A Broader Impact Statement

This work proposes an RL exploration method for contextual MDPs, which is a very broad framework. Many decision-making problems can be framed as contextual MDPs, such as autonomous driving (contexts represent cars/roads), household robotics (contexts represent houses), healthcare applications (contexts represent patients) and online recommendation/ad optimization (contexts represent customers). Like other RL exploration algorithms, our method facilitates learning a policy which maximizes some reward function specified by a designer. Depending on the goals of the reward function designer, executing the resulting policy could result in positive or negative consequences.

B Algorithm Details

Algorithm 2 Exploration via Episodic Elliptical Bonuses (E3B)

```
Initialize policy $\pi$, feature encoder $\phi$ and inverse dynamics model $f$.
while not converged do
    Sample context $c \sim \mu_C$ and initial state $s_0 \sim \mu_S(\cdot|c)$
    Initialize inverse covariance matrix: $C_0^{-1} = \frac{1}{\lambda} I$
    for $t = 0, ..., T$ do
        $a_t \sim \pi(\cdot|s_t)$                                          // Sample action
        $s_{t+1}, r_{t+1} \sim P(\cdot|s_t, a_t)$                          // Step through environment
        $b_{t+1} = \phi(s_{t+1})^\top C_t^{-1} \phi(s_{t+1})$              // Compute bonus
        $u = C_t^{-1} \phi(s_{t+1})$
        $C_{t+1}^{-1} = C_t^{-1} - \frac{1}{1+b_{t+1}} u u^\top$           // Update inverse covariance matrix
        $\bar{r}_{t+1} = r_{t+1} + \beta b_{t+1}$
    end for
    Perform policy gradient update on $\pi$ using rewards $\bar{r}_1, ..., \bar{r}_T$.
    Update $\phi$ and $g$ using $\{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$ to minimize the loss:
        $$\ell = -\log(p(a_t | f(\phi(s_t), \phi(s_{t+1}))))$$
end while
```

The full algorithm details are shown above.

Additional intuition: The elliptical bonus is related to the Mahalanobis distance [44] which uses a similar bilinear form. However, the Mahalanobis distance would normalize the matrix C_{t-1} in equation 4.1 by the number of observations $t-1$ in the episode, whereas the elliptical bonus does not.The elliptical bonus thus tends to decrease with the number of observations, similarly to the count-based bonus.

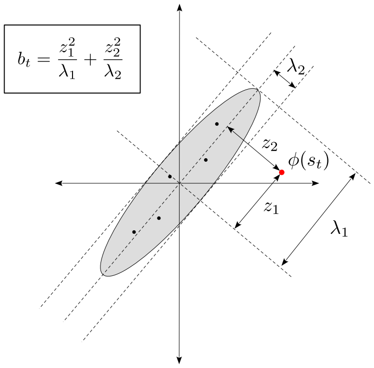


Figure 10: Illustration of the elliptical bonus in 2 dimensions.

C Experiment Details

C.1 MiniHack

C.1.1 Architecture Details

We follow the policy network architecture described in [58]. 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 [32] 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 [16]. 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.

C.1.2 RL Hyperparameters

For all algorithms we use IMPALA [23] as our base policy optimizer. Hyperparameters which are common to all methods are shown in Table 2. All algorithms were trained for 50 million environment steps. We did not anneal learning rates for any of the methods during training, since we found this yielded similar or better performance and simplified the setup.

Hyperparameters specific to E3B, NovelD, RIDE and ICM are shown in Tables 3, 4, 5 and 6. For both E3B and NovelD, we experimented with a rolling normalization of the intrinsic reward similar to that proposed in the RND paper [13]. 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. We found that this led to improved aggregate performance across environments for E3B and NovelD (the improvement for NovelD was relatively minor). We found that tuning the intrinsic reward coefficient was important for best performance for all methods, as well as the regularization coefficient of the C_t matrix λ for E3B. For RIDE, we used the default hyperparameters from the RIDE implementation in the MiniHack paper [58] and otherwise tuned the intrinsic reward coefficient. For ICM, we used the same hyperparameters for the forward and inverse models as for RIDE and also tuned the intrinsic reward coefficient.

