A Implementation Details

A.1 Task-driven exploration

Implementations of ViewX, NovelD, RIDE, and RND are built on the official codebase of NovelD[Zhang et al., 2021b]. For MADE, we directly take the results reported in the paper[Zhang et al., 2021a]. The intrinsic reward functions of the rest are as follows:

- ViewX: $(x_{t+1} - x_t + \lambda)/\sqrt{N(o_{t+1})}$, where x_t is the area of our constructed map and the map is reset at the beginning of each episode.
- NovelD: $\max[novelty(o_{t+1}) - \beta * novelty(o_t), 0] * \mathbb{1}\{N_e(o_{t+1} = 1)\}$, where $novelty(o_t) = \|\phi(o_t) - \hat{\phi}(o_t)\|^2$ - the difference between a random fixed target network $\hat{\phi}$ and a trainable predictor network ϕ. ϕ is trained to be close to $\hat{\phi}$ with the objective of minimizing $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$.
- RIDE: $\|\phi(o_t) - \phi(o_{t+1})\|^2/\sqrt{N(o_{t+1})}$, where ϕ is the state embedding network trained to minimize the prediction error of both an inverse and a forward dynamics model, $N(o_{t+1})$ is reset at the beginning of each episode.
- RND: $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$, where $\hat{\phi}$ is a fixed random network, ϕ is the state embedding network trained to minimize $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$.

Policy and value function The policy network and value function network are shared across different approaches. Given an input of dimension $7 \times 7 \times 3$ on MiniGrid, we fed it into three convolutional layers with kernel size 3×3, padding 1, number of channels 32, 128, 512, stride 1, 2, 2 respectively. Each convolutional layer is followed by a ELU activation function. After the last convolutional layer, the feature is flattened and processed by two linear layers with 1024 units and a ReLU activation function. The output from linear layers are put into an LSTM layer with 1024 units. After that, two separate fully-connected layers with 1024 units output action distribution as policy and value estimation as value function, respectively.

State embedding The state embedding network ϕ is learned for NovelD, RIDE, and RND. The input is the observation in MiniGrid with dimension $7 \times 7 \times 3$. It is passed through three convolutional layers with kernel size 3×3, padding 1, stride 1,2,2, number of channels 32, 128, 512 respectively. Each convolutional layer is followed by a ELU layer. The output is then passed to two linear layers of 2048 and 1024 units with ReLU activation.

Dynamics model RIDE requires training of the dynamics models to learn the state embedding network ϕ. For the forward dynamics model, the input is the state embedding and action. It is passed through two fully-connected layers with 256 and 128 units. The non-linear activation is ReLU activation function following each fully-connected layer. As for the inverse dynamics model, the input is state embeddings of two consecutive steps. The input is fed into two fully-connected layers with 256 units and a ReLU activation function.

Count dictionary In MiniGrid, the observation is of shape $7 \times 7 \times 3$ with values in the range of $[0, 10]$ as integers. We directly convert the observations o to tuples and store them as the keys in a dictionary to record the visitation counts $N(o)$.

Hyper-parameters Table 2 shows the values of hyper-parameters shared across different methods.

Parameter name	Value
Batch size	32
#Steps for LSTM	100
Optimizer	RMSProp
Learning rate	0.0001
Discount factor γ	0.99
Weight of policy entropy loss	0.0005
Weight of value function loss	0.5

Table 2: The hyper-parameters for experiments in task-driven setting.

For ViewX, we set the λ in Equation 1 as 0.01 for all tasks. For MultiRoom and KeyCorridor, the value of α is set to 0.01 and for ObstructedMaze environments it is set to 0.02.

For NovelD, we set $\alpha = 0.05$ for all the environments as is suggested in their official codebase.

Following [Raileanu and Rocktäschel, 2020], we use $\alpha = 0.1$ on KeyCorridor-S3R3 and $\alpha = 0.5$ on all other environments for RIDE and $\alpha = 0.1$ on all the environments for RND.

Compute resource Each job is run with an Nvidia TITAN X GPU and 40 CPUs. One job of ViewX takes around 4 hours for 10M steps.

In the supplementary material, we provide the code for experiments in the task-driven setting. More details can be found in the code.

