ENVIRONMENT PREDICTIVE CODING FOR VISUAL NAVIGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce environment predictive coding, a self-supervised approach to learn environment-level representations for embodied agents. In contrast to prior work on self-supervised learning for individual images, we aim to encode a 3D environment using a series of images observed by an agent moving in it. We learn these representations via a masked-zone prediction task, which segments an agent’s trajectory into zones and then predicts features of randomly masked zones, conditioned on the agent’s camera poses. This explicit spatial conditioning encourages learning representations that capture the geometric and semantic regularities of 3D environments. We learn the representations on a collection of video walkthroughs and demonstrate successful transfer to multiple downstream navigation tasks. Our experiments on the real-world scanned 3D environments of Gibson and Matterport3D show that our method obtains 2 - 6 × higher sample-efficiency and up to 57% higher performance over standard image-representation learning.

1 INTRODUCTION

In visual navigation, an intelligent embodied agent must move around a 3D environment using its stream of egocentric observations to sense objects and obstacles, typically without the benefit of a pre-computed map. Significant recent progress on this problem can be attributed to the availability of large-scale visually rich 3D datasets (Chang et al., 2017; Xia et al., 2018; Straub et al., 2019), and high-quality 3D simulators (Kolve et al., 2017; Savva et al., 2019a; Xia et al., 2020).

End-to-end reinforcement learning (RL) has been shown to achieve state-of-the-art navigation performance (Savva et al., 2019a; Wijmans et al., 2020). However, these approaches suffer from sample inefficiency and incur significant computational cost. Recent approaches try to mitigate these limitations by pre-training image representations offline and transferring them for navigation (Mirowski et al., 2016; Sax et al., 2020), or by performing auxiliary tasks and data augmentation in an online fashion during RL policy learning (Gordon et al., 2019; Kostrikov et al., 2021; Ye et al., 2020; 2021).

Current offline representation learning methods are flexible — once learned, the representations can be transferred to improve multiple embodied tasks. However, they are limited to learning image feature extractors (Gupta et al., 2017; Sax et al., 2020), or image-level proximity functions (Savinov et al., 2018; Chaplot et al., 2020c; Chang et al., 2020). Since embodied agents typically operate with limited field-of-view sensors, image representations only encode small parts of the scene in the nearby locality of the agent, and do not consider the broader context from the rest of the environment. We contend that embodied agents must learn higher-level semantic and geometric representations of the larger 3D environment around them, conditioned on their entire history of observations.

To that end, we introduce environment predictive coding (EPC), a self-supervised approach to learn environment-level representations that are transferrable to a variety of navigation-oriented tasks. The key idea is to learn an encoding of a 3D environment from a series of egocentric observations so as to be predictive of visual content that the agent has not yet observed. Consider the example in Fig. 1. An agent has observed the living room, lounge, and the bedroom in an unfamiliar house. The agent’s encoding of the observed spaces (i.e., the red trajectory) should be predictive of the visual features at an unseen location (x, y, θ), e.g., the green viewpoint, and enable inferences like “there is a small kitchen”, and “it contains a sink and an oven”. Learning such predictive representations can equip an agent with the ability to reason about 3D environments as it starts performing various
Predict visual features at \( \langle x, y, \theta \rangle \)

Figure 1: Environment Predictive Coding: During self-supervised learning, our model is given video walkthroughs of various 3D environments. We mask out portions of the trajectory (dotted lines) and learn to infer them from the unmasked parts (in red). The resulting EPC encoder builds environment-level representations of the seen content that are predictive of the unseen content (marked with a “?”), conditioned on the camera poses. We then transfer this learned encoder to agents performing various navigation tasks in novel environments.

navigation-oriented tasks. The proposed EPC model aims to learn such representations that capture the natural statistics of real-world environments in a self-supervised fashion, by simply watching video walkthroughs recorded by other agents.

To achieve this, we devise a self-supervised masked-zone prediction task in which the model learns environment embeddings in an offline fashion, by watching pre-collected video walkthroughs recorded by other agents navigating in 3D environments (See Fig. 1). The videos contain RGB-D and odometry. Specifically, we segment each video, into zones of temporally contiguous frames which capture local regions of the 3D environment. Then, we randomly mask out zones, and predict features for the masked zones conditioned on both the unmasked zones’ views and the masked zones’ camera poses. Since the overlap in scene content across zones is typically limited, the model needs to reason about the geometry and semantics of the environment to figure out what is missing.

Our general strategy can be viewed as a context prediction task in sequential data (Devlin et al., 2018; Sun et al., 2019b; Han et al., 2019)—but, very differently, aimed at learning high-level semantic and geometric representations of 3D environments to aid embodied agents acting in them. Unlike any prior video-based feature learning, our approach learns features conditioned on camera poses, explicitly grounding them in a 3D space; we demonstrate the impact of this important distinction.

Through simulated experiments in photorealistic scenes from Matterport3D (Chang et al., 2017) and Gibson (Xia et al., 2018), we show that transferring the EPC environment-level representations leads to 2 - 6× higher sample-efficiency and up to 57% better performance compared to only image-level transfer on 4 navigational tasks: room goal navigation, object visitation, flee, and area coverage.

Our contributions are: (1) we propose environment-predictive coding (EPC), a self-supervised approach to represent the underlying 3D environment given the observation sequences of an embodied agent, (2) we propose the proxy task of masked-zone prediction to learn environment-level representations from video walkthroughs captured by other agents, (3) we perform extensive experiments on Gibson and Matterport3D to demonstrate that EPC leads to good improvements on multiple navigation-oriented tasks, and study EPC’s design choices and noise robustness.

2 RELATED WORK

Self-supervised visual representation learning: Prior work leverages self-supervision to learn image and video representations from large unlabelled datasets. Image representation learning attempt proxy tasks such as inpainting (Pathak et al., 2016) and instance discrimination (Oord et al., 2018; Chen et al., 2020a; He et al., 2020), while video representation learning leverages signals such as temporal consistency (Jayaraman & Grauman, 2015; Wei et al., 2018; Kim et al., 2019) and contrastive predictions (Han et al., 2019; Sun et al., 2019a). VideoBERT (Sun et al., 2019a,b) jointly learns video and text representations from videos by filling in masked out information. Dense Predictive Coding (Han et al., 2019; 2020) learns video representations that capture the slow-moving semantics in videos. Whereas these methods tackle human activity recognition in videos, we aim to learn geometric and semantic cues in 3D spaces for embodied agents. Unlike the existing video models (Sun et al., 2019a;b; Han et al., 2019), which simply infer missing frame features conditioned on time, our approach explicitly grounds its predictions in the spatial structure of 3D environments.

Representation learning via auxiliary tasks for RL: Reinforcement learning approaches often suffer from high sample complexity, sparse rewards, and unstable training. Prior work tackles these using auxiliary tasks for learning image-level representations during online RL training (Mirowski...
Figure 2: We propose the masked-zone prediction task for self-supervised learning of environment embeddings. We learn from video walkthroughs generated by other agents moving under policies ignorant of our eventual downstream tasks. Each frame consists of the egocentric view and camera pose (top left). We group the frames in the video into seen zones in cyan \( \{Z_i^c\}^{n=0} \) and unseen zones in yellow \( \{Z_i^u\}^{n=0} \) (top row). The zones are generated automatically by grouping temporally contiguous sets of frames (bottom left). Given a camera pose \( p_i^c \) sampled from the unseen zone \( Z_i^u \), we use a transformer-based encoder-decoder architecture that generates environment embeddings \( E \) from the seen zones, and predicts the feature encoding \( f_i^c \) conditioned on the pose \( p_i^c \) (bottom center). The model is trained to distinguish the positive \( f_i^c \) from negatives in the same video \( \{f_j|f_j \neq f_i^c\} \) as well from other videos \( \{f_k\} \) (bottom right).

