $\hat{f}_\phi$ is the trainable component of CFN. We use a running mean and variance to normalize the prior so that $\mathbb{E}_{s\sim\mathcal{D}}[f^{(i)}_{\text{prior}}(s)^2] = 1$ over all output dimensions $i \in \{1, .., d\}$. This ensures that if $\hat{f}_\phi(s) = 0$ on a novel state $s$, then $\|f_\phi(s)\|^2 = 1$, i.e., states are added to the buffer with an approximate initial pseudocount of 1. As training progresses, the effect of the initialization will wash out and $\mathcal{B}(s)$ will eventually settle to its correct value. An analysis of the optimistic prior's contribution is in Appendix E.2.

### 3.6. Integrated CFN agent

Algorithm 1 outlines how CFN is combined with Rainbow (Hessel et al., 2018) to form a complete bonus-based exploration agent. Naturally, we can combine CFN with most off-the-shelf RL algorithms with minor changes (Haarnoja et al., 2018; Mnih et al., 2015; Lillicrap et al., 2015).

**Algorithm 1** Rainbow-CFN Agent

**Hyperparameters**: Reward scale $\lambda$, Number of coin flips $d$

Initialize Q-network $Q_\theta$ and target Q-network $Q_{\theta'}$.
Initialize CFN prior $f_{\text{prior}}$ and trainable network $\hat{f}_\phi$.
Initialize replay buffer for Rainbow $B_r$ and CFN $B_c$.
Initialize optimizer for Rainbow and for CFN.
Initialize the running mean $\mu_t$ and variance $\sigma_t^2$ for the optimistic prior $f_{\text{prior}}$.
**while** training **do**
  $s_0 =$ `env.reset()`
  **while** not `done` **do**
    $a_t = \text{argmax}_{a\in\mathcal{A}}\, Q_\theta(s_t, a)$
    $R_t, s_{t+1},$ `done` = `env.step`$(s_t, a_t)$
    Compute intrinsic reward $\mathcal{B}(s_t)$ using Equation 5.
    Update $\mu_t$ and $\sigma_t^2$ using $f_{\text{prior}}(s_t)$.
    Add transition $(s_t, a_t, R_t, \mathcal{B}(s_t), s_{t+1})$ to $B_r$.
    Sample a random coin-flip vector $\mathbf{c} \sim \{-1, 1\}^d$.
    Add state coin-flip tuple $(s_t, \mathbf{c})$ to $B_c$.
    Sample minibatch $(s, a, r, \mathcal{B}_s, s') \sim B_r$ and update $Q_\theta$ using Rainbow's optimizer and Eq 1.
    Update priority for minibatch using Rainbow.
    Sample minibatch $(s, \mathbf{c}) \sim B_c$ and use CFN's optimizer to update $f_\phi$ via one gradient step on the loss function corresponding to Equation 4.
    Update priority for minibatch using Equation 6.
  **end while**
**end while**

## 4. Experiments

Our empirical results establish CFN as a competitive count-based exploration algorithm. First, we show that CFN can extract accurate counts in domains with visual observations, in contrast to other bonus methods. We then solve 8 sparse-reward continuous control problems using CFN and show that we significantly outperform RND and baseline SAC. Finally, we show that our method scales gracefully to the challenging Atari game Montezuma's Revenge.

**Implementation details.** All exploration methods are built on top of Rainbow (Hessel et al., 2018) (when the action-space is discrete) or Soft Actor Critic (SAC) (Haarnoja et al., 2018) (when the action-space is continuous) using the Dopamine library (Castro et al., 2018). CFN has the same neural network architecture as RND's prediction network. We follow the experimental design of Taiga et al. (2020). We use a different set of hyperparameters for each *suite* of tasks, i.e, one for Visual Gridworld, one for *Fetch*, one for Ant, one for Adroit and one for Montezuma's Revenge. Details about CFN, including hyperparameters can be found in the Appendix D; details about environments can be found in Appendix C. All code for reproducing results can be found at the linked repository.[1]

[1] https://github.com/samlobel/CFN

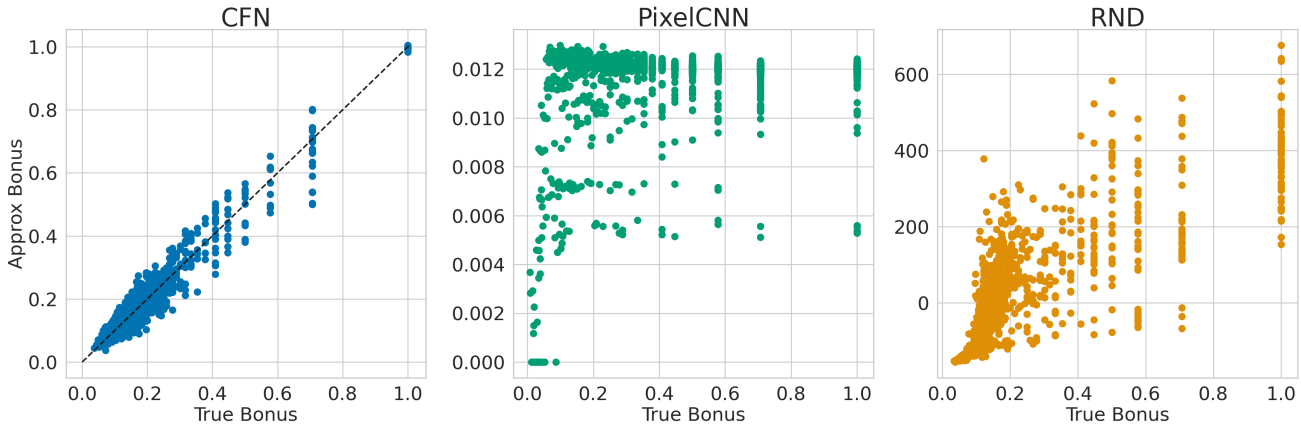


*Figure 2.* Predicted count-based bonuses for all three methods in Visual Gridworld after $100,000$ interactions. The horizontal axis is the ground truth $1/\sqrt{\hat{N}(s)}$ bonus, the vertical axis is the exploration bonus predicted by the different methods.

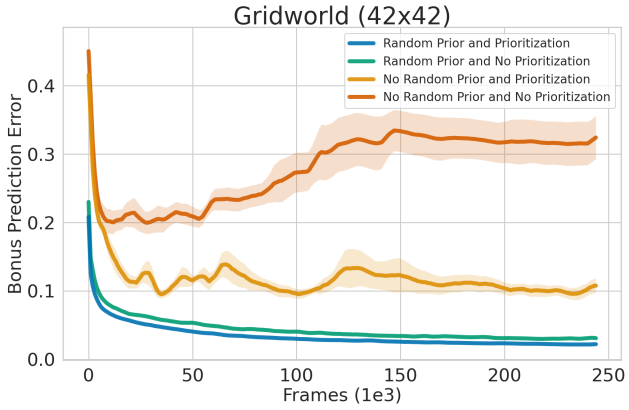


*Figure 3.* Ablating prioritized sampling and optimistic bonus initialization: vertical axis is the mean-squared error between the predicted and ground-truth count-based bonus (averaged over all visited states). Solid lines represent mean and bands represent standard error over 10 random seeds; lower is better.

### 4.1. Bonus prediction accuracy

We compare the exploration bonus from CFN to that of PixelCNN (Ostrovski et al., 2017) and RND (Burda et al., 2019) in Visual Gridworld (Allen et al., 2021). Observations are 84x84 images of a 42x42 grid; these images serve as inputs during training. The agent is initialized in the bottom-left, and achieves a sparse terminal reward of 1 for reaching the top-right within 150 timesteps. For evaluation, we keep track of the tabular state and the ground-truth visitation counts.