A.2 Task-agnostic exploration

For a fair comparison, we implement ViewX for task-agnostic exploration on the codebase of [Parisi et al., 2021]. We compare exploration methods (ViewX, C-BET, RIDE, RND) with different intrinsic reward functions in the same codebase. The basic RL algorithm is IMPALA, the same for all these methods. The intrinsic reward functions are as follows:

- ViewX: $(x_{t+1} - x_t + \lambda)/\sqrt{N(o_{t+1})}$, where x_t is area of our constructed map and the map is reset at the beginning of each episode, the count $N(o_{t+1})$ is reset with a probability of 0.001 at each step.
- C-BET: $1/(N(change) + N(o_{t+1}))$, where $change$ is the environment panorama change of a transition $\{o_t, a_t, o_{t+1}\}$. The count N of $change$ and o_{t+1} is reset with a probability of 0.001 at each step.
- COUNT: $1/\sqrt{N(o_{t+1})}$, where the count $N(o_{t+1})$ is never reset.
- RIDE: $\|\phi(o_t) - \phi(o_{t+1})\|^2/\sqrt{N(o_{t+1})}$, where ϕ is the state embedding network trained to minimize the prediction error of both an inverse and a forward dynamics model, $N(o_{t+1})$ is reset at the beginning of each episode.
- RND: $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$, where $\hat{\phi}$ is a fixed random network, ϕ is the state embedding network trained to minimize $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$.

A.2.1 MiniGrid

Environment setting The episode length is the default maximum steps in each environment. In the task-agnostic setting, task is unknown so the episode is not terminated when the goal is achieved. We only terminate an episode when the number of steps reaches the maximum.

Policy and value function The network architecture for the policy and value function are the same for all methods. The input is the partial observations from the environment, with the shape of $7 \times 7 \times 3$. The input is fed into the convolutional neural network with three convolutional layers, and each layer has 32 filters, kernel size 3×3, stride 2 and padding 1. The non-linear activation function is ELU following each convolutional layer. The output of the convolutional network is flattened and then passed through two linear layers with 1024 hidden units and ReLU activation function. After the linear layers, the output is processed by LSTM layer with 1024 units. Finally, two separate fully-connected layers of 1024 units are used for output of policy and value function, respectively.

State embedding RIDE and RND requires learning of the state embedding network ϕ. The input is the observation in MiniGrid with dimension $7 \times 7 \times 3$. It is passed through three convolutional layers with kernel size 3×3, stride 2, padding 1, number of channels 32, 32, 128 respectively. Each convolutional layer is followed by a ELU layer. The output feature is flattened as state embedding.

The architecture of the dynamics network for RIDE, and the way to record count $N(o)$ are the same as we explained in Appendix A.1.

Count dictionary For C-BET, $N(change)$ is the count of any change in the panoramic view with dimension $7 \times 7 \times 3 \times 4$. The change in panoramic view is directly converted to tuples and stored as the key in the count dictionary.

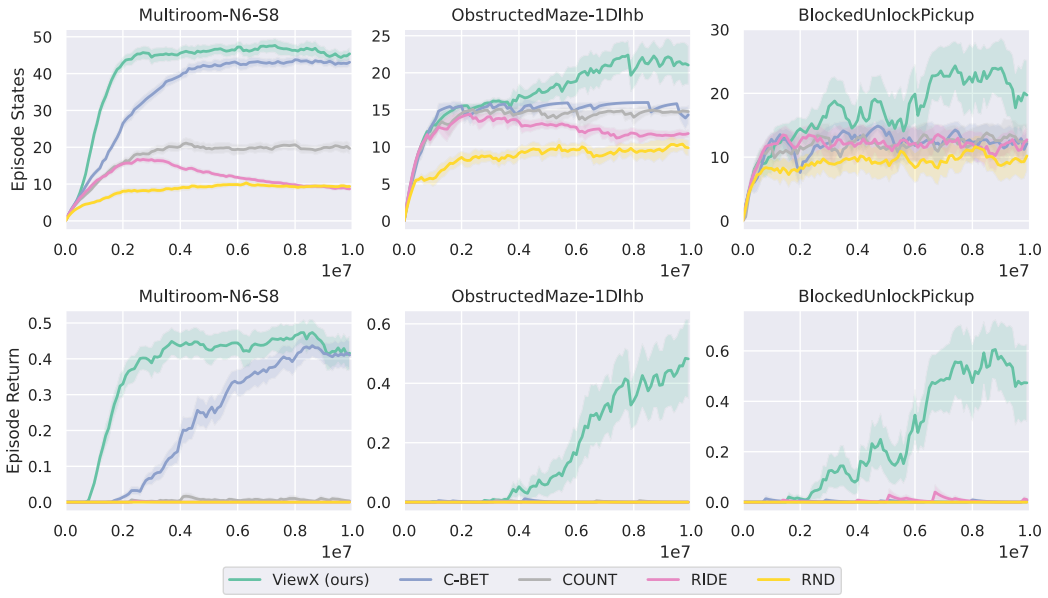

Figure 11: Training process of exploration policies in task-agnostic setting on MiniGrid

Hyper-parameters The values of hyper-parameters are listed in Table 3. The choice of the hyper-parameters follows the prior work [Parisi et al., 2021]. Most of the hyper-parameters are the same across different methods. The constant term λ in ViewX is set as 1 for simplicity without any hyper-parameter tuning.