3 APPROACH

We propose environment predictive coding (EPC) to learn environment-level representations via self-supervision on video walkthroughs (Sec. 3.1). To demonstrate the utility of these representations, we integrate them into a transformer-based architecture and refine them for individual navigation tasks (Sec. 3.2). Finally, we describe our procedure for generating video walkthroughs (Sec. 3.3).

3.1 ENVIRONMENT PREDICTIVE CODING

Our hypothesis is that it is valuable for an embodied agent to learn a predictive coding of the environment. The agent must not just encode the individual views it observes, but also learn to leverage
the encoded information to anticipate the unseen parts of the environment. We propose to train an encoder-decoder model that observes a subset of views from a video walkthrough in a 3D environment, and then infers the features of unobserved views \textit{conditioned on their camera poses}. To successfully infer features from unobserved views, the encoder must build a predictive representation of the underlying physical environment using the observed views. By transferring this encoder to a navigation agent, we equip the agent with the structural and semantic priors of 3D environments to quickly perform new tasks in new spaces, like mapping the house or room goal navigation.

We propose the self-supervised task of masked-zone prediction to achieve this goal (see Fig. 2). For this task, we use a dataset of egocentric video walkthroughs containing (possibly noisy) RGB-D and odometer sensor readings collected by other agents deployed in various unseen simulated environments (Fig. 2, top). These environments are inaccessible for interactive RL training, and the agent policies are ignorant of our eventual downstream tasks (see Sec. 3.3). Our method works as follows. First, we automatically segment each video into “zones” which contain temporally contiguous sets of frames. We then learn an environment encoder via the self-supervised masked-zone prediction task on the segmented videos. Finally, we transfer the learned environment encoder to an array of downstream navigation-oriented tasks. We explain each step in detail next.

**Zone generation** At a glance, one might first consider masking arbitrary individual frames in the training videos. However, doing so can result in poor representation learning since shared content from nearby unmasked frames can make the prediction task trivial. Instead, our approach masks \textit{zones} of frames at once. We define a zone to be a set of temporally contiguous frames in the video. By choosing a large-enough temporal window, we can reduce the amount of shared content with temporally adjacent zones. Given a video walkthrough of size $L$, we divide it into zones $\{Z_0, Z_1, \cdots \}$ of length $N$ (selected through validation):

$$Z_i = \{ (o_i, p_i) \mid \forall t \in [t_s, t_e] \} ,$$

where $t_s = i \times N$, $t_e = \min((i+1) \times N, L)$, $o_i$ is the RGB-D sensor reading, and $p_i$ is the camera pose obtained by accumulating odometer readings from time $0$ to $t$ (see Fig. 2, bottom left). While two zones may share visual content, we find that this simple approach works better than strictly limiting the overlap between zones (see Appendix. A8). Thus, the learning is guided by predicting parts of the environment that were never seen as well as those seen from different viewpoints.

**Masked-zone prediction** Having segmented the video into zones, we next present our EPC masked-zone prediction task to learn environment embeddings (see Fig. 2). The main idea is to infer unseen zones in a video by previewing the context spanning multiple seen zones. We randomly divide the zones into seen zones $\{Z_i^s\}_{i=1}^n$ and unseen zones $\{Z_i^u\}_{i=1}^m$. Given the seen zones and the mean camera pose from an unseen zone $p_i^u$, we need to infer a feature encoding of the unseen zone $Z_i^u$. To perform this task efficiently, we first extract visual features $x_i$ from each RGB-D frame $o_i$ in the video using pretrained CNNs (described in Sec. 3.2). These features are concatenated with the corresponding pose $p_i$ and projected using an MLP $\mathcal{M}$ to obtain the image-level embedding. The target features for the unseen zone $Z_i^u$ are obtained by averaging\(^1\) all the MLP projected features:

$$f_i^u = \frac{1}{|Z_i^u|} \sum_{x \in Z_i^u} \mathcal{M}([x, \overrightarrow{0}]) ,$$

where we mask out the pose (i.e., $p = \overrightarrow{0}$) in the target to avoid trivial solutions. We use a transformer encoder-decoder model (Vaswani et al., 2017) to infer the zone features (see Fig. 2, bottom). An environment encoder uses self-attention over the image-level embeddings from all the seen zones, i.e., $\{\mathcal{M}([x, p]) \mid \forall (x, p) \in Z_i^s, \forall i \in [1, n]\}$, to generate the environment embeddings $\mathcal{E}$. A zone decoder then attends to $\mathcal{E}$ conditioned on the camera pose $p_i^u$ from the unseen zone and predicts the zone features:

$$\hat{f_i^u} = \text{ZoneDecoder}(\mathcal{E}, p_i^u) .$$

Following Fang et al. (2019), we transform all poses in the input zones relative to $p_i^u$ before encoding, which provides the model an egocentric view of the world. As we will show in experiments, conditioning on pose is critical to learn useful representations. The environment encoder, zone decoder, and projection function $\mathcal{M}$ are trained end-to-end using noise-contrastive estimation (Gutmann & Hyvärinen, 2010). We use $f_i^u$ as the anchor and $\hat{f_i^u}$ from Eqn. 2 as the positive. We sample

\(^1\)We found this to be better than randomly sampling features within a zone.
Environment embeddings for embodied agents

Once a self-supervised environment encoder is trained, we transfer the encoder to various agents for performing navigation-oriented tasks. To this end, we integrate our pretrained environment encoder into the Scene Memory Transformer (SMT) from Fang et al. (2019). While our idea is potentially applicable to other memory models, our choice of SMT is motivated by the recent successes of transformers in NLP (Devlin et al., 2018) and vision (Sun et al., 2019b; Fang et al., 2019). We briefly overview the SMT architecture (Fig. 3, center). We extract visual features from each RGB-D input (concatenated along the channel dimension) using a ResNet-18 image encoder (He et al., 2016). We store the visual features and agent poses \( \{(x_i, p_i)\}_{i=0}^t \) observed during the episode in a scene memory. We then use an environment encoder to perform self-attention over the scene memory and generate a rich set of environment embeddings \( \mathcal{E} \). We use a policy decoder to attend to \( \mathcal{E} \) conditioned on the inputs \( o_t = [x_t, p_t] \) at time \( t \), and use the decoder outputs to sample an action \( a_t \) and estimate the value \( v_t \). We detail each component in the Appendix A2.