Table 2: 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	0.0005
Baseline Cost	0.5
Discounting Factor	0.99

Table 3: Hyperparameters for E3B

Hyperparameter	Values considered	Final Value
Running intrinsic reward normalization	{`True, False`}	`True`
Ridge regularizer λ	$\{1.0, 0.1, 0.01\}$	0.1
Entropy Cost	$\{0.0005, 0.005\}$	0.005
Intrinsic reward coefficient β	$\{0.0001, 0.001, 0.01, 0.1, 1, 10\}$	1

Table 4: Hyperparameters for NovelD

Hyperparameter	Values considered	Final Value
Running intrinsic reward normalization	{`True, False`}	`True`
Scaling factor α	$\{0.1, 0.5\}$	0.1
Entropy Cost	$\{0.0005, 0.005\}$	0.005
Intrinsic reward coefficient β	$\{0.001, 0.01, 0.1, 1, 10, 100\}$	1

Table 5: Hyperparameters for RIDE

Hyperparameter	Values considered	Final Value
Forward Model loss coefficient	1.0	1.0
Inverse Model loss coefficient	0.1	0.1
Entropy Cost	$\{0.0005, 0.005\}$	0.0005
Intrinsic reward coefficient β	$\{0.001, 0.01, 0.1, 1, 10, 100\}$	0.1

Table 6: Hyperparameters for ICM

Hyperparameter	Values considered	Final Value
Forward Model loss coefficient	1.0	1.0
Inverse Model loss coefficient	0.1	0.1
Entropy Cost	$\{0.0005, 0.005\}$	0.0005
Intrinsic reward coefficient β	$\{0.001, 0.01, 0.1, 1, 10, 100\}$	0.1

C.1.3 Environment Details

We used 16 MiniHack environments in total. These include the following 9 navigation-based tasks:

'MiniHack-MultiRoom-N4-Locked-v0', 'MiniHack-MultiRoom-N6-Lava-v0', 'MiniHack-MultiRoom-N6-Lava-OpenDoor-v0', 'MiniHack-MultiRoom-N6-LavaMonsters-v0', 'MiniHack-MultiRoom-N10-OpenDoor-v0', 'MiniHack-MultiRoom-N10-Lava-OpenDoor-v0', 'MiniHack-LavaCrossingS19N13-v0', 'MiniHack-LavaCrossingS19N17-v0', 'MiniHack-Labyrinth-Big-v0'

as well as the following 7 skill-based tasks:

'MiniHack-Levitate-Potion-Restricted-v0', 'MiniHack-Levitate-Boots-Restricted-v0', 'MiniHack-Freeze-Horn-Restricted-v0', 'MiniHack-Freeze-Wand-Restricted-v0', 'MiniHack-Freeze-Random-Restricted-v0', 'MiniHack-LavaCross-Restricted-v0', 'MiniHack-WoD-Hard-Restricted-v0'

The MultiRoom-N4-Locked and MultiRoom-N*-Lava, Labyrinth-Big, LavaCrossingS*N*, as well as all the skill-based tasks, are taken from the official MiniHack github repository (https://github.com/facebookresearch/minihack). We made some of these harder by increasing the number of rooms.

We also include some new environments which we designed to better test the limits of the different algorithms. Specifically, 'MiniHack-MultiRoom-N*OpenDoor-v0' are variants on the standard MultiRoom tasks, the only difference being that the doors connecting the rooms are initialized to be open rather than closed. This is because opening a door causes a message to appear "the door opens". By initializing the door to be closed, the agent does not see any messages when passing from one room to the next. As discussed in Appendix D, this seemingly trivial change can cause the NovelD-message variant to fail completely, shedding light on its lack of robustness.

We found that the MiniHack-MultiRoom-N6-Extreme task was impossible to solve consistently even for a human player due to the large quantities of monsters, so we did not include it. We instead included a variant with less monsters and lava called MultiRoom-N6-LavaMonsters.