Parameter name	Value
Batch size	8
#Steps for LSTM	100
Optimizer	RMSProp
Learning rate	0.0001
Discount factor γ	0.99
Gradient clip max norm	40
Weight of intrinsic reward α	0.005 (ViewX,C-BET, COUNT) 0.1(RIDE, RND)
Weight of policy entropy loss	0.0005
Weight of value function loss	0.5
Weight of forward dynamics loss	1.0 (RIDE)
Weight of inverse dynamics loss	0.1 (RIDE)
Weight of random network loss	0.1 (RND)
Constant term λ	1.0 (ViewX)

Table 3: The hyper-parameters for experiments in task-agnostic setting.

Compute resource Each job is run with an Nvidia TITAN X GPU and 10 CPUs. We set IMPALA using 10 actors. One job of ViewX takes around 20 hours for 10M steps.

We add more experiment details in Figure 11. We present the learning curves of the exploration policies on the training environments. The statistics we recorded during training is the number of visited cells and total extrinsic rewards in an episode. The episode rewards are measured only for evaluation, not for policy learning. The test performance of the learned exploration policy are presented in Figure 5,6,7 in the paper.

A.2.2 Habitat

Environment setting At each step, the agent can move forward by 0.25 meter or turn left/right by $10°$. The episode length is 500 steps and each episode terminates at 500 steps.

Policy and value function The input is an environment partial observation with dimension $64 \times 64 \times 3$. It is passed through five convolutional layers and each layer has 32 filters, kernel size 3×3, stride 2 and padding 1. The non-linear activation function is ELU. The output of the convolutional layers is flattened and fed into two fully-connected layers with 1024 hidden units and ReLU activation. Then the output is processed by LSTM layer with 1024 units. Finally, two separate linear layers of 1024 units are after LSTM and used for policy and value function respectively.

Count dictionary The observations are RGB images of dimension $64 \times 64 \times 3$, with integer values in the range of $[0, 255]$. We convert the observations to 128 bits with a hash function, the same as [Parisi et al., 2021]. The hash encoding is then used as the key of the count dictionary.

Compute resource Each job is run with an Nvidia TITAN X GPU and 10 CPUs. We set IMPALA using 10 actors. One run of ViewX tasks around 20 hours for 2M steps.

The network architecture of the state embedding network, forward dynamics model, inverse dynamics model, and the hyper-parameters are the same as experiments in MiniGrid.

In the supplementary materials, we provide some examples of videos showing the exploration behavior of different methods on training and test environments.

B Ablation Study

In this section, we provide more experiment details for Section 4.4.

B.1 Task-driven exploration

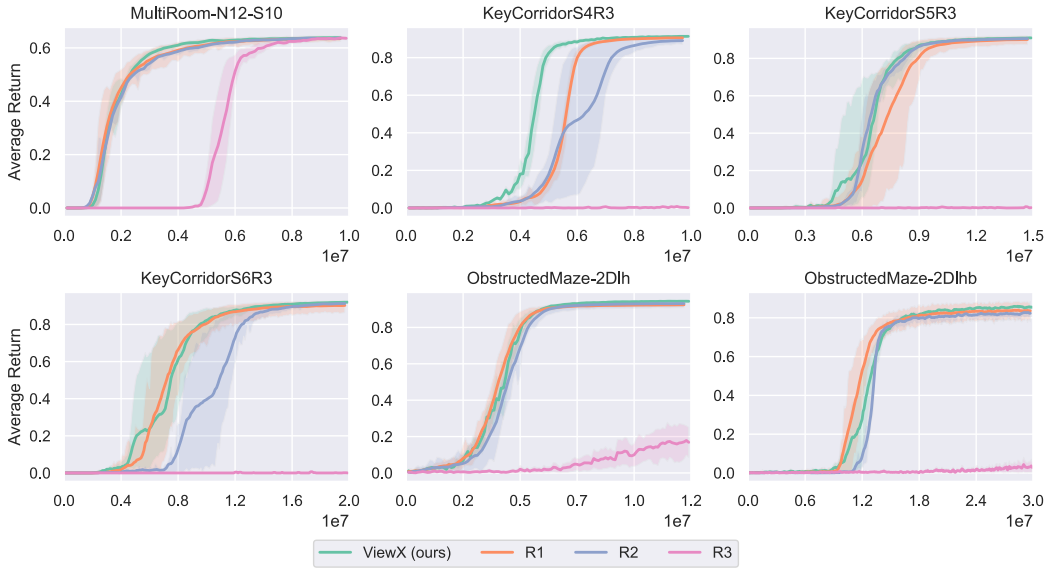

Figure 12: Learning curves of ViewX and the ablations.

