A Broader Impact Statement

This work makes progress towards better understanding and designing methods which can efficiently explore contextual MDPs, a very broad framework with applications in video games, virtual reality, autonomous driving, robotics and healthcare. Efficient exploration typically reduces sample complexity, which make real-world application more feasible. Like any RL algorithm, our approach aims to facilitate discovering a policy that maximizes some user-specified reward. Depending on the reward function, executing such a policy could have positive or negative consequences.

B Reproducibility Statement

All our code will be made public upon acceptance.

C Additional Related Work

Exploration in singleton MDPs is a well-studied problem in RL (Sutton & Barto, 2018; Schmidhuber, 1991; Oudeyer et al., 2007; Oudeyer & Kaplan, 2009). Many theoretical works exist which propose provably efficient algorithms for tabular or linear MDPs (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002; Strehl & Littman, 2006; Jin et al., 2020; Cai et al., 2020; Agarwal et al., 2020; Kolter & Ng, 2009; Fruit & Lazaric, 2017; Fruit et al., 2018a;b; Tarbouriech et al., 2020). A number of methods which combine deep RL agents with exploration bonuses have also been proposed for general MDPs (Stadie et al., 2015; Achiam & Sastry, 2017). These include model-free methods such as RND (Burda et al., 2019), ICM (Pathak et al., 2017) and pseudocounts (Bellemare et al., 2016b; Strehl & Littman, 2008; Bellemare et al., 2016a; Ostrovski et al., 2017; Martin et al., 2017; Tang et al., 2017; Machado et al., 2020), as well as model-based approaches (Shyam et al., 2019; Henaff, 2019; Sekar et al., 2020; Zhang et al., 2021a; Manek & Kolter, 2021). However, these are all designed for the singleton MDP setting and use some form of global bonus which, as we show in Section 3, is not always appropriate to the more general CMDP setting we consider here. We also note the work of (Stanton & Clune, 2018) which used episodic bonuses for singleton MDPs.

D Experiment Details for MiniHack

We used the same architectures and hyperparameters for the experiments with count-based bonuses in Section 3 and with function approximation in Section 4.

D.1 Architecture Details

We follow the policy network architecture described in Samvelyan et al. (2021). The policy network has four trunks: i) a 5-layer convolutional trunk which maps the full symbol image (of size 79×21) to a hidden representation, ii) a second 5-layer convolutional trunk which maps a 9×9 crop centered at the agent to a hidden representation, iii) an MLP trunk which maps the stats vector to a hidden representation, and iv) a 1-D convolutional trunk with interleaved max-pooling layers, followed by a fully-connected network which maps the message to a hidden representation. The hidden representations are then concatenated together, passed through a 2-layer fully-connected network followed by an LSTM Hochreiter & Schmidhuber (1997) layer. The output of the LSTM layer is then passed to linear layers which produce action probabilities and a value function estimate.

The convolutional trunks i) and ii) have the following hyperparameters: 5 layers, filter size 3, symbol embedding dimension 64, stride 1, filter number 16 at each layer except the last, which is 8, and ELU non-linearities Clevert et al. (2016). The MLP trunk iii) has 2 hidden layers of 64 hidden units each with ReLU non-linearities. The trunk iv) for processing messages has 6 convolutional layers, each with 64 input and output feature maps. The first two have kernel size 7 and the rest have kernel size 3. All have stride 1 and there are max-pooling layers (kernel size 3, stride 3) after the 1st, 2nd and 6th convolutional layers. The last two layers are fully-connected and have 128 hidden units and ReLU non-linearities.

For E3B, we used the same architecture as the policy encoder for the feature embedding ϕ, except we removed the last layers mapping the hidden representation to the actions and value estimate. The

inverse dynamics model is a single-layer fully-connected network with 256 hidden units, mapping two concatenated ϕ outputs to a softmax distribution over actions.

For RND and NovelD, we also used the same architecture as above for the target and predictor networks. For AGAC, we used the same network as the policy for the adversary.