C.1.4 Action Space Restriction

In their default setup, the skill-based tasks have a large context-dependent action space of 78 actions. Many of these actions are unrelated to the task at hand, but produce unique in-game messages when executed which nevertheless do not affect the underlying state of the MDP. For example, trying to pay a non-existent shopkeeper by executing the PAY action results in the message "There appears to be no shopkeeper here to receive your payment.", and executing the FIGHT action in the absence of adversaries results in the message "You attack thin air." We found that neither the baselines nor our proposed method were able to solve the skill-based tasks with the full action space. For the NovelD variants which use positions or symbolic images for the episodic counts, this is reasonable since the tasks are not navigation-based (these methods did not work even with restricted action spaces). For NovelD with the message-based bonus, we found that the agent ends up learning a policy where it executes many different actions which produce different messages, but do not change the underlying state of the game (such as the PAY and FIGHT actions described above). This makes sense from the perspective of optimizing intrinsic reward, since each new message seen within the episode provides additional intrinsic reward; however, this does not help explore the state space in a way that is helpful for discovering the true environment reward. We observed similar behavior for E3B, and hypothesize that the encoding learned through the inverse dynamics model keeps message information since this is useful for predicting actions, even if they have no real effect (such as PAY and FIGHT). In this case, the policy can then maximize intrinsic reward by executing these actions.

To avoid this unwanted behavior (which applies to all the methods we tested), we restricted the action space to only the actions which were useful for the tasks at hand. These included actions corresponding to direction movements, as well as different combinations of PICKUP, QUAFF, ZAP, FIRE, and WEAR (full details can be found in our code release). We denote the versions of the tasks with action space restriction with the "-Restricted-v0" suffix, as opposed to the original "-v0" suffix. We believe that better understanding this failure mode and designing exploration bonuses and/or representation learning methods which are robust to it is an important direction for future work.

C.1.5 Compute Details

Each algorithm was trained using 40 Intel(R) Xeon(R) CPU cores (E5-2698 v4 @ 2.20GHz) and one NVIDIA GP100 GPU. We used PyTorch [51] for all our experiments. Each run took between approximately 10 and 30 hours to complete. The total runtime depended on two factors: the computation time of the algorithm, and the behavior of the policy. In terms of algorithm computation time, E3B was roughly 1.5 to 2 times slower than the other methods, due to the fact that in our implementation an additional forward pass through the embedding network (used to compute the elliptical bonus) was performed on CPU. It is possible that a different implementation where this step would be done on GPU would be faster.

A second factor which influenced the total runtime was how quickly the agent learned to avoid dying. For certain environments (for example, those which contain lava), the agent can die quickly which causes the environment to be regenerated. For environments which call the MiniGrid library on the backend, this can be a performance bottleneck since generating new MiniGrid environments can be slow. We found that algorithms which cause the agent to die frequently were much slower on the `MiniHack-MultiRoom-N*Lava*` and `MiniHack-LavaCrossing*` environments, both of which are based on MiniGrid.

C.2 VizDoom

C.2.1 Architecture Details

For all VizDoom experiments, we used the same policy network architecture as in [56]: four 2D convolutional layers with 32 input and 32 output channels, kernel size 3×3, stride 2 and padding 1, interleaved with Exponential Linear Unit (ELU) non-linearities [16]. The output of the convolutional layers feeds into a 2-layer LSTM [32] with 256 hidden units, followed by linear layers mapping the hidden units to a distribution over actions and a scalar value estimate.

The architecture for E3B's encoder is identical to that of the policy network, except that the LSTM is replaced by a linear layer (there is thus no recurrence in the encoder network).

C.2.2 RL Hyperparameters

For our VizDoom experiments, we also used IMPALA as our base policy optimizer. For E3B we used the same IMPALA hyperparameters as for MiniHack. The hyperparameters specific to E3B, ICM and RIDE are given below. For both ICM and RIDE, we found that performance was quite sensitive to the coefficients of the forward and inverse dynamics model losses. We ended up using the hyperparameters from [56], which we found gave the best performance.

Table 7: Hyperparameters for E3B on VizDoom

Hyperparameter	Values considered	Final Value
Running intrinsic reward normalization	{`False`}	`False`
Ridge regularizer λ	$\{0.1, 0.01\}$	0.1
Intrinsic reward coefficient β	$\{3 \cdot 10^{-7}, 10^{-6}, 3 \cdot 10^{-6}, 10^{-5}, 3 \cdot 10^{-5}, 10^{-4}\}$	$3 \cdot 10^{-6}$

Table 8: Hyperparameters for RIDE on VizDoom

Hyperparameter	Values considered	Final Value
Running intrinsic reward normalization	{`False`}	`False`
Forward loss coefficient	$\{0.5, 1.0\}$	0.5
Inverse loss coefficient	$\{0.1, 0.8\}$	0.8
Intrinsic reward coefficient β	$\{10^{-2}, 3 \cdot 10^{-3}, 10^{-3}, 3 \cdot 10^{-4}, 10^{-4}\}$	$3 \cdot 10^{-2}$