To incorporate our EPC environment embeddings into SMT, we first initialize the image encoder using CNNs pre-trained using a state-of-the-art MoCo-v2 method (Chen et al., 2020b) (see Fig. 3, right). The image encoder is pre-trained for 7,000 epochs on RGB-D images sampled from the video walkthroughs generated for EPC pre-training. Note that this is the CNN used to pre-extract RGB-D image features during EPC masked-zone prediction. Next, and most importantly, we initialize the environment encoder with our EPC pre-training on masked-zone prediction (see Fig. 3, left). We then finetune the SMT model end-to-end on the downstream task—whether that is room navigation, object visitation, flee, or area coverage (cf. Sec. 4). As we will demonstrate in the experiments, initializing the environment encoder using EPC leads to 2-6x higher sample efficiency and better performance when compared to initializing only the image encoder.

\[ L_i = -\log \frac{\text{sim}(f_i^n, f'_o)}{\sum_j \text{sim}(f_i^n, f_j) + \sum_k \text{sim}(f_i^n, f'_k)}, \]

where \( f_i^n \) are zone features from other videos, and \( \text{sim}(q, k) = \exp \left( \frac{q_k}{\|q\| \|k\| \tau} \right) \) is a similarity measure with temperature \( \tau = 0.1 \). The idea is to predict zone representations that are closer to the ground truth, while being sufficiently different from the negative zones. Since the seen and unseen zones may only have limited overlap, the model needs to effectively reason about the geometric and semantic context in the seen zones to perform this task. We qualitatively analyse the masked-zone prediction results from the learned EPC model in Fig. 4.

3.2 Environment embeddings for embodied agents

Figure 3: Integrating environment-level pretraining for navigation. Right: First, we transfer image-level representations to encode each RGB-D image. We transfer weights from an image encoder pre-trained for the self-supervised MoCo-v2 task on video walkthrough images. Left: Next, and most importantly, we transfer the environment-level representations to encode the entire history of observations. We transfer weights from the environment encoder and projection function \( \mathcal{M} \) pre-trained for the proposed EPC masked-zone prediction task on video walkthroughs. Center: Finally, we finetune the SMT end-to-end on each task using RL.
3.3 VIDEO WALKTHROUGH GENERATION

As discussed previously, EPC relies on a dataset of video walkthroughs containing RGB-D and odometer readings for self-supervision. Since EPC aims to learn the natural statistics of real-world environments (and not agent behavior), it suffices to use walkthroughs recorded in diverse environments by agents performing their day-to-day tasks. In particular, we do not require that the scenes used to generate walkthroughs are available for interactive policy learning, nor that the agent behaviors used for generating walkthroughs are tied to the downstream navigation tasks of interest. This means that we can, in parallel, collect videos from different agents operating in a large number of environments across geographical locations. We now realize this process in photorealistic Gibson scenes; we leave leveraging in-the-wild consumer videos as a challenge for future work.

We generate 2 sets of egocentric video walkthroughs by deploying simulated agents in photorealistic scanned indoor environments from Gibson\(^3\) (Xia et al., 2018). 1) Strong Exploration: an SMT agent that was trained for area coverage on MP3D, and 2) Weak Exploration: a heuristic forward-biased navigation agent that moves forward until colliding, then turns randomly, and repeats. The weak exploration agent generates less-informative walkthroughs as it tends to repeatedly explore the same regions. Experimenting with EPC on both types of walkthrough allows us to test its dependence on walkthrough quality. Methods robust to this quality are desirable since it may be easier to collect lower quality walkthroughs on a large-scale. In both cases, the agents explore each Gibson environment starting from multiple locations for 500 steps per location and record the RGB-D and odometer readings. This results in 5,047 videos per agent, which we divide into an 80-20 train/val split (i.e., \(\sim\)2M training frames). As we will show in Sec. 4, the EPC encoders learned on these walkthroughs are applicable to navigation tasks not tied to the exploration agents.

4 EXPERIMENTS

First, we review the experimental setup (Sec. 4.1). We then evaluate the pre-trained EPC embeddings on multiple downstream navigation-oriented tasks in unmapped environments (Sec. 4.2). Finally, we present an ablation study of the data and proxy task used for EPC (Sec. 4.3).

4.1 EXPERIMENTAL SETUP FOR DOWNSTREAM TASKS

We perform experiments on the Habitat simulator (Savva et al., 2019b) with Matterport3D (MP3D) (Chang et al., 2017) and Gibson (Xia et al., 2018), two challenging and photorealistic 3D datasets with 90 and 572 scanned real-world indoor environments, respectively. Our observation space consists of \(171 \times 128\) RGB-D images and agent pose \(p = (x, y, \theta)\) relative to the starting

\(^3\)We use 332 Gibson environments from the Gibson 2+ training split (Wijmans et al., 2020).
pose at $t = 0$ (obtained by accumulating odometer readings). Our action space consists of: MOVE-FORWARD by 25cm, TURN-LEFT by $30^\circ$, TURN-RIGHT by $30^\circ$, and STOP (only for RoomNav). For all methods, we assume noise-free sensing during training, then evaluate with both noise-free and noisy sensing (pose, depth).

We perform interactive RL training on 61 MP3D train scenes. We evaluate the learned policies on 11 val and 18 test scenes in MP3D, and 14 val scenes in Gibson. These are disjoint from the 332 Gibson train scenes used for walkthroughs. We use episode lengths of $T = 1000$ for MP3D, and $T = 500$ for Gibson since the scenes are smaller. We further divide Gibson results into small and large environments (Ramakrishnan et al., 2020; Chaplot et al., 2020b). We evaluate our approach on four standard navigation tasks from the literature:

1. **Area coverage** (Chen et al., 2019; Chaplot et al., 2020b; Ramakrishnan et al., 2021): Agent is trained to maximize the area covered (in $m^2$) within a fixed time budget.
2. **Flee** (Gordon et al., 2019): Agent is trained to maximize the ‘flee distance’, i.e., the geodesic distance (in m) between its start and end positions, within a fixed time budget.
3. **Object visitation** (Fang et al., 2019; Ramakrishnan et al., 2021): Agent is trained to maximize the # object categories visited within a fixed time budget. An object is ‘visited’ if it is visible to the agent within 1.5m of the agent’s position. We report both the # categories and # instances visited.
4. **RoomNav** (Savva et al., 2017; Wu et al., 2019; Narasimhan et al., 2020): Agent is trained to find the nearest room instance of a provided room category. We evaluate using the two standard metrics: SPL and Success (Anderson et al., 2018a). Please see Appendix A4 for the task details.

We choose these tasks since they capture different forms of geometric (tasks 1 & 2) and semantic (tasks 3 & 4) inference in 3D environments. All tasks are different from the agent behavior in the weak exploration trajectories. While task 1 is well-aligned with the strong exploration trajectories, all other tasks are distinct. Object visitation is a challenging task since the agent must navigate within 1.5m of each object and directly observe it. RoomNav is the most challenging task since the goal is to visit a specific room (not arbitrary exploration).

We compare our approach to several baselines.

**Scratch baselines:** We randomly initialize the visual encoders and policy, and train them end-to-end for each task. Images are encoded using ResNet-18. Agent pose and past actions are encoded using FC layers. *Reactive (scratch)* has no memory. *RNN (scratch)* uses a 2-layer LSTM as the temporal memory. *SMT (scratch)* uses a Scene Memory Transformer (SMT) for aggregating observations.

**SMT (MidLevel):** This SMT-based model uses image encoders pre-trained for various supervised visual-tasks. The image encoders are kept frozen during RL (Sax et al., 2020; Zamir et al., 2020).

**SMT (MoCo):** This SMT-based model uses a ResNet-18 image encoder pre-trained using MoCo-v2 (Chen et al., 2020b) and finetunes it end-to-end during RL. This is an ablation of our model from Sec. 3.2 where only the image encoder is pre-trained on the video walkthroughs. This SoTA image-level pre-training is critical to isolate the impact of our proposed environment-level pre-training.