Setting	ViewX	R1	R2	R3
KC-S4R3 (10M)	**0.9134** (±0.0029)	0.9057(±0.0024)	0.8969(±0.0029)	0.0023(±0.0008)
KC-S5R3 (15M)	**0.924** (±0.0031)	0.9185 (±0.0011)	0.9234 (±0.0047)	0.0031 (±0.0028)
KC-S6R3 (20M)	**0.9332**(±0.0024)	0.9245 (±0.0037)	0.9313 (±0.0024)	0.0000 (±0.0000)
MR-N12-S10(10M)	**0.6402**(±0.0029)	0.6401 (±0.0010)	0.6336 (±0.0086)	0.6399 (±0.0054)
OM-2Dlh (12M)	**0.9461** (±0.0007)	0.9239 (±0.0068)	0.9434 (±0.0014)	0.1359 (±0.1280)
OM-2Dlhb (20M)	**0.8598** (±0.0038)	0.8402 (±0.0143)	0.8281 (±0.0106)	0.0113 (±0.0015)

Table 4: Mean episodic rewards and standard deviations after training with a specific timesteps

Figure 12 shows the learning curves of ViewX and the three intrinsic reward designs we proposed in section 4.4 in task-driven setting. The performance of learned policy after task-driven training are summarized in Table 4. R1 has comparable sample efficiency with ViewX, but usually converges to sub-optimal policies. R2 has slightly worse sample efficiency than ViewX and R1, and R3 fails to learn in hard environments like KC-S6R3.

B.2 Task-agnostic exploration

In Figure 13, we show the learning process of exploration policies with different intrinsic reward functions. The policies are trained only with intrinsic rewards, and ViewX agent is about to visit more cells in the grid in an episode on the training environment. When trained on a difficult task like BlockedUnlockPickup (BUP), ViewX agent explores environments better than the variations R1, R2, and R3. In Table 5, we list the test performance of exploration policies in various environments, including training environment BUP and test environments. R1 and R2 agents fail to solve the BUP task because the view coverage maximization reward is mostly 0 before opening the door. Simply doing random exploration without any reward makes it challenging to open the door through a tedious process of picking up the key and moving around the obstacle. The agent has a slight chance to correctly interact with objects or find passages to leave the current room via random exploration. R3 agent takes the indicator of coverage increase for bonus reward, but this bonus is less informative

compared to ViewX. At the same time, the use of indicator may make the R3 agent prefer to only see a bit more in each step, and thus, it explores much worse than ViewX in complex environments.

Figure 13: Learning curves of exploration policies in task-agnostic setting.

Setting	ViewX	R1	R2	R3
BUP	**0.6788** (±0.1669)	0.0131 (±0.0129)	0.0123 (±0.0041)	0.0049 (±0.0039)
OM-1Dl	**0.5798** (±0.2270)	0.0257 (±0.0243)	0.0296 (±0.0169)	0.0078 (±0.0051)
OM-2Dl	**0.4154** (±0.1222)	0.0444 (±0.0212)	0.0629 (±0.0151)	0.0125 (±0.0034)
DK-6x6	**0.1920** (±0.1318)	0.1077 (±0.0254)	0.1473 (±0.0466)	0.0629 (±0.0137)
DK-8x8	**0.2617** (±0.1420)	0.0901 (±0.0370)	0.1349 (±0.0584)	0.0408 (±0.0182)
DK-16x16	**0.1405** (±0.0598)	0.0328 (±0.0174)	0.0555 (±0.0307)	0.0127 (±0.0043)

Table 5: Mean episode rewards and standard deviation after during with 10M timesteps.

C Map construction process

In Figure 14 and Figure 15, we provide an illustrative example to explain our process to construct the map m_t. At each step t, we suppose the partial observation o'_t from top-down viewpoint is available in (or can be learned from) the environment partial observation o_t. We assume the agent's relative position and orientation in any two consecutive steps are available. At the start of each episode, we assign a simple position and orientation p_0 (e.g. (0, 0, up) in MiniGrid environment) and then update p_t step by step based on the relative position and orientation. The map m_0 is reset as empty at the start of each episode. The top-down view observations o'_t are filled into our map according to the assumed position, and thus, m_t is incrementally accumulated in the episode.