Table 9: Hyperparameters for ICM on VizDoom

Hyperparameter	Values considered	Final Value
Running intrinsic reward normalization	{`False`}	`False`
Forward loss coefficient	$\{0.2, 1.0\}$	0.2
Inverse loss coefficient	$\{0.1, 0.8\}$	0.8
Entropy cost	$\{0.0001, 0.005\}$	0.005
Intrinsic reward coefficient β	$\{10^{-2}, 3 \cdot 10^{-3}, 10^{-3}, 3 \cdot 10^{-4}, 10^{-4}\}$	$3 \cdot 10^{-3}$

C.3 Habitat

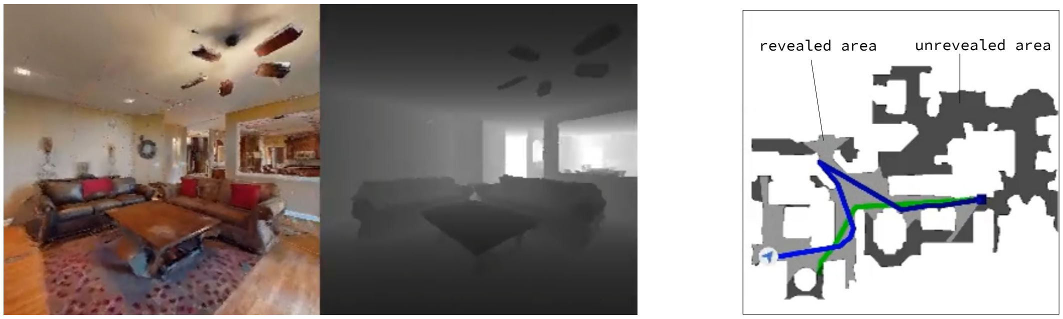
C.3.1 Environment Details

We used the HM3D [57] dataset, which consists of 1000 high-quality renderings of indoor scenes. Observations consist of 4 modalities: an RGB and depth image (shown in Figure 11a), 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 11b) for an illustration. For the results in Figure 8, 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 11: 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.

C.3.2 Architecture Details

For all Habitat experiments we used the same policy network as in [71], which includes a ResNet50 visual encoder [31] and a 2-layer LSTM [32] 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 (E3B 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.

C.3.3 RL Hyperparameters

The DD-PPO hyperparameters which are common to all the algorithms are listed in Table 10. The hyperparameters which are specific to each algorithm are listed in Table 11, 12, 4. For NovelD's

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

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 10: 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

Table 11: 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 12: 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 13: Hyperparameters for NovelD 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
Scaling factor α	0.1	0.1

Table 14: 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

C.3.4 Compute Details

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

C.4 Codebases Used

Our codebase was built atop the following codebases:

- The official NovelD codebase: https://github.com/tianjunz/NovelD (Creative Commons Attribution-NonCommercial 4.0 license) for NovelD, RND, RIDE and count-based baselines (this codebase is build atop the official RIDE codebase below)
- The official MiniHack codebase: https://github.com/facebookresearch/minihack for network architectures appropriate to MiniHack environments (Apache 2.0 license)
- The official RIDE codebase: https://github.com/facebookresearch/impact-driven-exploration (Creative Commons Attribution-NonCommercial 4.0 license) for network architectures appropriate to VizDoom environments
- The official Habitat codebase: https://github.com/facebookresearch/habitat-lab/tree/main/habitat_baselines for Habitat experiments

D Additional Results and Discussion

D.1 Additional MiniGrid Results

Here we provide results for RIDE, AGAC and NovelD on additional MiniGrid environments used in prior work, with and without their count-based episodic terms. For all algorithms, we used the code released by the authors with the default hyperparameters [3]. Results in Figure 12 confirm the trend in Figure 1: in all cases, removing the count-based episodic term results in a failure to learn or makes learning much slower (such as for RIDE in `MiniGrid-ObstructedMaze-2Dlh-v0`). We ran less seeds for AGAC because the official release's multithreading implementation was not compatible with our cluster.