**SMT (Video):** This SMT-based model is inspired by Dense Predictive Coding (Han et al., 2019). The image encoder is initialized using weights from SMT (MoCo). It pre-trains the environment encoder as a ‘video-level’ model on the video walkthroughs. It takes 25 consecutive frames as input and predicts the features from the next 15 frames (following timespans used in Han et al. (2019)). During SSL, we mask the input camera poses and query based on time, unlike EPC which uses input poses and queries based on pose during SSL.

**ANS:** This is our implementation of the SoTA hierarchical policy from Active Neural SLAM (Chaplot et al., 2020b), upgraded to perform depth-based occupancy mapping (instead of using only RGB).

All RL agents receive RGB-D images and pose inputs. All models are trained for 13M-15M frames with 60 parallel processes. We train the scratch and ANS baselines for 2M more frames to account for the 2M frames in video walkthroughs used for other methods. Both SMT (MoCo) and SMT (Video) are given the advantage of using strong exploration walkthroughs for SSL (see Sec. 3.3). However, we test EPC with either strong or weak exploration walkthroughs (variants denoted as S.E and W.E, respectively). See Appendix A5 for optimization hyperparameters.

### 4.2 Downstream Task Performance

We transfer the EPC features to downstream navigation tasks. Tab. 1 shows the complete results. For brevity in the text, we also report the change in a method’s performance relative to another
with EPC (W.E) and EPC (S.E). On area coverage, while ANS performs better on Gibson, EPC outperforms it on the larger MP3D environments. On RoomNav, ANS fails to learn the STOP action resulting in 0% success. Note that ANS was not originally designed with a STOP action. Unlike the other agents with 4 actions, ANS has 24 × 24 location actions (see Appendix A3). Adding a STOP action to this huge action space makes RL optimization difficult.

Finally, EPC matches or outperforms the state-of-the-art ANS (Chaplot et al., 2020b) on both geometric and semantic tasks. We relatively improve over ANS by (2%, 24%, 24%, N/A) with EPC (W.E) and (2%, 40%, 15%, N/A) with EPC (S.E). On area coverage, while ANS performs better on Gibson, EPC outperforms it on the larger MP3D environments. On RoomNav, ANS fails to learn the STOP action resulting in 0% success. Note that ANS was not originally designed with a STOP action. Unlike the other agents with 4 actions, ANS has 24 × 24 location actions (see Appendix A3). Adding a STOP action to this huge action space makes RL optimization difficult.

In Fig. 5, we plot the validation metrics of SMT (MoCo) and EPC over the course of training. We see that environment-level pretraining offers significantly better sample efficiency: both EPC variants reach the best performance of SMT (MoCo) up to 6× faster. This confirms our hypothesis: transferring environment-level encoders learned via spatial reasoning helps embodied agents learn faster compared to the current approach of transferring image-level encoders alone (Sax et al., 2020).

Table 1: Downstream task performance at the end of the episode. The two tasks in blue are geometric, and the two tasks in green are semantic. Gib-S/L means Gibson small/large. (# i / # c) means number of object instances/categories visited. All methods are trained and evaluated on 2 and 3 random seeds, respectively. In each column, the best methods are highlighted in bold (using a one-sided T-test with p = 0.05). We report only the mean due to space constraints. See Appendix A11 for performance vs. time step plots.

<table>
<thead>
<tr>
<th>Method</th>
<th>Gib-S</th>
<th>Gib-L</th>
<th>MP3D</th>
<th>Gib-S</th>
<th>Gib-L</th>
<th>MP3D</th>
<th>Gibson</th>
<th>MP3D</th>
<th>Gibson</th>
<th>MP3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive (scratch)</td>
<td>28.2</td>
<td>50.1</td>
<td>121.8</td>
<td>3.0</td>
<td>4.0</td>
<td>6.6</td>
<td>6.4/4.0</td>
<td>12.1/4.9</td>
<td>1.5/0.1</td>
<td>0.9/0.2</td>
</tr>
<tr>
<td>RNN (scratch)</td>
<td>30.8</td>
<td>57.6</td>
<td>130.1</td>
<td>5.3</td>
<td>8.1</td>
<td>12.0</td>
<td>6.7/4.2</td>
<td>13.8/5.3</td>
<td>6.8/2.2</td>
<td>5.0/1.6</td>
</tr>
<tr>
<td>SMT (scratch)</td>
<td>32.2</td>
<td>62.6</td>
<td>166.5</td>
<td>4.3</td>
<td>6.4</td>
<td>11.0</td>
<td>9.2/5.4</td>
<td>17.8/6.3</td>
<td>5.0/2.3</td>
<td>3.0/1.4</td>
</tr>
<tr>
<td>SMT (MidLevel)</td>
<td>31.4</td>
<td>57.1</td>
<td>147.2</td>
<td>4.8</td>
<td>6.7</td>
<td>11.2</td>
<td>7.6/4.6</td>
<td>15.2/5.7</td>
<td>10.4/4.8</td>
<td>5.5/2.3</td>
</tr>
<tr>
<td>SMT (MoCo)</td>
<td>32.4</td>
<td>66.4</td>
<td>177.5</td>
<td>5.1</td>
<td>8.0</td>
<td>13.1</td>
<td>9.1/5.3</td>
<td>18.8/6.5</td>
<td>20.1/8.9</td>
<td>12.7/5.5</td>
</tr>
<tr>
<td>SMT (Video)</td>
<td>31.9</td>
<td>61.3</td>
<td>157.1</td>
<td>4.9</td>
<td>7.7</td>
<td>11.8</td>
<td>8.7/5.1</td>
<td>16.4/6.0</td>
<td>18.1/7.1</td>
<td>12.0/5.0</td>
</tr>
<tr>
<td>ANS</td>
<td>33.3</td>
<td>81.5</td>
<td>172.8</td>
<td>3.6</td>
<td>7.8</td>
<td>13.0</td>
<td>10.0/5.9</td>
<td>20.7/6.9</td>
<td>0.0/0.0</td>
<td>0.0/0.0</td>
</tr>
<tr>
<td>EPC (W.E)</td>
<td>33.9</td>
<td>76.1</td>
<td>195.1</td>
<td>5.9</td>
<td>10.6</td>
<td>16.6</td>
<td>11.9/6.5</td>
<td>24.0/7.6</td>
<td>28.9/12.6</td>
<td>21.1/9.7</td>
</tr>
<tr>
<td>EPC (S.E)</td>
<td>33.7</td>
<td>75.8</td>
<td>198.2</td>
<td>6.0</td>
<td>10.3</td>
<td>15.8</td>
<td>12.0/6.6</td>
<td>24.5/7.7</td>
<td>31.5/13.9</td>
<td>19.8/8.9</td>
</tr>
</tbody>
</table>

Figure 5: Sample efficiency on Matterport3D val split: We compare the sample-efficiency of EPC to that of the standard approach of pretraining only image-level features. Our environment-level pretraining leads to 2× to 6× better training sample efficiency on all four tasks. See Appendix A6 for corresponding Gibson plots.
In future work, we aim to extend our idea to in-the-wild videos and apply it to multimodal tasks such as embodied question answering, instruction following, and audio-visual navigation.