Figure 2 shows that while CFN is able to predict the count-based exploration bonus with high accuracy, PixelCNN and RND are not. PixelCNN and CFN, being pseudo-count methods, should ideally both output bonuses on the dashed line. Not only does PixelCNN mispredict the scale of the exploration bonus, it also assigns the same bonus to states visited once ($x = 1$) versus those visited 25 times ($x = 0.2$). RND's trendline is better than PixelCNN, although it has much higher variance than CFN. It is notable that for states with high count, its bonus falls off more sharply than $1/\sqrt{\hat{N}(s)}$. A similar experiment is repeated for the more challenging TAXI domain (Dietterich, 1998) (with image observations); results are in Appendix E.1.

### 4.2. Ablation: prioritization and random prior

We now ablate the contribution of prioritized sampling (Section 3.5.1) and random prior (Section 3.5.2). In Figure 3 we show how mean bonus prediction error evolves over time; the plot indicates that both additions lead to more accurate predictions. The prediction error is the mean-squared difference between the predicted and ground-truth exploration bonuses, averaged over unique states the agent has observed.

In Appendix E.3 we provide more insight into each of these curves, and show how both these additions to CFN improve its bonus accuracy on states in the low-count regime.

### 4.3. RL performance on Visual Gridworld

Figure 4 shows that CFN outperforms baseline Rainbow and PixelCNN, and performs similarly to RND on this task. An important feature to note about this environment is that it is deterministic. As such, the $1/\sqrt{\hat{N}(s)}$ bonus may not be appropriate because it is explicitly constructed to deal with stochastic environments (Auer, 2002; Jin et al., 2018). So, we compare CFN to RND on a series of increasingly stochastic versions of Visual Gridworld.[2] Our results show CFN's count-based bonus yields a more significant performance boost over RND in more stochastic versions of the problem. The slight performance bump at noise 0.1 can be attributed to the exploration benefit of random action-selection.

[2]Stochasticity is introduced by replacing the chosen action with a randomly selected action with some predefined probability.

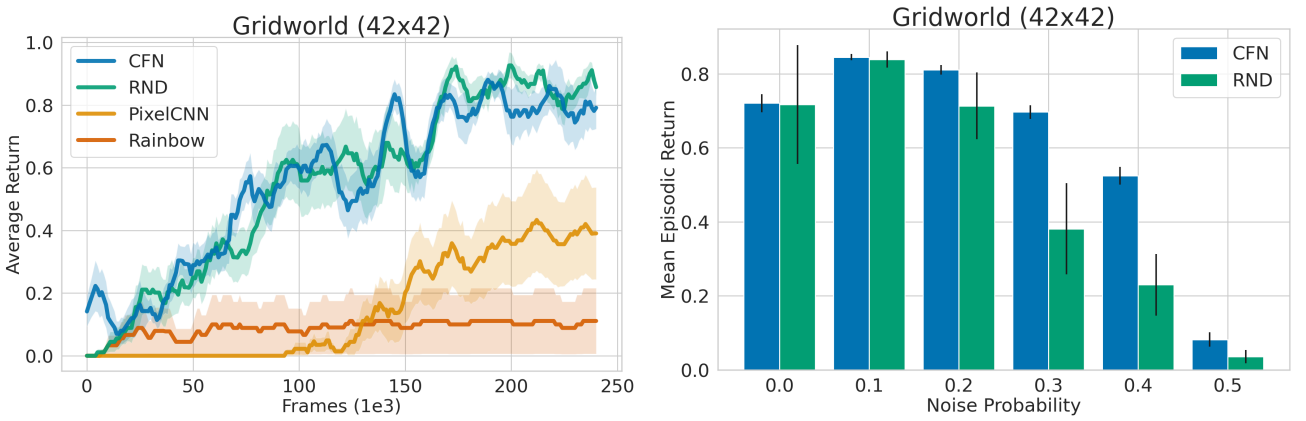


*Figure 4.* **Left**: Learning curves in deterministic Visual Gridworld comparing our method (CFN) with RND, PixelCNN and baseline Rainbow (with noisy networks). Solid lines denote mean episodic return, bands represent standard error.. **Right**: Comparison between CFN and RND on increasingly stochastic versions of Visual Gridworld. Bars represent mean episodic return averaged over training run, error bars denote standard error. All results are averaged over 10 random seeds.

### 4.4. Continuous Control Experiments

We now consider a series of challenging continuous control tasks. These are taken from two different suites: FETCH (Plappert et al., 2018) and D4RL (Fu et al., 2020). For all tasks, we sparsify the reward function to make exploration challenging. We compare CFN to RND and baseline SAC; we do not compare against PixelCNN because the inputs are not images.

**Fetch manipulation tasks.** These tasks involve controlling a simulated Fetch robot to perform a series of manipulation tasks: pushing, sliding, or lifting an object to a goal location (Plappert et al., 2018). We consider 3 modes for each task: default, medium and hard; these modes differ in start-goal configurations. The default task randomizes the start-goal states (which occasionally exposes the agent to very simple episodes), medium and hard versions fix them to different levels of difficulty. Figure 6 shows that both exploration methods outperform baseline SAC in all tasks; CFN outperforms RND on 6 out of 9 tasks and ties in 1.

**Ant-navigation and Adriot manipulation tasks.** Next, we consider tasks from D4RL.(Fu et al., 2020)[3] The first involves controlling a quadrupedal "ant" robot in a U-shaped maze. The remaining 4 tasks involve controlling a high-dimensional "Adriot" hand to perform various tasks: pick-and-place, reorienting a pen, opening a door and learning how to use a hammer (Rajeswaran et al., 2018). Similar to the Fetch tasks, we remove random restarts because they obviate the need for exploration (Lobel et al., 2022). Figure 5 shows that CFN outperforms SAC on all tasks and RND on 4 out of 5 tasks. More interestingly, the performance gains over RND are largest on the hardest exploration tasks (ANT U-MAZE and RELOCATE; as evidenced by SAC's inability to experience any positive rewards). This supports the hypothesis that CFN provides a more thorough exploration bonus than RND.

[3]We use the domains from this suite, not their offline datasets.

### 4.5. Performance in MONTEZUMA'S REVENGE

Finally, we test our method on the challenging exploration benchmark: MONTEZUMA'S REVENGE. We follow the experimental design suggested by Machado et al. (2015) and compare CFN to baseline Rainbow, PixelCNN and RND. Figure 7 shows that we comfortably outperform Rainbow in this task. All exploration algorithms perform similarly, a result also corroborated by Taiga et al. (2020).

Since all exploration methods perform similarly on the default task, we created a more challenging versions of MONTEZUMA'S REVENGE by varying the amount of transition noise (via the "sticky action" probability (Machado et al., 2018)). Figure 7 *(right)* shows that CFN outperforms RND at higher levels of stochasticity; this supports our hypothesis that count-based bonuses are better suited for stochastic environments than prediction-error based methods.

Notably, we find that having a large replay buffer for CFN slightly improves performance, which increases memory requirements for this experiment. More discussion about the impact of buffer size can be found in Appendix D.3; details about hyperparameters can be found in Appendix D.