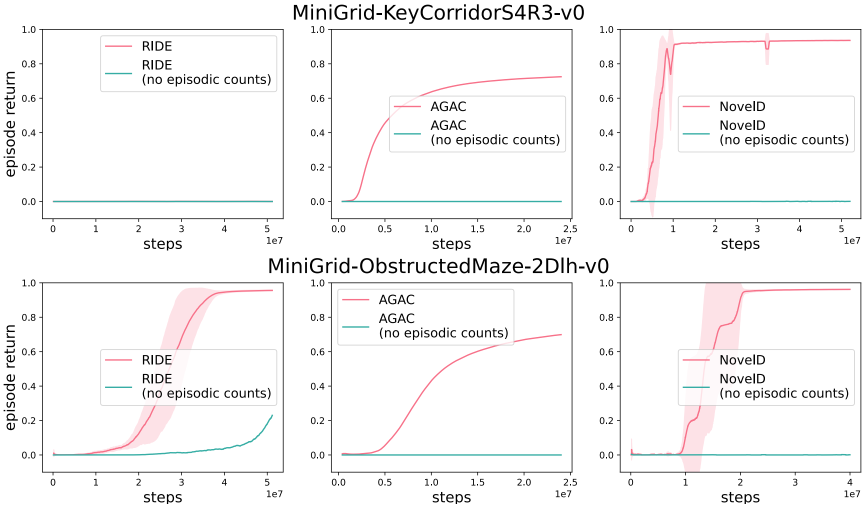


Figure 12: Results for RIDE, AGAC and NovelD with and without the count-based episodic bonus, over 5 random seeds (1 seed for AGAC). Shaded region indicates one standard deviation.

D.2 Effect of rank-1 updates

Here we compare the wall-clock time for computing the elliptical bonus using the rank-1 updates and the naive method of maintaining and inverting the full covariance matrix. We performed these experiments on the `MiniHack-Freeze-Random-Restricted-v0` environment using the standard

[3] NovelD: https://github.com/tianjunz/NovelD, RIDE: https://github.com/facebookresearch/impact-driven-exploration, AGAC: https://github.com/yfletberliac/adversarially-guided-actor-critic

hyperparameter setup, on a machine with 80 Intel(R) Xeon(R) CPU cores (E5-2698 v4 @ 2.20GHz) and one NVIDIA GP100 GPU:

E3B using rank-1 update	767 FPS
E3B using matrix inversion	256 FPS

This shows that using the rank-1 update is essential for fast performance.

D.3 Negative results for RIDE with modified count-based bonuses

We ran some preliminary experiments investigating modifications to RIDE similar to those for NovelD. Specifically, the modified RIDE reward bonus is:

$$b(s_t) = \|\phi(s_{t+1}) - \phi(s_t)\|_2 \cdot \frac{1}{\sqrt{N_e(\psi(s_{t+1}))}}$$

where ϕ is the embedding learned with the inverse dynamics model and $\psi(s_{t+1})$ extracts some aspect of the state s_{t+1}. We investigated a version where $\psi(s_{t+1})$ extracts the (x_{t+1}, y_{t+1}) position information from the state (this method we called RIDE-POSITION and is the same version that was run in [58]) and a version where ψ extracts the message portion of the state (this method we called RIDE-MESSAGE). Despite tuning the intrinsic reward coefficient over the range $\{0.0001, 0.001, 0.01, 0.1, 1, 10\}$ on two tasks (`MiniHack-MultiRoom-N10-v0` and `MiniHack-Freeze-Horn-Restricted-v0`) over 3 random seeds, none of the seeds was able to achieve positive reward for either of the tasks with either of the methods. Note that our results are consistent with those reported in [58], which also found that RIDE-POSITION did not outperform IMPALA on most tasks.

D.4 Additional MiniHack Results and Discussion

Results for all methods on individual tasks are shown in Figure 13. The first 9 tasks are navigation-based while the remaining 7 are skill-based.

First, as noted in the main text we see that when using the position bonus, NovelD succeeds in most of the tasks which primarily require the agent to spatially explore the environment, such as the `Labyrinth-Big` task or the different `MultiRoom` variants. On the other hand, this modality performs poorly on tasks based on skill acquisition, such as `Freeze`, `LavaCross` and `WoD` tasks. These tasks require the agent to pick up and use objects, and exploring the state space involves trying different action combinations which do not affect the agent's position. Therefore, an exploration bonus which only encourages the agent to visit many different positions is not helpful.