### 5 Conclusions

We introduced Environment Predictive Coding, a self-supervised approach to learn environment-level representations for embodied agents. By training on video walkthroughs generated by other agents, our model learns to infer missing content through a masked-zone prediction task. When transferred to multiple downstream embodied agent tasks, the resulting embeddings lead to 2-4× higher sample efficiency and up to 57% better performance when compared to the current practice of transferring only image-level representations. Our work highlights the advantages of self-supervised learning in agent videos by predicting features grounded in the spatial structure of 3D environments. In future work, we aim to extend our idea to in-the-wild videos and apply it to multimodal tasks such as embodied question answering, instruction following, and audio-visual navigation.
6 Reproducibility Statement

We describe our method in Sec. 3 with architecture diagrams and equations to support the textual description. We describe our experimental setup in Sec. 4.1, and provide the necessary hyperparameters in Appendix A5. In Appendix A12, we provide more details about the overall experimental pipeline for EPC and refer to appropriate sections in the main paper and appendix as needed. We will release the source code for generating the walkthroughs, performing EPC pre-training and downstream task finetuning, and the episode datasets for RoomNav upon acceptance.

7 Ethics Statement

Our idea for self-supervised learning relies on having a dataset of video walkthroughs captured in a diverse set of environments. In our experiments, we demonstrate a proof-of-concept by generating these walkthroughs in photorealistic Gibson scenes using simulated agents. However, obtaining such a large-scale dataset of walkthroughs in real-world environments can be challenging due to privacy concerns. To create such a dataset, we would need to carefully consider the privacy of the people and personal belongings recorded as a part of this data.

We perform experiments on the publicly available Gibson and Matterport3D datasets. Since these datasets primarily contain data from work places and economically well-off residences, our results may be influenced by the social and geographical distribution of the scanned buildings in these datasets. In experiments, we demonstrate good generalization of embodied policies trained in MP3D train scenes to novel MP3D test scenes and novel Gibson val scenes. However, it is possible that the generalization may be impacted when the policies are evaluated in environments that are significantly outside the distribution of these datasets.

References


Appendices

We provide additional information about the experimental settings as well as quantitative results to support the experiments from the main paper. Below is a summary of the sections in the Appendix:

- (§A1) Supplementary visualization
- (§A2) Scene memory transformer
- (§A3) Model architectures
- (§A4) Additional task details
- (§A5) Hyperparameters
- (§A6) Sample efficiency on Gibson
- (§A7) Ablations of EPC with strong exploration videos
- (§A8) Complex zone creation schemes for EPC
- (§A9) Noise robustness of learned policies
- (§A10) ANS performance on Flee
- (§A11) Downstream task performance vs. time
- (§A12) EPC pipeline for reproducibility
- (§A13) Relation to visual place recognition

A1 SUPPLEMENTARY VIDEO

The supplementary visualization “epc_visualization_supp.pdf” provides a brief overview of environment predictive coding and the masked-zone prediction task. We also show an example of inter-video retrievals (similar to Fig. 4 from main paper). In the demonstrated example, we show that our EPC model is able to accurately retrieve the masked-zone from the current scene, and even retrieve similar zones from other scenes.

A2 SCENE MEMORY TRANSFORMER

We provide more details about individual components of the Scene Memory Transformer (Fang et al., 2019). As discussed in the main paper, the SMT model consists of a ResNet-18 visual encoder that extracts features $x_t$ from RGB-D inputs, and a scene memory for storing the visual features $\{x_i\}_{i=0}^t$ and agent poses $\{p_i\}_{i=0}^t$ seen during an episode. The environment encoder uses self-attention on the scene memory to generate a richer set of environment embeddings $\{e_i\}_{i=1}^t$. The policy decoder attends to the environment embeddings using the inputs $o_t$ at time $t$, which consist of the visual feature $x_t$, the agent pose $p_t$, and optionally, the navigation goal $g_t$. The outputs of the policy decoder are used to sample an action $a_t$ and estimate the value $v_t$. Next, we discuss the details of the individual components.

**Visual Encoder** At each time step $t$, the visual encoder extracts features $x_t$ from the RGB-D input.$$
 x_t = \text{VisualEncoder}([r_t, d_t])
$$

where $r_t$ and $d_t$ are the RGB and depth inputs, respectively. We consider two visual encoders for our work. The first variant is a modified ResNet-18 encoder from (Wijmans et al., 2020) where the number of output channels are halved and the BatchNorm layers are replaced by GroupNorm. We used this encoder for the scratch models, SMT (MoCo), SMT (Video), and the EPC variants in the main paper. Next, we consider MidLevel features derived from various pre-trained CNNs to solve midlevel perception tasks (Sax et al., 2020; Zamir et al., 2020). For RGB inputs, we extract features from the pre-trained models in the max-coverage set proposed in Sax et al. (2020). These include 4 encoders trained for predicting surface normals, keypoints, semantic segmentation, and 2.5D segmentation. For depth inputs, we extract features from pre-trained models that predict...
surface normals and keypoints from depth (Zamir et al., 2020). These encoders are kept frozen during reinforcement learning, following Sax et al. (2020). MidLevel features were used for SMT (MidLevel) in the main paper. We do not use the MidLevel encoder for EPC since we found the ResNet-18 encoder to be more efficient and achieved better performance.

**Scene Memory** It stores the visual features \( x_t \) derived from the input images and the agent poses \( p_t \) at each time-step \( t \). Following Fang et al. (2019), the pose \( p_t \) consists of the 2D coordinates \((x, y)\) of the agent, heading \( \theta \), and the time \( t \). While we assumed that the ground-truth pose \((x_t, y_t, \theta_t)\) was available for downstream tasks in the main experiments, we evaluate the impact of noisy odometry in Appendix A9.

**Attention Mechanism** Following the notations from Vaswani et al. (2017), we define the attention mechanism used in the environment encoder and policy decoder. Given two inputs \( X \) and \( R \) in Appendix A9, available for downstream tasks in the main experiments, we evaluate the impact of noisy odometry during reinforcement learning, following Sax et al. (2020). MidLevel features were used for SMT (Video), and EPC, we use the same architecture as SMT (scratch) and initialize pre-trained weights according to the method used.

For ANS, Fig. A1 describes a global policy that samples as actions an \((x, y)\) location or STOP. The input occupancy map is a \(300 \times 300 \times 2\) egocentric map where the two channels track occupied

\[
\text{Attn}(X, Y) = \text{softmax} \left( \frac{Q_X K_Y^T}{\sqrt{d_k}} \right) V_Y
\]

where \( Q_X \in \mathbb{R}^{n_1 \times d_x}, K_Y \in \mathbb{R}^{n_2 \times d_y}, V_Y \in \mathbb{R}^{n_2 \times d_y} \) are the queries, keys, and values computed from \( X \) and \( Y \) as follows: \( Q_X = X W^q, K_Y = Y W^k \), and \( V_Y = Y W^v \). \( W^q, W^k, W^v \) are learned weight matrices. The multi-headed version of Attn generates multiple sets of queries, keys, and values to obtain the attended context \( C \in \mathbb{R}^{n_1 \times d_x} \).

\[
\text{MHAttn}(X, Y) = \text{FC}([\text{Attn}^h(X, Y)]_{h=1}^H).
\]

We use the transformer implementation from PyTorch (Paszke et al., 2019). Here, the multi-headed attention block builds on top of MHAttn by using residual connections, LayerNorm (LN) and fully connected (FC) layers to further encode the inputs.

\[
\text{MHAttnBlock}(X, Y) = \text{LN} (\text{MLP}(H) + H)
\]

where \( H = \text{LN}(\text{MHAttn}(X, Y) + X) \), and MLP has 2 FC layers with ReLU activations.