## 5. Conclusion and Future Work

Though count-based exploration is a principled way to do exploration, it is not the dominant approach used in practice; we aim to remedy that. As a step in that direction,

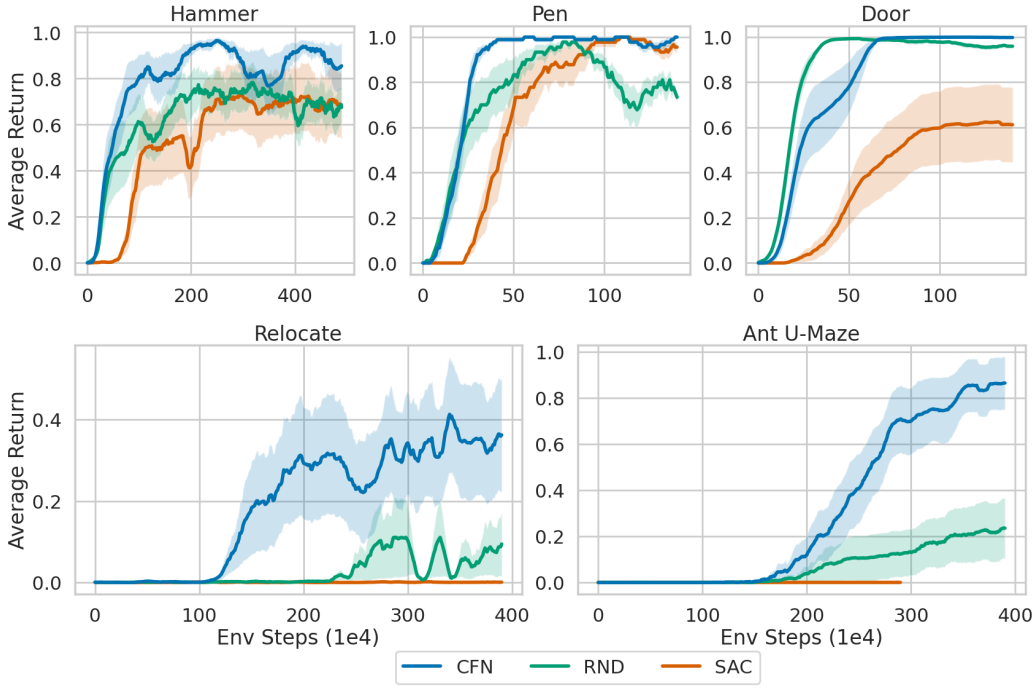


*Figure 5.* Learning curves in the D4RL tasks. Bottom row shows the 2 most challenging tasks in this task suite. All curves are averaged over 9 independent runs.

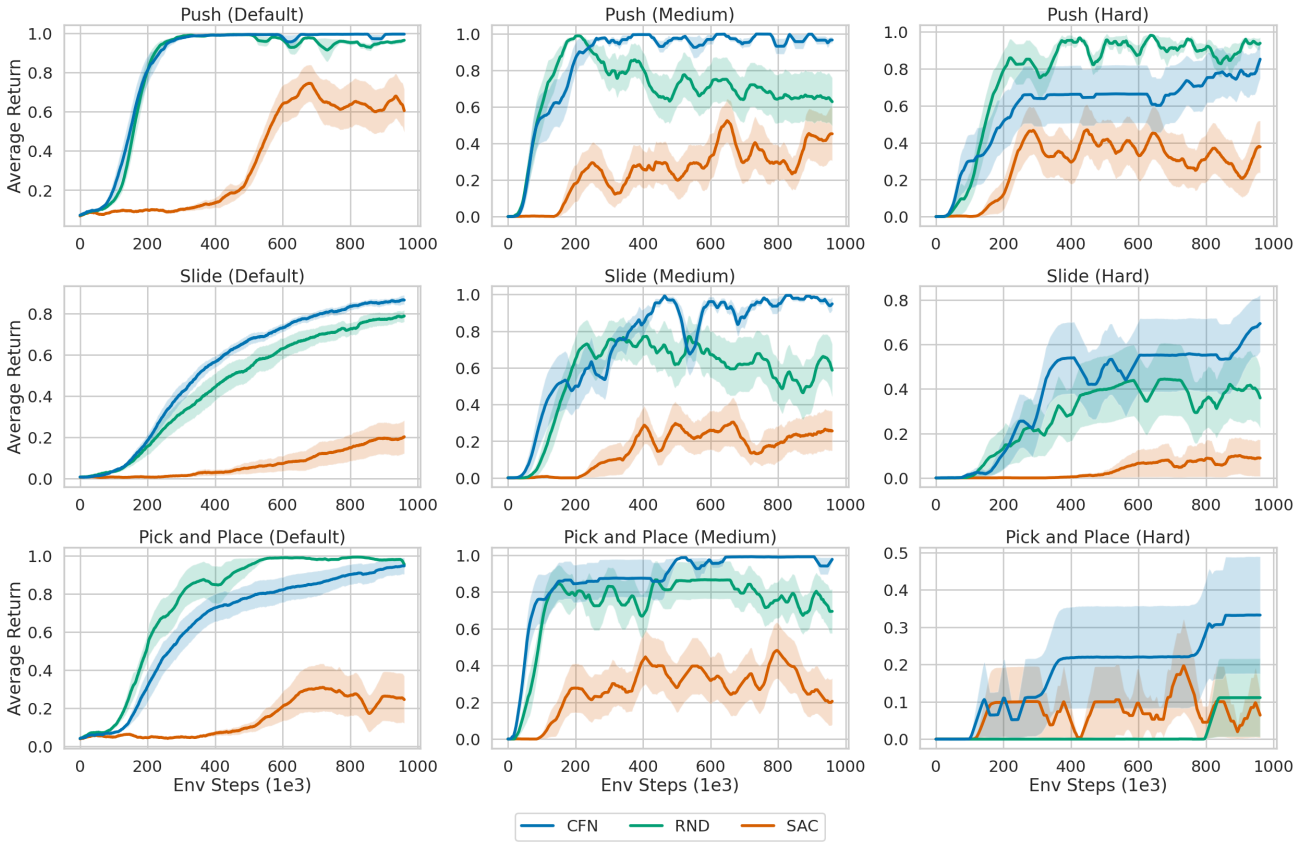


*Figure 6.* Results on the simulated FETCH manipulation tasks with 3 levels of difficulty (default/easy, medium and hard). All curves are averaged over 9 independent runs.

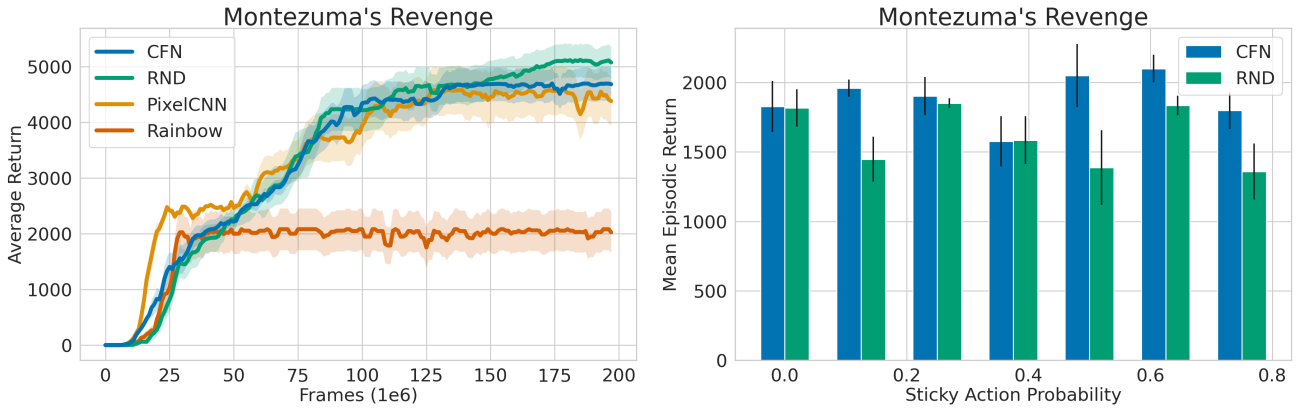


*Figure 7.* **Left**: Learning curve in Montezuma's Revenge comparing our method (CFN) with RND, PixelCNN and baseline Rainbow (with noisy networks). Solid lines denote mean episodic return, bands represent standard error averaged over 12 random seeds. **Right**: Comparison between CFN and RND in terms of mean cumulative reward over 100 million frames on versions of Montezuma's Revenge with varying "sticky action" probabilities (stochastic transitions; 0.25 is the default sticky action probability (Machado et al., 2018)). Error bars represent standard error over 5 seeds.

we presented a new method for count-based exploration which extracts pseudocounts by learning to average samples from the Rademacher distribution. In contrast to prior pseudocount methods, ours produces accurate counts in simple problems and can be flexibly applied to a variety of observation spaces. We demonstrate strong results on MONTEZUMA'S REVENGE and 8 challenging continuous control problems. Directions for future work include the use of representation learning techniques that capture MDP-specific structure (Allen et al., 2021), incorporating actions into the exploration bonus, as well as a mechanism for "forgetting" high-count states from the replay buffer.