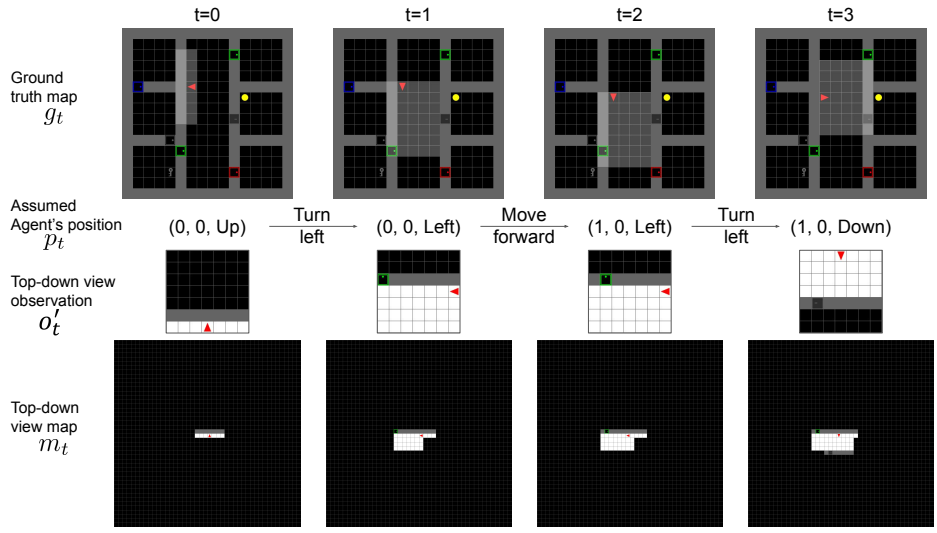

Figure 14: Illustrative example of map construction on MiniGrid Environment. The small red triangle annotates the agent's position and orientation, but this agent information is not available in observation o_t from the environment. We show it here only for better visualization and clearer explanation.

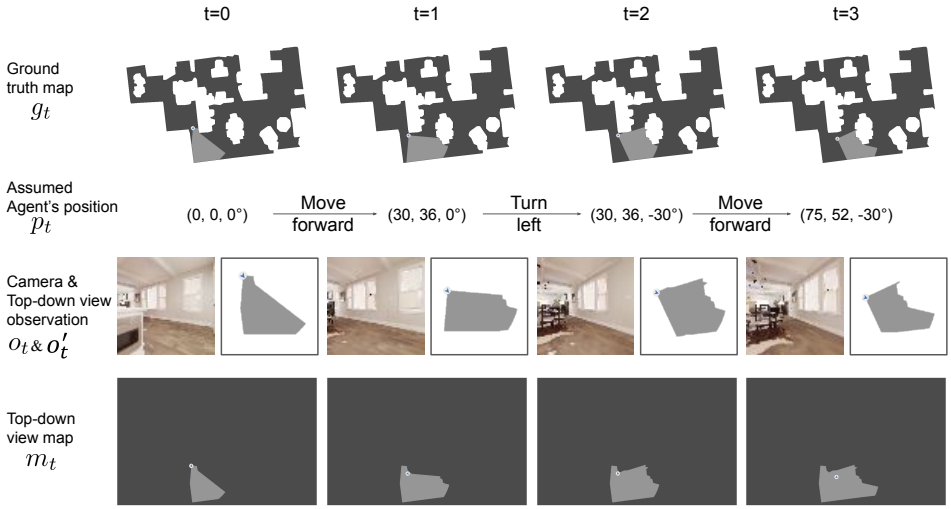

Figure 15: Illustrative example of map construction on Habitat. For more clear visualization, we make the action 'move forward', 'turn left' and 'turn right' to move more aggressively than the default setting (see Appendix A), so the observation difference in consecutive steps is more obvious.

As for our experiments in Section 4.3.2 in the Habitat environments, we assume that we have the ground truth top-down view observations o'_t corresponding to the partial observations o_t, as shown in

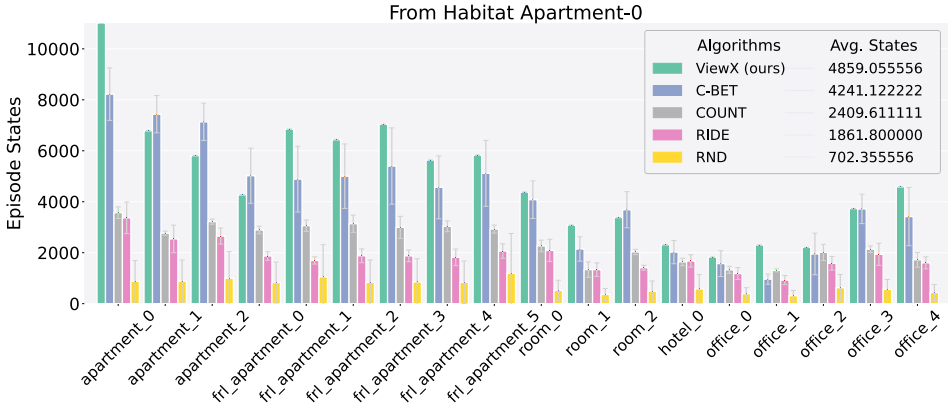

Figure 16: Exploration performance of ViewX with learned top-down view map

Figure 15. We further relax this assumption by utilizing an approach to approximate top-down view map from the first-person view observations [Chaplot et al., 2020b]. The performance of our approach and baselines is shown in Figure 16. ViewX with learned top-down view map still outperforms the baselines.