We now describe the memory attention mechanisms used in SMT. At time \( t \), the agent receives features \( x_t \), pose \( p_t \), and optionally the navigation goal \( g_t \). First, we transform the pose vectors \( \{p_i\}_{i=1}^n \) from the scene memory relative to the current agent pose \( p_t \). This allows the agent to maintain an egocentric view of the past inputs (Fang et al., 2019). Next, the environment encoder performs self-attention between the features stored in the scene memory \( M \) to obtain the environment encoding:

\[
\mathcal{E} = \text{EnvironmentEncoder}(M) = \text{MHAttnBlock}(M, M)
\]

Next, the policy decoder attends to the environment encodings \( \mathcal{E} \) using a linear projection of the current observation \( f = \mathcal{M}([x_t, p_t, g_t]) \).

\[
\mathcal{D} = \text{PolicyDecoder}([x_t, p_t, g_t], \mathcal{E}) = \text{MHAttnBlock}(f, \mathcal{E})
\]

**Policy** It consists of two FC layers corresponding to the actor \( \pi \) and the critic \( V \). Given the output \( \mathcal{D} \) of the PolicyDecoder, the policy samples an action \( a_t \sim \pi(\mathcal{D}) \) and predicts the value function \( v_t = V(\mathcal{D}) \). Please see the next section for more details on the exact architectures used.

### A3 Model Architectures

We provide the architectures for the RNN (scratch), SMT (scratch), and ANS baselines in Fig. A1. For Reactive (scratch), we remove the “Transformer Encoder” and “Transformer Decoder” blocks from SMT (scratch) and directly feed the input features to the policy. For SMT (MidLevel), we replace the ResNet encoder with the MidLevel encoders discussed in Sec. A2. For SMT (MoCo), SMT (Video), and EPC, we use the same architecture as SMT (scratch) and initialize pre-trained weights according to the method used.

For ANS, Fig. A1 describes a global policy that samples as actions an \((x, y)\) location or STOP. The input occupancy map is a \(300 \times 300 \times 2\) egocentric map where the two channels track occupied
Figure A1: **Baseline architectures**. We show the detailed architectures for each of the key baselines. “GoalEncoder” is a 1-layer MLP. M is a 2-layer MLP. Embed, FC, Conv, GroupNorm, and ReLU are PyTorch layers corresponding to nn.Embedding(), nn.Linear(), nn.Conv2d(), nn.GroupNorm(), and nn.ReLU(), respectively (Paszke et al., 2019). The architectures shown are used for the RoomNav task. For the remaining tasks, we remove the room goal “Embed” and “GoalEncoder” layers and keep the rest of the architecture unchanged.

and explored regions. It captures a $48m \times 48m$ region around the agent. A local policy, which is a low-level navigator, then navigates to the $(x, y)$ location for 25 steps. We use an analytical mapper+planner as the local policy as this was found to perform as well as a learned policy (Chaplot et al., 2020b;a). For area coverage, flee, and object visitation, we use a $48m \times 48m$ action space for the global policy. We try both discrete $(24 \times 24)$ actions and continuous action spaces, and pick the best choice based on the validation performance. For RoomNav, we use a discrete action space and add STOP as an additional action, giving us $24 \times 24 + 1 = 577$ actions.

### A4 ADDITIONAL TASK DETAILS

We provide more details about the object visitation and RoomNav tasks below.

**OBJECT VISITATION** To determine if an object is visited, we check if it is within 1.5m of the agent, present in the agent’s field of view, and if it is not occluded (Ramakrishnan et al., 2021). We use a shaped reward function that rewards the agent for visiting a new object category and a cell-coverage reward to encourage exploration (similar to Fang et al. (2019)):

$$R_t = O_t - O_{t-1} + 0.02(C_t - C_{t-1}),$$

(11)

where $O_t, C_t$ are the number of object categories and 2D grid-cells visited by time $t$. For MP3D, we use all 21 object categories defined for the ObjectNav task in Habitat (Batra et al., 2020; Savva et al., 2019a):

- chair, table, picture, cabinet, cushion, sofa, bed, chest of drawers,
- plant, sink, toilet, stool, towel, tv monitor, shower, bathtub, counter,
- fireplace, gym equipment, seating, clothes

For Gibson, we use the semantic annotations from the 3D Scene Graph dataset (Armeni et al., 2019). We evaluate on the following categories:
We report both the number of categories and instances of the above objects visited. For MP3D, we evaluate on 550/900 validation/test episodes. For Gibson, we evaluate on 300 validation episodes. We do not split Gibson semantic results into small/large since there are only 6 val semantic scenes.

**ROOMNAV** For an agent to successfully reach a specified room category, it needs to be within 0.1m of a navigable location inside any room belonging to that category. We use a shaped reward function (Savva et al., 2019a):

\[ R_t = d_t - 1 - d_t + 2.5S_t - 0.001 \]  

where \( d_t \) is the distance to the nearest target room location, \( S_t \) indicates success at time \( t \), and 0.001 is a slack penalty. We evaluate on Gibson and MP3D rooms of the following categories: bathroom, bedroom, office, kitchen, living room, dining room.

We measure both success rate and SPL, i.e., success weighted by path length (Anderson et al., 2018a). We calculate SPL similar to (Batra et al., 2020), where the shortest path length is defined based on the target room closest to the agent’s starting point. We evaluate on 410/1600 val/test episodes on MP3D, and 598 val episodes on Gibson. We use 500-step episodes following Narasimhan et al. (2020) (unlike 1000-step episodes for other tasks on MP3D).

**A5 HYPERPARAMETERS**

All models are trained in PyTorch (Paszke et al., 2019) with DD-PPO (Wijmans et al., 2020) for 13M-15M frames with 60 parallel processes. Since SMT (MoCo), SMT (Video) and EPC benefit from 2M frames of off-policy experience in the video walkthroughs, we train the scratch and ANS baselines for 2M more frames to account for this. Note that SMT (MidLevel) is already pre-trained on 4M frames of annotated data.

We detail the list of hyperparameter choices for different tasks and models in Tab. A1. For ANS, we use 4 PPO mini-batches, 4 PPO epochs (not 2), and entropy coefficient of 0.003. For SMT (Video), we randomly sample 40 consecutive frames in the video and predict the final 15 frames from the initial 25 frames (following time-spans from Han et al. (2019)). For EPC, we randomly mask out \( m \) zones of size \( N \) in the video and predict them from the remaining video. We selected these values based on a grid-search and the validation performance on downstream tasks. For area coverage, flee, and object visitation, we found \( N = 5, m = 6 \) to work best. For RoomNav, we found \( N = 5 - 40, m = 6 \) to work best on weak exploration videos, and \( N = 40, m = 6 \) to work best on strong exploration videos. The hyperparameter search results are shown in Figs. A2, A3.

**A6 SAMPLE EFFICIENCY ON GIBSON**

We plot the Gibson validation performance as a function of training experience in Fig. A4. EPC achieves 2 - 6× higher sample efficiency through environment-level pre-training when compared to the only image-level pre-training from SMT (MoCo).

**A7 ABLATIONS OF EPC WITH STRONG EXPLORATION VIDEOS**

In Sec. 4.3 from the main paper, we presented an ablation study of EPC (W.E). We now provide the corresponding results for EPC (S.E) in Tab. A2. Our conclusions remain the same as in Sec. 4.3.

1. EPC needs spatial conditioning, i.e., to query using pose during SSL (instead of only time).
2. EPC is robust to noise in depth and odometer sensors used for collecting the video walkthroughs.
3. EPC benefits from global input context during SSL (instead of a local input context of 25 frames).