## Acknowledgements

We would like to thank Georg Ostrovski for his guidance, Adrien Ali Taiga for pointing us to the bonus-based exploration code, and Cameron Allen for help with the Visual Gridworld and Taxi domain implementations. Part of this research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University. This research was supported in part by NSF grant #1955361, NSF CAREER award #1844960, NSF GRFP award #2040433, and an Amazon Research Award.

## References

Allen, C., Parikh, N., Gottesman, O., and Konidaris, G. Learning markov state abstractions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Azar, M. G., Osband, I., and Munos, R. Minimax regret bounds for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 263–272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/azar17a.html.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020a.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020b.

Bagaria, A., Senthil, J. K., and Konidaris, G. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, pp. 521–531. PMLR, 2021.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

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

Brafman, R. I. and Tennenholtz, M. R-max-a general polynomial time algorithm for near-optimal reinforcement

learning. *Journal of Machine Learning Research*, 3(Oct): 213–231, 2002.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*, 2018.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110.

Dayan, P. Improving generalization for temporal difference learning: The successor representation. *Neural computation*, 5(4):613–624, 1993.

Dietterich, T. The MAXQ method for hierarchical reinforcement learning. In *ICML*, volume 98, pp. 118–126, 1998.

Diuk, C., Cohen, A., and Littman, M. L. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 240–247, 2008.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 590(7847): 580–586, 2021.

Ermolov, A. and Sebe, N. Latent world models for intrinsically motivated exploration. *Advances in Neural Information Processing Systems*, 33:5565–5575, 2020.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Golub, G. H. and Reinsch, C. Singular value decomposition and least squares solutions. *Linear algebra*, 2:134–151, 1971.

Guo, Z. D., Thakoor, S., Pîslar, M., Pires, B. A., Altché, F., Tallec, C., Saade, A., Calandriello, D., Grill, J.-B., Tang, Y., et al. Byol-explore: Exploration by bootstrapped prediction. *arXiv preprint arXiv:2206.08332*, 2022.

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

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.

Jin, C., Allen-Zhu, Z., Bubeck, S., and Jordan, M. I. Is q-learning provably efficient? *Advances in neural information processing systems*, 31, 2018.

Kakade, S., Kearns, M. J., and Langford, J. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 306–312, 2003.

Kapturowski, S., Campos, V., Jiang, R., Rakićević, N., van Hasselt, H., Blundell, C., and Badia, A. P. Human-level atari 200x faster. *arXiv preprint arXiv:2209.07550*, 2022.

Kearns, M. and Singh, S. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2):209–232, 2002.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lobel, S., Gottesman, O., Allen, C., Bagaria, A., and Konidaris, G. Optimistic initialization for exploration in continuous control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7612–7619, 2022.

Machado, M. C., Srinivasan, S., and Bowling, M. Domain-independent optimistic initialization for reinforcement learning. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

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

Machado, M. C., Bellemare, M. G., and Bowling, M. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5125–5133, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control

through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/8d8818c8e140c64c743113f563cf750f-Paper.pdf.

Osband, I., Aslanides, J., and Cassirer, A. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. Count-based exploration with neural density models. In *International conference on machine learning*, pp. 2721–2730. PMLR, 2017.

O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pp. 3836–3845, 2018.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pp. 2778–2787. PMLR, 2017.

Pitis, S., Chan, H., Zhao, S., Stadie, B., and Ba, J. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. *arXiv preprint arXiv:2007.02832*, 2020.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018. URL http://arxiv.org/abs/1802.09464.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. Skew-fit: State-covering self-supervised reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2019.

Raileanu, R. and Rocktäschel, T. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, 2020.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

Rashid, T., Peng, B., Boehmer, W., and Whiteson, S. Optimistic exploration even with a pessimistic initialisation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1xGP6VYwH.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Strehl, A. L. and Littman, M. L. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

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

Sutton, R. S., Bowling, M. H., and Pilarski, P. M. The Alberta plan for AI research. *arXiv preprint arXiv:2208.11173*, 2022.

Taiga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. On bonus based exploration methods in the arcade learning environment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJewlyStDr.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Zhang, S. and Sutton, R. S. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J. E., and Tian, Y. Noveld: A simple yet effective exploration criterion. *Advances in Neural Information Processing Systems*, 34:25217–25230, 2021.

# A. Proofs

## A.1. Any zero-mean unit-variance distribution can be used for counting

Let $z_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ where $x_i \sim \mathcal{X}$. Assume that the distribution $\mathcal{X}$ is such that $\mathbb{E}[\mathcal{X}] = 0$ and $\text{Var}[\mathcal{X}] = 1$. In many cases we make use of the fact that $\mathbb{E}[x_i x_j] = \delta_{ij}$ (Kronecker delta function), because if $i \neq j$, then $\mathbb{E}[x_i x_j] = \mathbb{E}[x_i]\mathbb{E}[x_j] = 0$

$$\begin{aligned}
\mathbb{E}\big[z_n^2\big] &= \mathbb{E}\Big[\Big(\frac{1}{n}\sum_{i=1}^{n} x_i\Big)^2\Big] \\
&= \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j\Big] \\
&= \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n} x_i x_i + \sum_{i=1}^{n}\sum_{j\neq i}^{n} x_i x_j\Big] \\
&= \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n} x_i^2\Big] + \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n}\sum_{j\neq i}^{n} x_i x_j\Big] \\
&= \frac{1}{n^2}(n+0) \\
&= \frac{1}{n}
\end{aligned}$$

This proves the well-known fact that the variance of the sample mean scales inversely with the number of samples.

## A.2. Functional form of the variance of $z_n^2$

We now know that $z_n^2$ is an unbiased estimator of $\frac{1}{n}$, the inverse count. What is the variance of this estimate?