D Additional Experimental Results on Habitat

In this section, we provide results of additional experiments on Habitat environment.

D.1 Ablation Study

Figure 17 shows the exploration performance of ViewX and the intrinsic reward design R1, R2, and R3 in task-agnostic setting. The exploration policy is trained in one scene Apartment 0 with only intrinsic rewards for 2M steps. Here the intrinsic reward design R1 maximizes its view coverage, R2 excludes the constant term λ, and R3 uses indicator of whether view expands instead of raw increment of view coverage. Similar to the experimental results on MiniGrid environments, the performance of ViewX is also better than the all of the three designs on most of the Habitat environments.

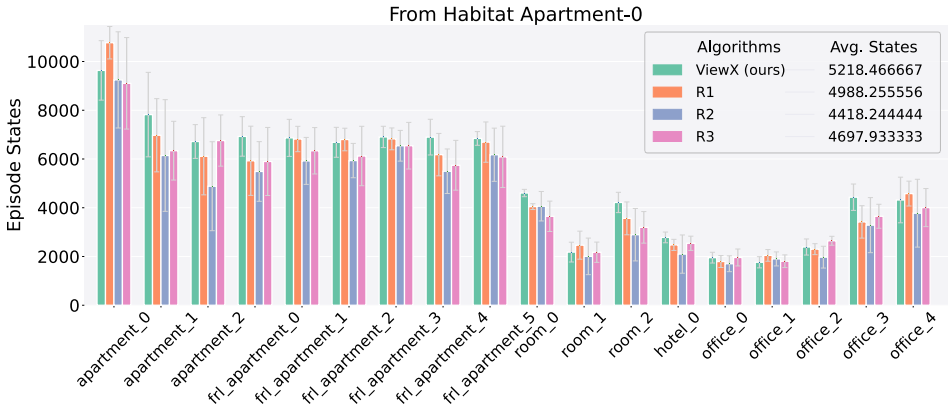

Figure 17: Exploration performance of ViewX and ablation models on scenes in Replica datasets

D.2 Additional Evaluation Metric

To further explore the capability of ViewX, we apply the learned exploration policy into Point Navigation task in test scenes in Replica datasets. Point Navigation task requires the agent to start from a specific position, and reaches the given goal in limited steps. Both the start and goal position are randomly sampled in the environment in every episode. We regard it as success once the distance between the goal and the agent is less than 0.2m, and the agent has 500 steps at maximum to explore the scene in each episode. On each scene, we report the average success rate in 200 test episodes. Figure 18 shows that ViewX outperforms the intrinsic reward design R1, R2, and R3 on the success rate of point navigation task, which means under the situation that both the start and goal position is unknown to the agent, ViewX helps to better explore unknown areas in limited steps.

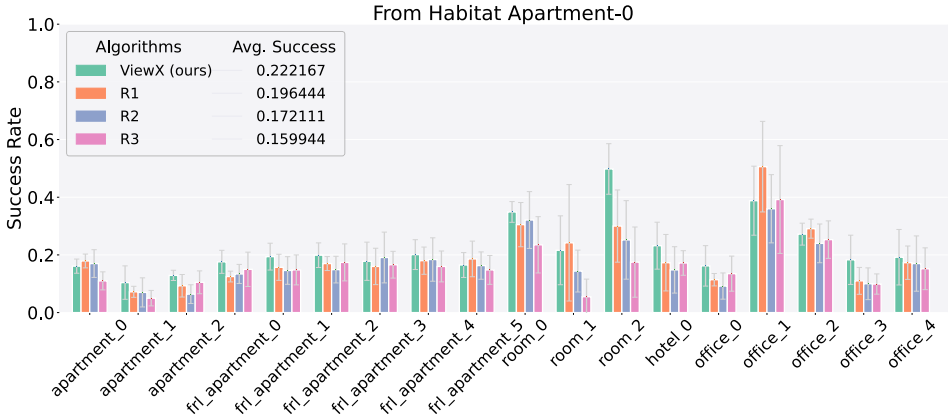

Figure 18: Success Rate of ViewX and ablation models on PointNav task on scenes in Replica datasets

E Relation with Information-Gain Exploration

In this section, we explain the relation between information gain exploration [Storck et al., 1995, Little and Sommer, 2013, Mobin et al., 2014, Charrow et al., 2015, Mirchev et al., 2018] and view coverage maximization method. The information gain exploration method maintains an internal model of the environment and optimizes the agent to achieve largest change in information in the internal model.