**A8 COMPLEX ZONE CREATION SCHEMES FOR EPC**

Over the course of arriving at our final EPC zone creation scheme, we tried an alternative scheme that attempted to strictly minimize the overlap between the seen and unseen zones. We measured...
Table A1: Hyperparameters for training our RL and self-supervised learning models. * - we use 4 PPO mini-batches and 4 PPO epochs for ANS. We disable normalized advantage and use 500-step episodes for RoomNav.

<table>
<thead>
<tr>
<th>RL Optimization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.00025</td>
</tr>
<tr>
<td># parallel actors</td>
<td>60/64</td>
</tr>
<tr>
<td>PPO mini-batches</td>
<td>2*</td>
</tr>
<tr>
<td>PPO epochs</td>
<td>2*</td>
</tr>
<tr>
<td>PPO clip param</td>
<td>0.2</td>
</tr>
<tr>
<td>Value loss coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>Entropy coefficient</td>
<td>0.01</td>
</tr>
<tr>
<td>Advantage estimation</td>
<td>GAE</td>
</tr>
<tr>
<td>Normalized advantage?</td>
<td>Yes*</td>
</tr>
<tr>
<td>Training episode length</td>
<td>1000*</td>
</tr>
<tr>
<td>GRU history length</td>
<td>128</td>
</tr>
<tr>
<td># training steps (in millions)</td>
<td>13-15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SMT hyperparameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden size</td>
<td>128</td>
</tr>
<tr>
<td>Scene memory length</td>
<td>500</td>
</tr>
<tr>
<td>Scene memory stride</td>
<td>2/4</td>
</tr>
<tr>
<td># attention heads</td>
<td>8</td>
</tr>
<tr>
<td># encoder layers</td>
<td>1</td>
</tr>
<tr>
<td># decoder layers</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Self-supervised learning optimization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.0001</td>
</tr>
<tr>
<td>Video batch size</td>
<td>10</td>
</tr>
<tr>
<td>Temperature ($\tau$)</td>
<td>0.1</td>
</tr>
</tbody>
</table>
the 3D point-cloud overlap between every pair of frames in a video walkthrough (projected using depth, camera pose, and camera intrinsics), and clustered the frames that have significant overlap together using hierarchical agglomerative clustering (Lukasová, 1979). We defined each cluster to be a zone, and used the same SSL training pipeline from Sec. 3. Tab. A3 shows the results on the MP3D validation split on 3 tasks. The spatial overlap variant works reasonably well. However, it is significantly more complex than our original proposal (requiring camera intrinsics, 3D point-cloud overlap computations and clustering), and performs worse. Since the videos are unconstrained, the agent might revisit the same areas multiple times, causing too many frames to be grouped together into a single zone. This results in very irregular zone sizes for masked-zone prediction across videos. This affects the quality of EPC SSL training. By not strictly limiting the overlap between zones, the agent learns by both inferring seen features (but from different viewpoints), as well as unseen features. Our current method is simpler, and works better.

### A9 Noise robustness of learned policies

In the main paper, we assumed the availability of ground-truth depth and pose sensors for downstream tasks (In Tab. 2, we added pose and depth noise to the walkthrough videos only). Now, we relax these assumptions and re-evaluate all methods by injecting noise in the depth and odometer sensors for downstream tasks (same noise models from prior work that we applied in Sec. 4.3), without any noise-correction. This is a common evaluation protocol for assessing noise robustness (Chen et al., 2019; Ramakrishnan et al., 2021; Chen et al., 2021). Note that the noise in agent’s pose estimate increases over time since the per-step noise in odometer readings are accumulated. We compare the noise-robustness of all methods in Tab. A4. As expected, the performance declines as we add noise to more sensors (depth, then pose). We observe larger deterioration in performance when noise is added to depth. Note that this is likely due to the domain shift in the depth inputs between noise-free training and noisy evaluation. Nevertheless, most approaches are reasonably stable. EPC outperforms all methods when all noise sources are added. ANS declines rapidly in the absence of noise-correction due to accumulated map errors.
### A10 ANS performance on Flee

ANS relies on a global policy that samples a spatial goal location for navigation. A local navigation policy then executes a series of low-level actions to reach that goal. Our qualitative analyses indicate that the global policy overfit to large MP3D environments. It often samples far away exploration targets, relying on the local navigator to explore the spaces along the sampled direction. However, this strategy fails in the small Gibson-S environments (typically a single room). Selecting far away targets results in the local navigator oscillating in place trying to exit a single-room environment. This does not affect area coverage much because it suffices to stand in the middle of a small room and look at all sides.

### A11 Downstream task performance vs. time

We show the downstream task performance as a function of time in Fig. A5. For each model, we train with 2 random seeds and evaluate with 3 random seeds. We report the mean and the 95% confidence interval in the plots. EPC converges faster than using only image-level pretraining in SMT (MoCo), and outperforms a state-of-the-art Active Neural SLAM (ANS) on most tasks.

### A12 EPC pipeline for reproducibility

We provide more details regarding the EPC pipeline for pre-training and downstream task evaluation in order to aid reproducibility. We will release the source code upon acceptance.

The EPC pipeline involves the following key steps.

1. **Generating video walkthroughs:** We deploy weak and strong exploration agents in Gibson 2+ training environments (Wijmans et al., 2020) to gather 2 sets of video walkthroughs.
2. **Pre-training image-level representations:** We pre-train a ResNet-18 encoder using MoCo v2 (Chen et al., 2020b) on images sampled from walkthroughs.
3. **Extracting features for video walkthroughs:** Using the pre-trained ResNet-18 encoder, we extract visual features for each image all video walkthroughs.
4. **Pre-training environment-level features via EPC:** Given the dataset of video walkthrough features + agent camera poses, we pre-train the environment encoder and projection MLP $M$.
5. **Transferring to downstream tasks:** We transfer both the image-level and environment-level encoders to an SMT agent and finetune it end-to-end a navigation task.

We will now detail each of the above individual steps.

**Generating video walkthroughs** We use the 332 Gibson 2+ training scenes defined in Wijmans et al. (2020) for generating video walkthroughs. These walkthroughs consist of RGB-D and odometry readings for each step. We use two types of agents to generate video walkthroughs (see Sec. 3.3). The strong exploration agent is a SMT (scratch) model trained to perform area coverage on 61 MP3D training scenes for 10M frames. The weak exploration agent is a simple forward-biased heuristic that keeps moving forward till colliding. Upon collision, it rotates left or right by a randomly sampled angle between $[0^\circ, 150^\circ]$, and then continues to move forward. Given an environment, each...
agent is spawned on a set of different locations and executed for 500 steps from each start location. We select the start locations by clustering the navigable locations within the environment into \( K \) clusters using K-means clustering. The number of clusters \( K \) is decided based on the navigable area in each environment. Across all the Gibson 2+ train environments, we generate 2 sets of 5,047 video walkthroughs (one set per exploration agent). We randomly divide each set of videos into an 80-20 train/val split.

**Pre-training image-level representations** To learn a strong image-level encoder from the video walkthroughs, we randomly sample RGB-D images from the walkthroughs and train a ResNet-18 encoder using MoCo-v2 for 7,000 epochs. Specifically, we sample a batch of \( N = 128 \) videos, and sample \( M = 4 \) frames per video. The frames are uniformly spread across the video. This gives us a batch of 512 images for each MoCo-v2 update. We use the publicly available MoCo-v2 implementation from https://github.com/facebookresearch/moco. Since this image-level pre-training corresponds to the important SMT (MoCo) baseline, we provide it the benefit of using strong exploration walkthroughs.