D.2 RL HYPERPARAMETERS

For all algorithms we use IMPALA Espeholt et al. (2018) as our base policy optimizer. Hyperparameters which are common to all methods are shown in Table 1. All algorithms were trained for 50 million environment steps. We did not anneal learning rates for any of the methods during training.

Hyperparameters specific to the E3B, RND, NovelD and AGAC components are shown in Tables 2, 3 and 4. For all algorithms using an exploration bonus, we used a rolling normalization of the intrinsic reward similar to that proposed in the RND paper Burda et al. (2019). Specifically, we maintained a running standard deviation σ of the intrinsic rewards and divided the intrinsic rewards by σ before feeding them to the policy optimizer. For E3B and NovelD, we set the hyperparameters to the values reported in (Henaff et al., 2022). For AGAC, we set the adversary learning rate to be $0.3\times$ the policy learning rate as was done in the official code release. We used the same coefficient for the adversary loss (0.00004)–we also experimented with higher values of the adversary loss, but these performed less well.

Table 1: Common IMPALA Hyperparameters for MiniHack

Learning Rate	0.0001
RMSProp smoothing constant	0.99
RMSProp momentum	0
RMSProp ϵ	10^{-5}
Unroll Length	80
Number of buffers	80
Number of learner threads	4
Number of actor threads	256
Max gradient norm	40
Entropy Cost	0.005
Baseline Cost	0.5
Discounting Factor	0.99

Table 2: E3B Hyperparameters

Ridge λ	0.1
Intrinsic Reward Normalization	True
Intrinsic Reward Coefficient	1.0

Table 3: RND Hyperparameters

Predictor Learning Rate	0.0001
Intrinsic Reward Normalization	True
Intrinsic Reward Coefficient	1.0

For the experiments in Section 4.2, we tuned the β hyperparameter over the range $\{1, 10, 100, 1000, 10000, 100000\}$. In initial experiments we noticed that the episodic bonus was several orders of magnitude smaller than the episodic bonus, hence we used a high range of values for the β hyperparameter to bring the global bonus to a similar range.

Table 4: NovelD Hyperparameters

Predictor Learning Rate	0.0001
Scaling Factor c	0.1
Intrinsic Reward Normalization	True
Intrinsic Reward Coefficient	1.0

Table 5: AGAC Hyperparameters

Adversary Learning Rate	0.00003
Adversary loss term	0.00004
Intrinsic Reward Normalization	True
Intrinsic Reward Coefficient	1.0

D.3 Task Details

We used the following set of 18 MiniHack tasks: `'MiniHack-KeyRoom-S5-v0'`, `'MiniHack-MultiRoom-N4-Locked-v0'`, `'MiniHack-MultiRoom-N6-Lava-v0'`, `'MiniHack-MultiRoom-N6-Lava-OpenDoor-v0'`, `'MiniHack-MultiRoom-N6-LavaMonsters-v0'`, `'MiniHack-MultiRoom-N10-v0'`, `'MiniHack-MultiRoom-N10-OpenDoor-v0'`, `'MiniHack-MultiRoom-N10-Lava-v0'`, `'MiniHack-MultiRoom-N10-Lava-OpenDoor-v0'`, `'MiniHack-LavaCrossingS19N13-v0'`, `'MiniHack-LavaCrossingS19N17-v0'`, `'MiniHack-Labyrinth-Big-v0'`, `'MiniHack-Levitate-Potion-v1'`, `'MiniHack-Levitate-Boots-v1'`, `'MiniHack-Freeze-Horn-v1'`, `'MiniHack-Freeze-Wand-v1'`, `'MiniHack-Freeze-Random-v1'`.

Note that the `-v1` versions of the tasks have restricted action spaces, as described in (Henaff et al., 2022).