In our case, the map m_t is the agent's internal model of the environment. The change in information is quantified as the KL divergence between the map variable distributions at the start and end of an episode $KL(p(m|o_1, o_2, \cdots, o_T) \| p(m))$, where $m \in \mathbb{R}^{H \times W}$ is the environment map with height H and width W, and T is the limit of timesteps in an episode.

The distribution of the map variable at each cell (i, j) is independent from each other. Thus, $p(m) = \prod_{1 \le i \le H, 1 \le j \le W} p(m^{i,j})$ and $p(m|o_1, o_2, \cdots, o_T) = \prod_{1 \le i \le H, 1 \le j \le W} p(m^{i,j}|o_1, o_2, \cdots, o_T)$.

For each cell, the initial distribution $p(m^{i,j})$ follows the discrete uniform distribution on $1, 2, \cdots, K$, where K is the total number of possible objects and it is a prefixed constant. If the agent ever sees the cell (i, j) in its view after T steps, $p(m^{i,j}|o_1, o_2, \cdots, o_T) = 1$; if $m^{i,j}$ is the correct object k at the cell (i, j) then $p(m^{i,j}|o_1, o_2, \cdots, o_T) = 0$ otherwise.

According to these assumptions, the change in information can be simplified as:

$$
\begin{aligned}
& KL(p(m|o_1, o_2, \cdots, o_T) \| p(m)) \\
= & \sum_{m \in \{1,2,\cdots,K\}^{H \times W}} p(m|o_1, o_2, \cdots, o_T) \log \frac{p(m|o_1, o_2, \cdots, o_T)}{p(m)} \\
= & \sum_{m \in \{1,2,\cdots,K\}^{H \times W}} p(m|o_1, o_2, \cdots, o_T) \log \frac{\prod_{1 \le i \le H, 1 \le j \le W} p(m^{i,j}|o_1, o_2, \cdots, o_T)}{\prod_{1 \le i \le H, 1 \le j \le W} p(m^{i,j})} \\
= & \sum_{m \in \{1,2,\cdots,K\}^{H \times W}} p(m|o_1, o_2, \cdots, o_T) \sum_{1 \le i \le H, 1 \le j \le W} \log \frac{p(m^{i,j}|o_1, o_2, \cdots, o_T)}{p(m^{i,j})} \\
= & \sum_{1 \le i \le H, 1 \le j \le W} \sum_{m \in \{1,2,\cdots,K\}^{H \times W}} p(m|o_1, o_2, \cdots, o_T) \log \frac{p(m^{i,j}|o_1, o_2, \cdots, o_T)}{p(m^{i,j})} \\
= & \sum_{1 \le i \le H, 1 \le j \le W} \sum_{m^{/i,j} \in \{1,2,\cdots,K\}^{H \times W - 1}} p(m^{/i,j}|o_1, o_2, \cdots, o_T) \\
& \sum_{m^{i,j} \in \{1,2,\cdots,K\}} p(m^{i,j}|o_1, o_2, \cdots, o_T) \log \frac{p(m^{i,j}|o_1, o_2, \cdots, o_T)}{p(m^{i,j})} \\
= & \sum_{1 \le i \le H, 1 \le j \le W} \sum_{m^{i,j} \in \{1,2,\cdots,K\}} p(m^{i,j}|o_1, o_2, \cdots, o_T) \log \frac{p(m^{i,j}|o_1, o_2, \cdots, o_T)}{p(m^{i,j})} \\
= & \sum_{1 \le i \le H, 1 \le j \le W} KL(p(m^{i,j}|o_1, o_2, \cdots, o_T) \| p(m^{i,j}))
\end{aligned}
$$

If the cell (i, j) is never seen at this episode for T steps, the distribution $p(m^{i,j}|o_1, o_2, \cdots, o_T)$ is the same as initial distribution $p(m^{i,j})$). So the KL divergence for this cell in the map is 0.

If the cell (i, j) is seen in the agent's view and the object is $k \in \{1, 2, \cdots, K\}$, we have:

$$
\begin{aligned}
& KL(p(m^{i,j}|o_1, o_2, \cdots, o_T) \| p(m^{i,j})) \\
= & \sum_{m^{i,j} \in \{1,2,\cdots,K\}} p(m^{i,j}|o_1, o_2, \cdots, o_T) \log \frac{p(m^{i,j}|o_1, o_2, \cdots, o_T)}{p(m^{i,j})} \\
= & \ 1 \cdot \log \frac{1}{1/K} + \sum_{m^{i,j} \neq k, m^{i,j} \in \{1,2,\cdots,K\}} p(m^{i,j}|o_1, o_2, \cdots, o_T) \log \frac{p(m^{i,j}|o_1, o_2, \cdots, o_T)}{p(m^{i,j})} \\
= & \ \log K
\end{aligned}
$$

Overall, $KL(p(m|o_1, o_2, \cdots, o_T) \| p(m))$ can be written as *number of seen cells* $\times \log K$. In our notations, the number of seen cells after T steps in an episode is x_T.