The message bonus, on the other hand, succeeds on the skill acquisition tasks. On these tasks, the correct sequence of actions indeed causes a sequence of novel messages to appear. For example, let us consider the task `Freeze-Wand`. In this task, the agent is in a small room and must go to a wand, pick it up, pick the ZAP action, choose the wand and then choose a direction to zap. When conducting this sequence of actions, it encounters the following messages: "you see here a brass wand" (after navigating to the wand location), "f - a brass wand" (after executing the PICKUP action), "What do you want to zap? [f or ?*]" (after executing the ZAP action). Each time the agent visits a state-action pair required to solve the task, it receives one of these messages which provides it with a positive reward signal thanks to the message-based episodic bonus. This in turn reinforces the behavior leading to that state-action pair and allows it to ultimately solve the task.

When using the message bonus, NovelD fails completely on most of the navigation-based tasks, such as `MultiRoom-N*-Lava-OpenDoor`, `LavaCrossingS19N13` and `LavaCrossingS19N17`. For these tasks, the agent must navigate its way through a series of rooms with lava walls or lava rivers with only one crossing. The only message it receives is "it's a wall" if it runs into walls, which does not incentivize it to explore more locations which will eventually lead it to the goal. In contrast, the position bonus easily solves these tasks.

The symbolic image bonus performs poorly on the skill-based tasks, but performs well on some of the navigation-based ones. This is likely because for many of these tasks, the agent it the only