$$\begin{aligned}
\text{Var}\big[z_n^2\big] &= \mathbb{E}\big[z_n^4\big] - \mathbb{E}\big[z_n^2\big]^2 \\
&= \mathbb{E}\Big[(\frac{1}{n}\sum_{i=1}^{n} x_i)^4\Big] - \frac{1}{n^2} \\
&= \frac{1}{n^4}\mathbb{E}\Big[(\sum_{i=1}^{n} x_i)^4\Big] - \frac{1}{n^2} \\
\mathbb{E}\Big[(\sum_{i=1}^{n} x_i)^4\Big] &= \mathbb{E}\Big[\sum_i\sum_j\sum_k\sum_l x_i x_j x_k x_l\Big] \\
&= \mathbb{E}\Big[\sum_{i=j=k=l} x_i^4 + \sum_{i=j,k=l\neq i} x_i^2 x_k^2 + \sum_{i=k,l=j\neq i} x_i^2 x_l^2 + \sum_{i=l,j=k\neq i} x_i^2 x_k^2 + \sum_{\text{remaining}} x_i x_j x_k x_l\Big]
\end{aligned}$$

where here we have broken the summation across all indices $i, j, k, l$ apart into four sets of terms with even exponents, and one set which all have at least one odd exponent (and thus has expectation 0). There are $n$ elements in the first sum, and $n(n-1)$ elements of the second, third, and fourth sum. Thus:

$$\begin{aligned}
\mathbb{E}\Big[\sum_{i=j=k=l} x_i^4\Big] &= n\mathbb{E}\big[\mathcal{X}^4\big] \\
\mathbb{E}\Big[\sum_{i=j,k=l\neq i} x_i^2 x_k^2\Big] &= n(n-1)\mathbb{E}\big[\mathcal{X}^2\big]^2 \\
\mathbb{E}\Big[\sum_{\text{remaining}} x_i x_j x_k x_l\Big] &= 0
\end{aligned}$$

and therefore,

$$\begin{aligned}\mathbb{E}\Big[(\sum_{i=1}^{n} x_i)^4\Big] &= n\mathbb{E}\big[\mathcal{X}^4\big] + 3n(n-1)\mathbb{E}\big[\mathcal{X}^2\big]^2 \\ &= n\mathbb{E}\big[\mathcal{X}^4\big] + 3n^2 - 3n\end{aligned}$$

Plugging into the original equation we arrive at a functional form of $\text{Var}[z_n^2]$:

$$\begin{aligned}\text{Var}\big[z_n^2\big] &= \frac{1}{n^3}\mathbb{E}\big[\mathcal{X}^4\big] + \frac{3}{n^2} - \frac{3}{n^3} - \frac{1}{n^2} \\ &= \frac{1}{n^3}\mathbb{E}\big[\mathcal{X}^4\big] + \frac{2}{n^2} - \frac{3}{n^3}\end{aligned} \tag{7}$$

### A.3. Proof that using the coin-flip distribution yields the lowest-variance estimator of $\frac{1}{n}$

Above, we show that to reduce $\text{Var}\big[z_n^2\big]$ the only knob we can turn is reducing the $4^{th}$ moment, because $\mathbb{E}\big[\mathcal{X}\big] = 0$ and $\mathbb{E}\big[\mathcal{X}^2\big] = 1$ by construction. The $4^{th}$ moment of the coin-flip distribution $x \sim \mathcal{C} \in \{-1, 1\}$ is simply

$$\mathbb{E}[\mathcal{C}^4] = \frac{1}{2}1^4 + \frac{1}{2}(-1)^4 = 1.$$

Plugging this into Equation 7 yields, for $x \sim \mathcal{C}$:

$$\text{Var}\big[z_n^2\big] = \frac{2}{n^2} - \frac{2}{n^3}.$$

This holds for all $n \geq 1$. Therefore, $\text{Var}\big[z_1^2\big] = 0$, which is easy to confirm. Since variance must always be positive, this means that $\mathbb{E}[x^4] \geq 1$ for all $\mathcal{X}$ satisfying our assumptions, and hence using the coin-flip distribution achieves minimum possible variance in its estimation of $\frac{1}{n}$.

### A.4. Proof that variance scales inversely with vector-length

We have another, simpler method for reducing variance: we can increase the number of trials (equivalently, the number of coin flips), and average the results together.

Let $\{z_{ni} | i \in 1, ..., d\}$ be a set of $d$ independent draws of $z_n$. Then

$$\mathbb{E}\Big[\frac{1}{d}\sum_{i=1}^{d} z_{ni}^2\Big] = \frac{1}{d}\sum_{i=1}^{d}\mathbb{E}\big[z_{ni}^2\big] = \mathbb{E}\big[z_n^2\big] = \frac{1}{n}$$

as expected (the average of i.i.d samples is simply the expectation). Similarly:

$$\text{Var}\Big[\frac{1}{d}\sum_{i=1}^{d} z_{ni}^2\Big] = \frac{1}{d^2}\sum_{i=1}^{d}\text{Var}\big[z_{ni}^2\big] = \frac{1}{d}\text{Var}\big[z_n^2\big]$$

This is a well-known fact of how variance scales with samples. Therefore, we can decrease the variance of our estimator in two ways: picking a good distribution (coin-flip is best) and also increasing the number of trials.

## B. Analysis of Linear Coin Flip Network

We now analyze the solution to Equation 4 when our function approximator $f_\phi$ is a linear mapping between states and coin flips. Our goal is to recover an intuitive understanding of how bonuses are estimated when the model has to generalize across inputs. For simplicity, we consider the case of a single coin flip per encountered state. We represent each state $\mathbf{s}$ as a $p$-dimensional vector, with $\mathbf{S}$ being the $n \times p$ matrix of all encountered states , and $c \sim \{\text{-}1, 1\}^n$ being the $n$-dimensional vector of sampled coin flips.

Thus, $f_\phi(\mathbf{s}) = \mathbf{s} \cdot \boldsymbol{\phi}$, where $\boldsymbol{\phi}$ is the weight vector that parameterizes $f_\phi$. Under this formulation, Equation 4 reduces to solving the following linear least-squares regression problem:

$$\boldsymbol{\phi} = \arg\min_{\boldsymbol{\phi'}} \|\mathbf{S}\boldsymbol{\phi'} - \mathbf{c}\|^2.$$

For the following derivation, we assume that $\mathbf{S}$ has rank $p$ (and $n > p$) in order to recover a unique solution, however this result can be generalized by replacing the inverse with the pseudo-inverse. The solution to this linear regression (Golub & Reinsch, 1971) is

$$\boldsymbol{\phi} = (\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T\mathbf{c}.$$

We now rewrite $\mathbf{S}$ using its singular value decomposition (Golub & Reinsch, 1971): $\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^T$, where $\mathbf{U}$ is the $n \times n$ orthonormal "left singular vector" basis, $\mathbf{V}$ is the $p \times p$ orthonormal "right singular vector" basis, and $\boldsymbol{\Lambda}$ is a $n \times p$ rectangular diagonal "singular value" matrix. Thus, $\mathbf{S^T S} = \mathbf{V\Lambda^T U^T U\Lambda V^T} = \mathbf{V\Lambda^T\Lambda V^T}$. Therefore:

$$\begin{aligned}
\boldsymbol{\phi} &= (\mathbf{S^T S})^{-1}\mathbf{S^T c} \\
&= (\mathbf{V\Lambda^T\Lambda V^T})^{-1}\mathbf{V\Lambda^T U^T c} \\
&= \mathbf{V}(\boldsymbol{\Lambda^T\Lambda})^{-1}\mathbf{V^T V\Lambda^T U^T c} \\
&= \mathbf{V}(\boldsymbol{\Lambda^T\Lambda})^{-1}\boldsymbol{\Lambda^T U^T c} \\
&= \mathbf{V\Lambda^{-1}U^T c}
\end{aligned}$$

where $\boldsymbol{\Lambda}^{-1}$ is the pseudo-inverse of $\boldsymbol{\Lambda}$, which in the case of a rectangular diagonal matrix is simply the element-wise reciprocal of the diagonal entries, transposed.