Therefore, the information gain exploration seeking largest change in information is equivalent to seeking largest seen area in our setup. The view coverage maximization method with intrinsic reward $x_{t+1} - x_t$ for each step t is a special case of the information gain exploration.

F Hyper-parameter Sensitivity

The two major hyper-parameters for ViewX include the intrinsic reward coefficient and the λ from equation 1. For the intrinsic reward coefficient, most previous methods use 0.05, however, our intrinsic reward design includes an area change term which may be greater than 1. Therefore we lower the coefficient to 0.01 to avoid the intrinsic rewards being too large. For λ, we search the hyper-parameter in [0.001, 0.01, 0.05, 0.1]. As is shown in Figure 19, the performance of ViewX is not greatly influenced by λ and we set it to 0.01 across all the task-driven experiments for simplicity.

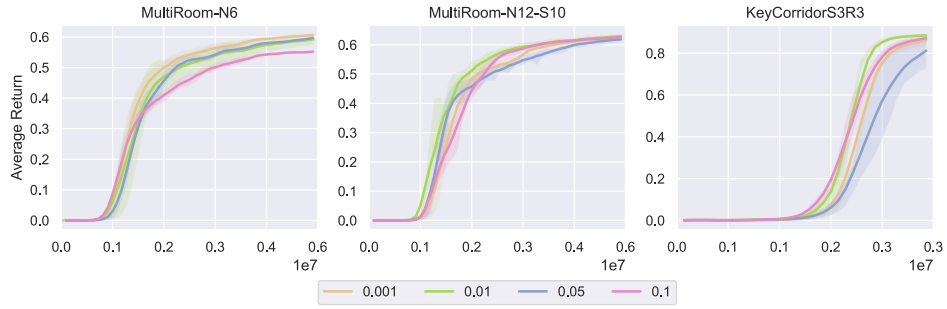

Figure 19: Task-driven experiment results with different λ.

G Comparison with Baselines using Localization Assumption

In this section, we experiment with one more baseline that uses localization assumption. Instead of recording the area that the agent "sees", we record the area where the agent has been to. More specifically, the intrinsic reward is designed as

$$r_t^{int} = \frac{l_{t+1} - l_t + \lambda}{\sqrt{N(o_{t+1})}} \tag{2}$$

where l_t denotes the total area that the agent has been to. The result is shown in Figure 20. We can see that changing view coverage to the coverage of the parts that the agent has physically been to makes the training much slower. The result is reasonable because in this baseline, if the agent sees some empty space, it still goes there for higher intrinsic reward, which prevents it from reaching the goal more efficiently. Therefore, visitation coverage l_t cannot replace view coverage x_t in our intrinsic reward design. Because the view coverage maximization reward in ViewX can be interpreted as an information-gain exploration bonus (see Appendix E), this comparison also implicitly verifies the benefit of pursuing information gain in ViewX intrinsic reward.

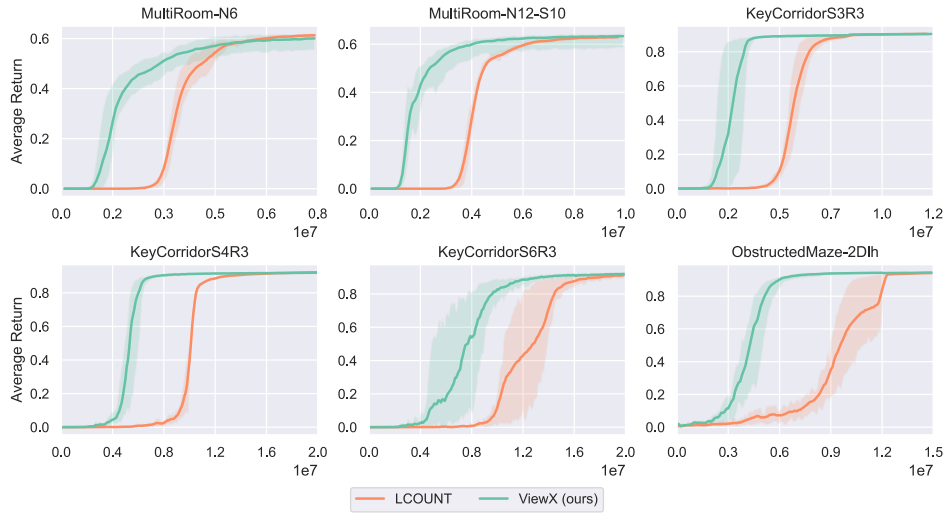

Figure 20: Compare ViewX with Location Count.