D.4 Experiment Details for Habitat

D.4.1 Environment Details

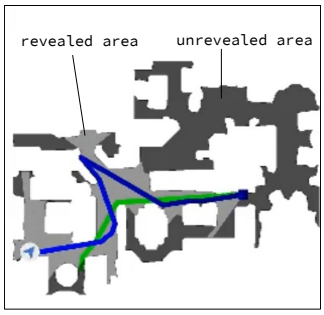
We used the HM3D Ramakrishnan et al. (2021) dataset, which consists of 1000 high-quality renderings of indoor scenes. Observations consist of 4 modalities: an RGB and depth image (shown in Figure 9a), GPS coordinates and the compass heading. The action space consists of 4 actions: $\mathcal{A} = \{$`stop_episode`, `move_forward (0.25m)`, `turn_left (10°)`, `turn_right (10°)`$\}$. The dataset scenes are split into $800/100/100$ train/validation/test splits. Since the test split is not publicly available, we evaluate all models on the validation split. Each scene corresponds to a different context $c \in \mathcal{C}$ in the CMDP framework.

To measure exploration coverage, we compute the area revealed by the agent's line of site using the function provided by the Habitat codebase [2], which uses a modified version of Bresenham's line cover algorithm. We define the exploration coverage to be:

$$\text{coverage} = \frac{\text{revealed area}}{\text{total area}}$$

See Figure 9b) for an illustration. For the results in Figure 3, we evaluated exploration performance for each algorithm by measuring its coverage on 100 episodes using scenes from the validation set (which were not used for training).

(a)



(b)

Figure 9: a) Visual observations in Habitat b) Exploration is measured as the proportion of the environment revealed by the agent's line of sight over the course of the episode.

[2] https://github.com/facebookresearch/habitat-lab/blob/main/habitat/utils/visualizations/fog_of_war.py

D.4.2 ARCHITECTURE DETAILS

For all Habitat experiments we used the same policy network as in Wijmans et al. (2020), which includes a ResNet50 visual encoder He et al. (2015) and a 2-layer LSTM Hochreiter & Schmidhuber (1997) policy. In addition to RGB and Depth images, the agent also receives GPS coordinates and compass orientation, represented by 3 scalars total, which are fed into the policy. See the official code release at https://github.com/facebookresearch/habitat-lab/tree/main/habitat_baselines for full details.

For exploration algorithms which use inverse dynamics models (and ICM), we set the architecture of the encoder ϕ to be identical to that of the policy network, except that the last layer mapping hidden units to actions is removed. The inverse dynamics model was a single layer MLP with 256 hidden units and ReLU non-linearities.

For exploration algorithms which use random network distillation (RND and NovelD), we set the architecture of the random network to be identical to that of the policy network.

D.4.3 RL HYPERPARAMETERS

The DD-PPO hyperparameters which are common to all the algorithms are listed in Table 6. The hyperparameters which are specific to each algorithm are listed in Table 7, 8, 4. For NovelD's count-based bonus, hashing the full image was too slow to be practical, so we subsampled images by a factor of 1000 used that for the count-based bonus, along with the GPS coordinates and compass direction.

Table 6: Common PPO/DD-PPO Hyperparameters for Habitat

Clipping	0.2
PPO epochs	2
Number of minibatches	2
Value loss coefficient	0.5
Entropy coefficient	0.00005
Learning rate	0.00025
ϵ	10^{-5}
Max gradient norm	0.2
Rollout steps	128
Use GAE	True
γ	0.99
τ	0.95
Use linear clip decay	False
Use linear LR decay	False
Use normalized advantage	False
Hidden size	512
DD-PPO Sync fraction	0.6

D.4.4 COMPUTE DETAILS

Each job was run for 225 million steps, which took approximately 3 days on 32 GPUs with 10 CPU threads.