Recall that $f_\phi(s) = \mathbf{s} \cdot \boldsymbol{\phi}$. We can gain more intuition about $f_\phi$ by representing $\mathbf{s}$ using the orthonormal basis $\mathbf{V}$: $\mathbf{s} = \sum_i (\mathbf{s^T} \cdot \mathbf{V_i^T})\mathbf{V_i^T} = \mathbf{pV^T}$ where $\mathbf{p}$ says how much of each basis vector there is in $\mathbf{s}$. Now we can derive a simple formula for the expected inverse-count of $\mathbf{s}$:

$$\begin{aligned}
\mathbb{E}[\frac{1}{\mathcal{N}(\mathbf{s})}] &= \mathbb{E}[\|2f_\phi(\mathbf{S})\|]^2 \\
&= \mathbb{E}[\mathbf{s}\boldsymbol{\phi}\boldsymbol{\phi^T}\mathbf{s^T}] \\
&= \mathbb{E}[\mathbf{pV^T V\Lambda^{-T}U^T cc^T U\Lambda^{-1}V^T Vp^T}] \\
&= \mathbf{pV^T V\Lambda^{-T}U^T}\ \mathbb{E}[\mathbf{cc^T}]\ \mathbf{U\Lambda^{-1}V^T Vp^T}
\end{aligned}$$

Since each element of $\mathbf{c}$ is independent of all others, $\mathbb{E}[\mathbf{cc^T}] = \mathbb{I}$. Furthermore, $\mathbf{U}$ and $\mathbf{V}$ are orthonormal bases and so $\mathbf{U^T U} = \mathbb{I}$ and $\mathbf{V^T V} = \mathbb{I}$. Thus, we get the following simplification:

$$\mathbb{E}[\frac{1}{\mathcal{N}(\mathbf{s})}] = \mathbf{p}(\boldsymbol{\Lambda^T\Lambda})^{-1}\mathbf{p^T}.$$

In other words, when using a linear model, CFN stores in its weights how much of each singular vector is present in the dataset $\mathbf{S}$. When presented with a new state $\mathbf{s}$, it computes an inverse count for each singular vector, and returns a linear combination of those, weighted by how much of that singular vector is present in $\mathbf{s}$.

This matches intuition in the tabular case. If each $s \in S$ is represented by a one-hot vector, then $\mathbf{S^T S}$ is a diagonal matrix that counts how many times each unique state has been seen, and $\mathbf{p}(\boldsymbol{\Lambda^T\Lambda})^{-1}\mathbf{p^T}$ returns the the inverse count of a given state. In the more general linear case, counting simply happens over a different basis than the identity matrix.

## C. Environment details

**Visual Gridworld.** We used a 42x42 gridworld, observations were 84x84 images (each grid-cell was visualized using 2x2 pixels). The player always started the episode at the bottom-left and the goal was at the top-right. Episodes lasted a maximum of 150 steps, unless the agent reached the goal, in which case the episode terminates with a sparse reward of 1. For the stochasticity experiments in Figure 4(right), the maximum number of steps per episode was determined by $\frac{150}{1-\eta}$ where $\eta$ is the action-noise probability. We did this to make sure that more stochastic versions of the task still had long-enough episodes to be solvable by a reasonably good agent: with this scaling the agent has the same number of actions under its own control in a given episode, independent of $\eta$.

**Visual Taxi.** We used two variants of the TAXI—one with a 5x5 grid (as in the default version) and another with a 10x10 grid (Diuk et al., 2008); episodes lasted a maximum of 50 steps in the former and 100 in the latter case. Similar to visual gridworld, the agent observed images of the game state; to show that the passenger was inside the taxi, we shaded the taxi differently and included a black border around the image.

**Fetch and Adroit Manipulation.** As mentioned in the main paper, we first sparsified the reward function—this means that there are no shaping rewards for reaching the object or for moving it to non-goal locations. To set the goal locations for the non-default versions of the tasks, we first visualized the environment and rendered the effect of random actions. The exact goal locations that we settled on can be found in our linked code (file `wrappers.py`). In the RELOCATE task, we truncate the episode when the ball leaves the table.

**Ant U-Maze.** D4RL randomly sets the goal location to a small distribution around $(x = 0, y = 8)$ and the ant's location to a small distribution around $(x = 0, y = 0)$. Predictably, we used the sparse-reward version of the task. Episodes last a maximum of 1000 steps.

**MONTEZUMA'S REVENGE** We followed the experimental protocol of Machado et al. (2015) which means that we used sticky actions, a frame stack of 4, action repeat of 4, grayscale images of shape 84x84 and a training budget of 200 million frames (50 million agent-environment interactions).

## D. Hyperparameters, Architecture and Training Details

The final values of hyperparameters for all experiments have been set as the default values in our configuration files, which can be found in the `configs/*.gin` files in our codebase. The file `intrinsic_motivation/intrinsic_rewards.py` contains architectural details such as number of layers and layer sizes, and unless otherwise noted are the same as presented in (Taiga et al., 2020). The neural network architecture of CFN is chosen to match that of RND's prediction network.

All hyperparameters *not* listed are chosen to match the default Rainbow (Hessel et al., 2018) and SAC (Haarnoja et al., 2018) implementations. For each task group, the tested hyperparameters are listed; multiple value indicate a grid search, with the chosen value listed in bold. On Montezuma's Revenge we use the best reported RND and PixelCNN hyperparameters from (Taiga et al., 2020).

### D.1. Shared Hyperparameters

We used Rainbow for Visual Gridworld (Section 4.3) and Montezuma's Revenge (Section 4.5) and SAC for all the continuous control experiments (Section 4.4). The hyperparameters for these base agents are reported below:

| Rainbow Hyperparameter | Gridworld | Atari |
|---|---|---|
| Discount Factor $\gamma$ | 0.99 | 0.99 |
| Adam LR | **1.25e-4**, 1e-5 | **1.25e-4**, 6.25e-5 |
| Adam $\epsilon$ | 1.5e-4 | 1.5e-4 |
| Multi-step return $n$ | 3 | 3 |
| Min history to start learning | 1,000 | 20,000 |
| Distributional Atoms | 51 | 51 |
| Distributional Min/Max values | $\pm 10$ | $\pm 10$ |
| Batch Size | 32 | 32 |

| SAC Hyperparameter | Fetch | Adroit | Ant |
|---|---|---|---|
| Discount Factor $\gamma$ | 0.99 | 0.99 | 0.99 |
| Adam LR | 3e-4 | **1e-4** 3e-4 | 3e-4 |
| Adam $\epsilon$ | 1e-8 | 1e-8 | 1e-8 |
| Multi-step return $n$ | 3 | 3 | 3 |
| Min history to start learning | 10,000 | 10,000 | 10,000 |
| SAC Reward Scale Factor | 1.0 | 1.0, 3.0, 10.0, **30.0** | 0.1 |
| Batch Size | 256 | 256 | 256 |

### D.2. CFN Hyperparameters

| Hyperparameter | Gridworld | Fetch | Adroit | Ant | Atari |
|---|---|---|---|---|---|
| Intrinsic Reward Scale | 0.001, 0.003 **0.01**, 0.03 | **0.001**, 0.003 0.01, 0.03 | **0.001**, 0.003 0.01, 0.03 | 0.001, 0.003 **0.01**, 0.03 | 0.001, 0.003 **0.01**, 0.03 |
| Reward Normalization* | **Yes**, No | No | No | No | No |
| CFN learning rate | 1e-4 | 1e-4 | 1e-4 | 1e-3 | 1e-4 **1e-5** |
| CFN replay buffer size | 1e6 | 1e6 | 1e6 | 1e6 | 2e7 |
| CFN batch size | 1024 | 1024 | 1024 | 1024 | 512 |
| CFN update period | 1 | 1 | 1 | 1 | 4 |
| Number of coin flips $d$ | 20 | 20 | 20 | 20 | 20 |

* **Reward normalization.** In Visual Gridworld, we found it helpful to use reward normalization (Burda et al., 2019), which normalizes the exploration bonus by subtracting its running mean and dividing by its running variance.