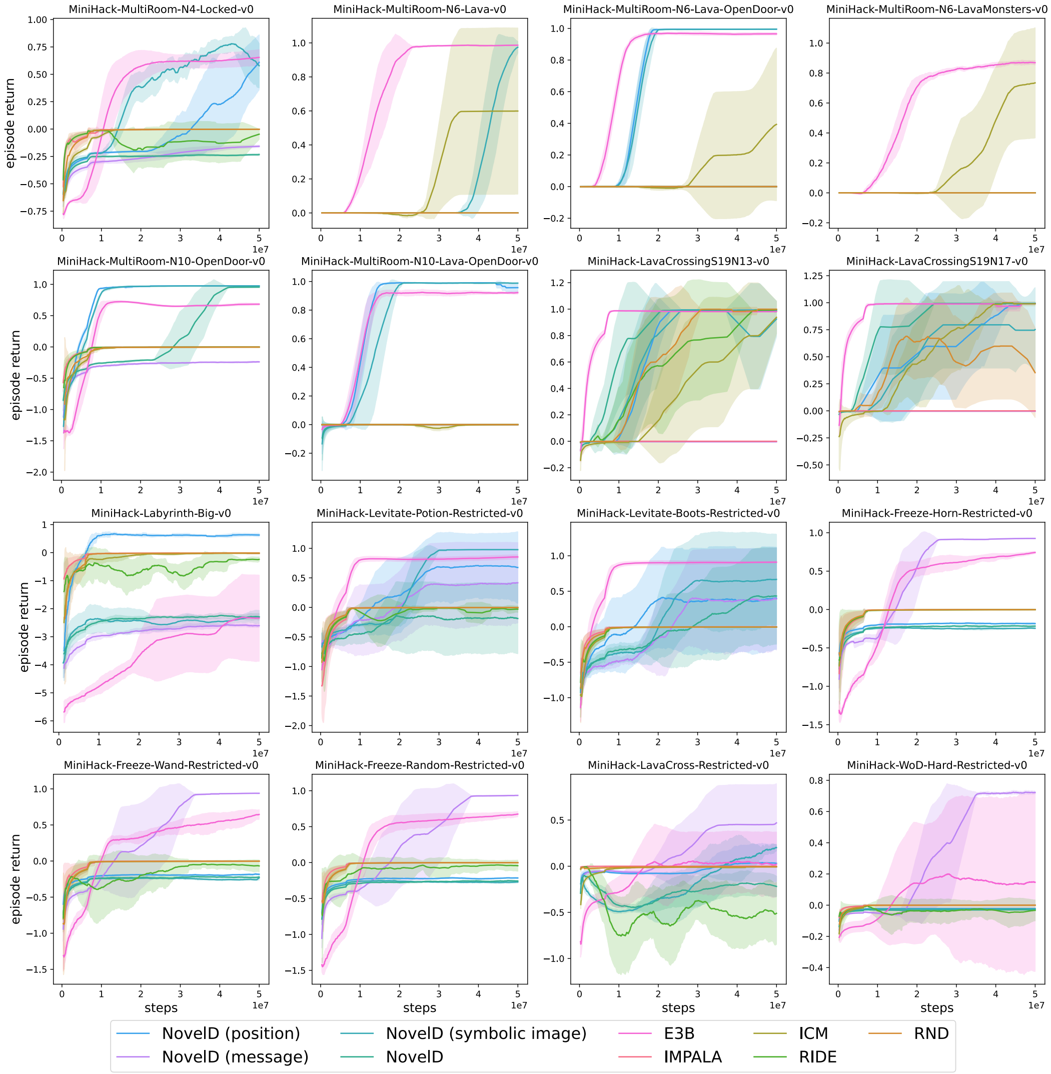


Figure 13: Mean results on individual tasks over 5 different seeds. Shaded region indicates one standard deviaton.

moving entity, and hence the positional bonus and the image bonus will be similar. However, this is not always true. For example, in the `Labyrinth-Big` task, the environment is initially mostly hidden and gets revealed over time as the agent explores. This means that there is not a one-to-one correspondence between symbolic images and agent positions, as would be the case if the entire map was initially revealed. On this task, the position bonus succeeds but the symbolic bonus does not. Another particularly revealing example is the fact that the symbolic bonus succeeds on the `MultiRoom-N6-Lava-OpenDoor` environment, but fails on the `MultiRoom-N6-Lava` environment. The only difference between these two tasks is that in the former, all doors are initially open, which is marked by the - symbol. In the latter, they are initially closed (marked by the + symbol), and as they are opened by the agent, the symbol switches to `-`. In the open door version, there is a one-to-one correspondence between agent positions and symbolic images. On the closed door version, the number of possible symbols is the number of positions times the number of possible combinations of doors being open and closed—a much larger number. Once any door is opened, if the agent revisits a position it visited when the door was closed, it will once again receive bonus. This encourages the agent to revisit previously visited locations each time a door is opened, which does not align well with the task. This explains the poor performance of the symbolic image bonus on the closed door version of the task. This again illustrates how count-based episodic bonuses can be sensitive to slight changes in the task's construction.

When using the count-based bonus based on the full observation, which does not make any assumptions about which parts of it are useful for the task, NovelD fails on all the tasks except for two. A close look at the stats vector (see Figure 4) reveals that it contains a time counter which increments each time step: this effectively makes each observation unique, and hence the episodic bonus is constant [4].

D.5 Sensitivity of λ regularizer

Figure 14 shows E3B's performance on MiniHack for different values of the covariance regularizer λ. There is no statistically significant difference between $\lambda = 0.1$, which is the value we used in our experiments, and $\lambda = 0.01$ or $\lambda = 1.0$. For $\lambda = 0.001$, we observe a statistically significant drop in performance for the IQM metric only. This shows that E3B is fairly robust to the value of λ.

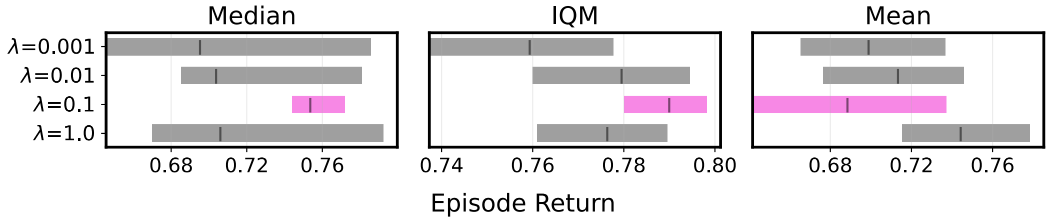


Figure 14: MiniHack performance for different values of λ parameter. All MiniHack experiments in the paper use the value $\lambda = 0.1$ unless otherwise noted. Intervals are computed using 5 random seeds using stratified bootstrapping.

[4] This is a simplification: there are some exceptions where the time counter is not incremented, such as if the agent runs into a wall or is searching through its inventory. This counts as a time step for the RL environment wrapper, but not for the underlying NetHack game engine. However these instances are relatively rare.