Table 7: Hyperparameters for E3B on Habitat

Hyperparameter	Values considered	Final Value
Ridge regularizer λ	$\{0.1\}$	0.1
Intrinsic reward coefficient β	$\{1.0, 0.1, 0.01, 0.001, 0.0001$	0.1
Inverse Dynamics Model updates per PPO epoch	3	3

Table 8: Hyperparameters for RND on Habitat

Hyperparameter	Values considered	Final Value
Intrinsic reward coefficient β	$\{1.0, 0.1, 0.01, 0.001, 0.0001$	0.1
Predictor Model updates per PPO epoch	3	3

Table 9: Hyperparameters for ICM on Habitat

Hyperparameter	Values considered	Final Value
Intrinsic reward coefficient β	$\{1.0, 0.1, 0.01, 0.001, 0.0001$	0.1
Forward Dynamics Model loss coefficient	$\{1.0\}$	1.0

E ADDITIONAL EXPERIMENT RESULTS

In this section we show the evolution of the global and episodic bonus terms for the E3B×NovelD algorithm for four of the MiniHack tasks (see Figure 10). The first row displays the inverse dynamics model loss used for learning the ϕ embedding in E3B's episodic bonus, the second row shows the E3B episodic bonus itself, the third shows the NovelD global bonus, and the fourth shows the true extrinsic reward provided by the environment.

First, note that the global bonus spans a much larger range of values than the episodic bonus does, initially starting at a high value, exhibiting a first rapid decay, and then further decaying at a slower rate. In contrast, the episodic bonus spans a more limited range, and during most of the training it has higher magnitude than the global bonus. For the episodic bonus, we first see an initial decrease in magnitude, which is likely due to the ϕ features stabilizing (note that this coincides with the stabilization of the inverse dynamics loss used for learning ϕ). After this, the episodic bonus increases, indicating that the agent's policy is learning to maximize the episodic bonus. The episodic bonus then decreases again once the extrinsic reward starts to increase, indicating that the agent has found the true environment reward and is switching to exploitation rather than exploration.

We hypothesize that additively combining the episodic and global bonuses does not offer much benefit over the episodic bonus alone because the magnitude of the global bonus decays significantly over time. Recall that the global bonus is computed using *all* the agent's experience, so it eventually becomes exhausted. Since the episodic bonus is reset each episode, it does not become exhausted the same way, as evidenced by the fact that it *increases* once the feature encoder ϕ has stabilized. If we add the two together, eventually the contribution of the global bonus will be small compared to the contribution of the episodic bonus. However, if we combine the two multiplicatively the global bonus will still have an effect regardless of its scale.

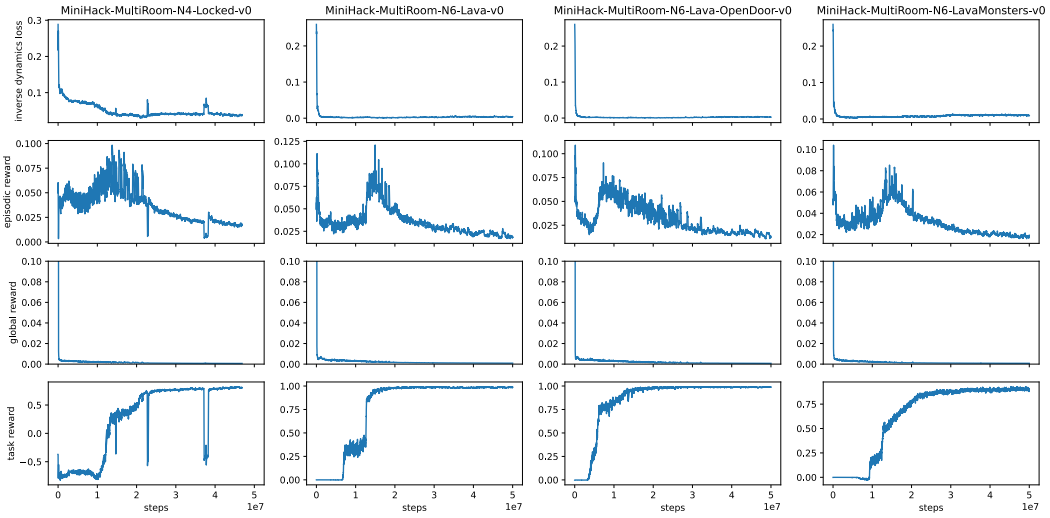


Figure 10: Evolution of inverse dynamics loss, episodic bonus, global bonus and extrinsic reward throughout training for E3B×NovelD.