### D.3. CFN replay buffer size

Note that we used a much larger CFN replay buffer for Montezuma's Revenge. Usually, a small FIFO queue is used to implement the replay buffer. As shown by Zhang & Sutton (2017), larger replay buffers hurt RL performance because it

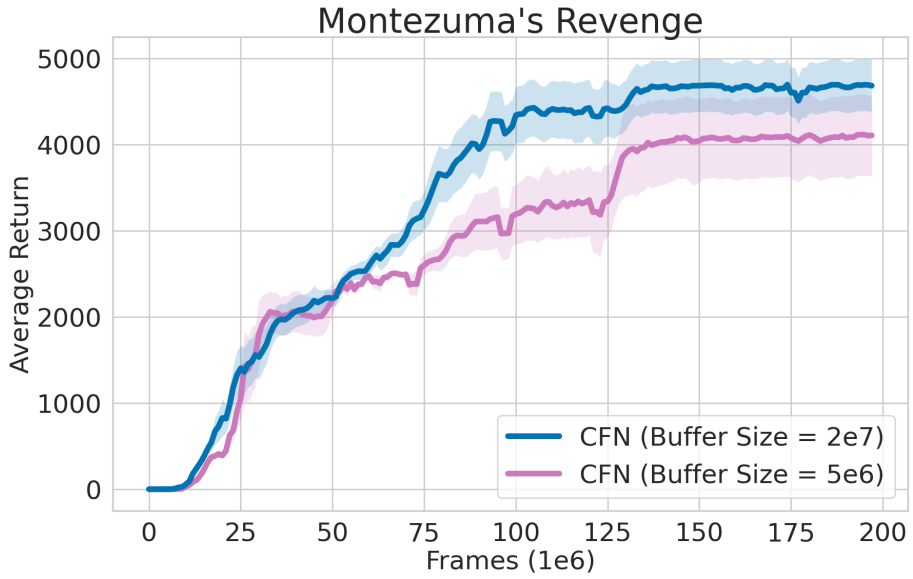


*Figure 8.* Analyzing the impact of large replay buffer for CFN: while a very large buffer size of $2e7$ leads to better performance, a smaller buffer size of $5e6$ still performs well. Results are averaged over 6 random seeds.

makes the Q-learning updates more off-policy. Since the CFN objective is a standard regression problem (and does not use bootstrap targets), larger buffers almost always improve performance. Furthermore, shorter CFN buffers often cause high-count states to be removed from the replay buffer and eventually re-appear as novel to the agent; a problem that is mitigated with larger buffer sizes. In future work, we would like to revert to smaller replay buffers by implementing a "forgetting" strategy in which low novelty states are discarded from replay with higher probability.

Figure 8 shows that while a very large replay buffer does indeed yield better performance in Montezuma's Revenge, a moderately sized replay buffer also performs respectably.

### D.4. PixelCNN Hyperparameters

| Hyperparameter | Gridworld | Atari |
|---|---|---|
| Intrinsic Reward Scale | 0.1, **0.5**, 1.0 | 0.1 |
| Prediction Gain Scale | 0.1, **0.5**, 1.0, 5.0 | 1.0 |
| PixelCNN learning rate | 1e-4 | 1e-4 |

### D.5. RND Hyperparameters

| Hyperparameter | Gridworld | Fetch | Adroit | Ant | Atari |
|---|---|---|---|---|---|
| Intrinsic Reward Scale | 5e-5,**1e-4**, 5e-4,1e-3 | **5e-5**,1e-4, 5e-4,1e-3 | **5e-5**,1e-4, 5e-4,1e-3 | **5e-5**,1e-4, 5e-4,1e-3 | 5e-5 |
| RND learning rate | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |

### D.6. Compute Resources

The wallclock time of all intrinsic reward methods is roughly similar, however on Atari domains CFN requires significantly more memory—see Section D.2 for discussion. All experiments are performed on a SLURM cluster using nodes equipped with 4 CPUs and 1 3090-Ti NVIDIA GPU. We reserve 16GB of memory for all experiments except for CFN's Montezuma's Revenge, where we reserve 160GB.

# E. Additional Experiments

## E.1. Counts on Visual Taxi

We report count reconstruction-accuracy on visual versions of both the 5x5 and the 10x10 TAXI environments in Figure 9. See Appendix C for environment details. These tasks are visually more complex and have many more possible states than Visual Gridworld. Since the policy used to collect data is a confounding variable when comparing counting accuracy, in these experiments all interaction is performed with a random policy, and only the bonus modules are trained. For both domains, we present bonuses computed after $200,000$ interactions, with each method taking one training step per interaction. In this time, the random policy visits approximately $900$ unique states in the 5x5 domain, and approximately $6,500$ unique states in the 10x10 domain. We note a similar trend as in Gridworld, where CFN procudes more accurate bonuses than PixelCNN and RND.

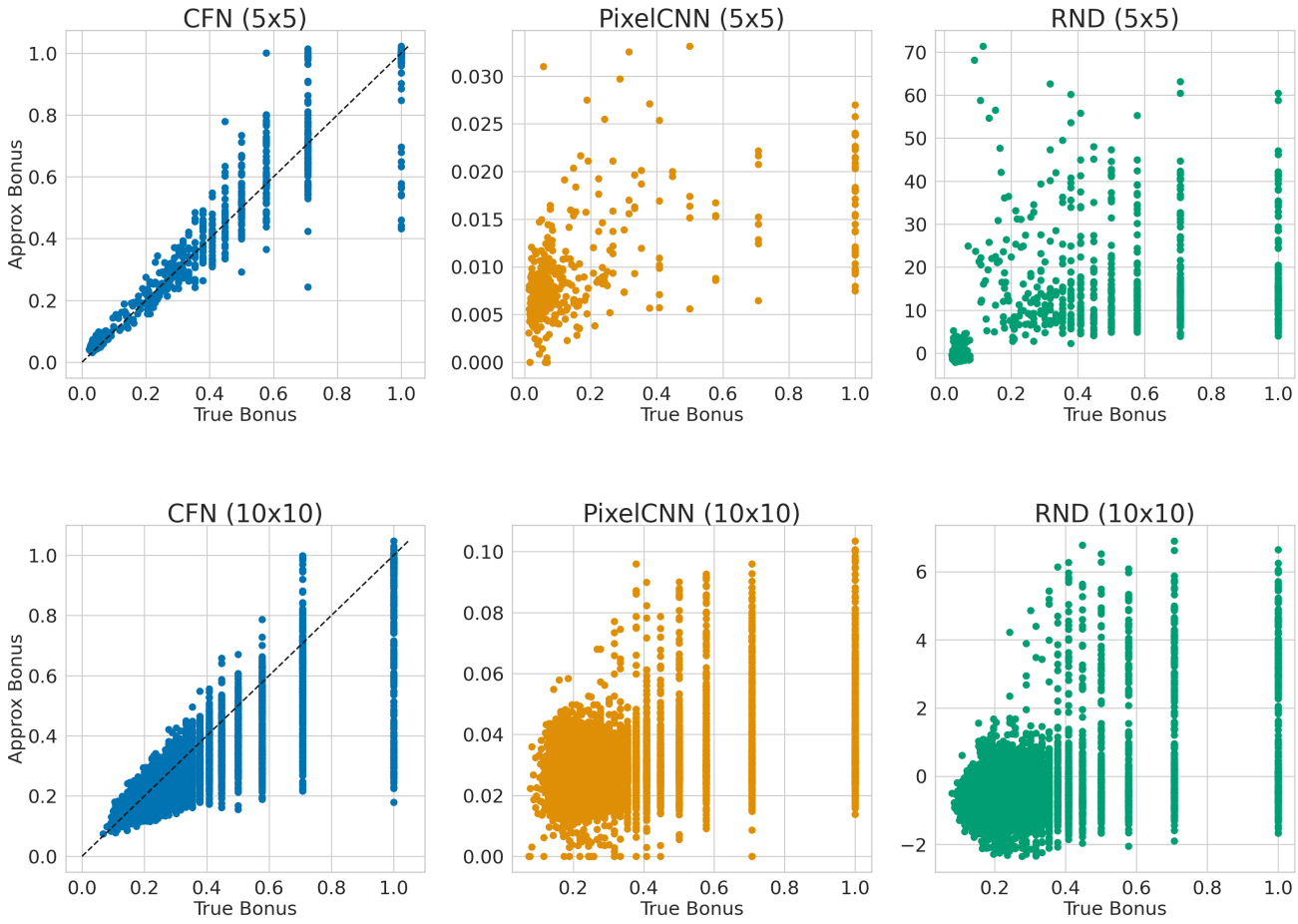


*Figure 9.* Bonuses for CFN, PixelCNN, and RND on both Taxi domains.

### E.2. Effect of Random Prior without Coin Flips

Can the accuracy of the predicted exploration bonuses in Figure 2 (main paper) solely be attributed to the use of the random prior (discussed in Section 3.5.2)? To answer this question, we devised an experiment in which we removed the randomness introduced by the coin-flip vectors by setting them to $\{0\}^d$ instead of sampling them from $\{-1, 1\}^d$.

Figure 10 shows the result of this experiment: states seen for the first time get a high exploration bonus, implying that the random prior initializes their pseudocount near 1 (as intended). On the other hand, states visited more than once get almost no exploration bonus; this highlights the importance of the full CFN objective to get an exploration bonus that falls off smoothly as $1/\sqrt{\mathcal{N}(s)}$.

Furthermore, notice that states with true bonus of 1 form two distinct clusters—with high and low predicted bonus respectively. We posit that states in the lower bonus cluster were encountered less recently and thus have been sampled by CFN, which has wiped out the effect of their optimistic initialization.

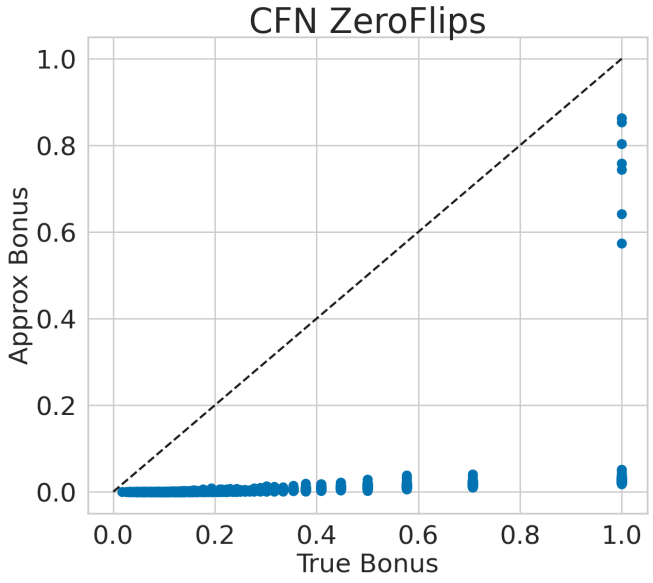


*Figure 10.* CFN bonus prediction versus ground-truth bonus on the 42x42 Visual Gridworld domain using $c \sim \{0\}^d$.

### E.3. Prioritization and Random Prior ablation count plots

Figure 11 shows the importance of prioritized sampling and optimistic bonus initialization. Using both methods results in the most accurate bonus predictions across the entire range of novelty. When only random prior is used, CFN does not train as frequently on novel states, and thus underestimates their count. When only prioritization is used, a novel state may be first inserted into the dataset with low novelty (and thus low priority); so, the state may not be sampled for training, which retains its underestimate of novelty. When neither prioritization nor random prior are used, bonuses are still accurate for states observed more than 5 times, but are inaccurate for very novel states.

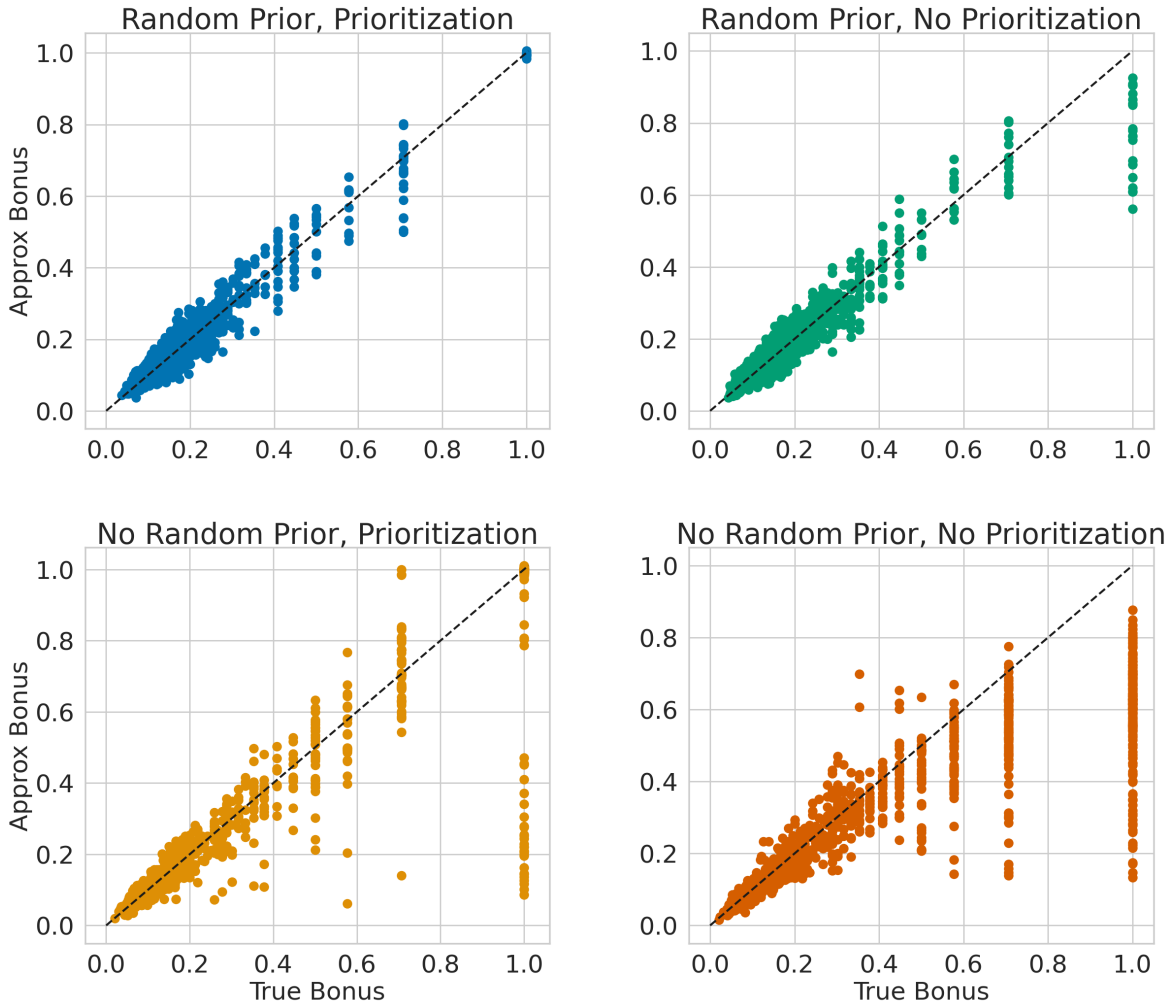


*Figure 11.* True versus approximate bonus of including/excluding random prior and prioritization.