**Extracting features for video walkthroughs** For lower memory consumption and faster training, we first pre-extract the visual features for each image in the video walkthroughs dataset. We extract features using the ResNet-18 encoder pre-trained using MoCo-v2 on the strong exploration walkthroughs. We use the same image encoder for extracting features from both the weak and strong exploration walkthroughs. We observed in experiments that the quality of the image-encoder did not vary significantly based on the walkthroughs used. By using a single pre-trained image encoder for the EPC variants, we can consistently evaluate the impact of walkthrough quality on EPC pre-training. Note that the image encoders are kept frozen during EPC pre-training.

**Pre-training environment-level features via EPC** Having extracted the visual features for the walkthrough dataset (either strong or weak exploration), we perform the environment-level pre-training of the environment-encoder \( E \) and projection MLP \( M \) (see Fig. 2 from the main paper). We use our proposed masked-zone prediction task for self-supervised pre-training. Given a video walkthrough of length \( L \), we divide it into zones \( \{Z_0, Z_1, \cdots\} \) of length \( N \) as indicated in Eqn. 1 from the main paper. We randomly mask \( m \) zones (i.e., the unseen zones) from the video, and learn representations by predicting the features for an each unseen zone \( Z_u^i \). Given the list of visual features and camera poses \((x, p)\) sampled from the seen zones, we transform the poses \( p \) relative to the average camera pose \( p_u^i \) from zone \( Z_u^i \). We then transform each \((x, p)\) using the projection MLP \( M \). The outputs of the projection MLP \( M \) are used as inputs for the environment encoder which generates rich environment-embeddings \( E \) using self-attention. We then use the environment embeddings \( E \) and the average camera pose \( p_u^i \) as inputs to the zone decoder, which predicts the zone feature \( f_u^i \). The target feature for the zone is the average of MLP projected features from the zone (see Eqn. 2). The model is trained using the contrastive loss in Eqn. 4. The hyperparameters are provided in Appendix A5.

**Transferring to downstream tasks** For transferring the EPC pre-trained encoders, we need to transfer both the image-level ResNet-18 encoder pre-trained on MoCo-v2, and most importantly, the environment encoder pre-trained on EPC. The SMT model is initialized using the image and environment encoders and fine-tuned end-to-end of the downstream navigation task of interest. Please refer to Sec. 3.2 from the main paper for details regarding the transfer. For area coverage, flee, and object coverage, we use the starting positions from the PointNav-v1 dataset available on https://github.com/facebookresearch/habitat-lab. For RoomNav, we create our own habitat-compatible datasets for Gibson and MP3D which we will release along with the source code.

### A13 Relation to visual place recognition

Prior work on visual place recognition build maps and recognize whether the current visual information is from a place recorded in the map (Lowry et al., 2015). A place could be an exact location (Kuipers, 2000), rooms (Kuipers, 2000), or locally distinctive regions (Kuipers & Byun, 1991; Bailey et al., 1999). Zones in EPC can be viewed as a specific ‘place’ in 3D environments. However,
instead of learning place-level features for recognizing previously seen places (Chen et al., 2017), we learn environment-level features by inferring features for unseen places.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Gibson-S</th>
<th></th>
<th>Gibson-L</th>
<th></th>
<th>MP3D</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Methods</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
</tr>
<tr>
<td></td>
<td>Reactive (scratch)</td>
<td>28.2</td>
<td>29.9</td>
<td>30.2</td>
<td>50.1</td>
<td>49.2</td>
<td>50.8</td>
<td>121.8</td>
<td>119.5</td>
</tr>
<tr>
<td></td>
<td>RNN (scratch)</td>
<td>30.8</td>
<td>33.0</td>
<td>33.1</td>
<td>57.6</td>
<td>56.4</td>
<td>56.9</td>
<td>130.1</td>
<td>106.6</td>
</tr>
<tr>
<td></td>
<td>SMT (scratch)</td>
<td>32.2</td>
<td>34.5</td>
<td>34.6</td>
<td>62.6</td>
<td>62.8</td>
<td>62.1</td>
<td>106.5</td>
<td>153.8</td>
</tr>
<tr>
<td></td>
<td>SMT (MidLevel)</td>
<td>31.4</td>
<td>33.2</td>
<td>33.0</td>
<td>57.1</td>
<td>54.5</td>
<td>53.7</td>
<td>147.2</td>
<td>143.0</td>
</tr>
<tr>
<td></td>
<td>SMT (MoCo)</td>
<td>32.4</td>
<td>34.5</td>
<td>35.1</td>
<td>60.4</td>
<td>66.4</td>
<td>60.2</td>
<td>177.5</td>
<td>152.5</td>
</tr>
<tr>
<td></td>
<td>SMT (Video)</td>
<td>31.9</td>
<td>34.3</td>
<td>34.7</td>
<td>61.3</td>
<td>61.9</td>
<td>59.4</td>
<td>157.1</td>
<td>138.5</td>
</tr>
<tr>
<td></td>
<td>ANS</td>
<td>33.3</td>
<td>35.6</td>
<td>33.3</td>
<td>81.5</td>
<td>84.4</td>
<td>62.7</td>
<td>172.8</td>
<td>171.7</td>
</tr>
<tr>
<td></td>
<td>EPC (W.E)</td>
<td>33.9</td>
<td>36.4</td>
<td>36.0</td>
<td>76.1</td>
<td>77.0</td>
<td>75.6</td>
<td>195.1</td>
<td>179.7</td>
</tr>
<tr>
<td></td>
<td>EPC (S.E)</td>
<td>33.7</td>
<td>36.5</td>
<td>36.4</td>
<td>75.8</td>
<td>77.1</td>
<td>75.7</td>
<td>198.2</td>
<td>183.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Gibson-S</th>
<th></th>
<th>Gibson-L</th>
<th></th>
<th>MP3D</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Methods</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
</tr>
<tr>
<td></td>
<td>Reactive (scratch)</td>
<td>3.0</td>
<td>3.0</td>
<td>3.1</td>
<td>4.0</td>
<td>4.1</td>
<td>4.1</td>
<td>6.6</td>
<td>7.8</td>
</tr>
<tr>
<td></td>
<td>RNN (scratch)</td>
<td>5.3</td>
<td>5.0</td>
<td>5.0</td>
<td>8.1</td>
<td>7.5</td>
<td>7.4</td>
<td>12.0</td>
<td>11.1</td>
</tr>
<tr>
<td></td>
<td>SMT (scratch)</td>
<td>4.3</td>
<td>4.2</td>
<td>4.2</td>
<td>6.4</td>
<td>6.5</td>
<td>6.5</td>
<td>11.0</td>
<td>10.8</td>
</tr>
<tr>
<td></td>
<td>SMT (MidLevel)</td>
<td>4.8</td>
<td>4.6</td>
<td>4.5</td>
<td>6.7</td>
<td>6.1</td>
<td>6.1</td>
<td>11.2</td>
<td>10.8</td>
</tr>
<tr>
<td></td>
<td>SMT (MoCo)</td>
<td>5.1</td>
<td>5.1</td>
<td>5.0</td>
<td>8.0</td>
<td>8.0</td>
<td>8.0</td>
<td>13.1</td>
<td>12.3</td>
</tr>
<tr>
<td></td>
<td>SMT (Video)</td>
<td>4.9</td>
<td>4.9</td>
<td>4.4</td>
<td>7.7</td>
<td>7.6</td>
<td>7.0</td>
<td>11.8</td>
<td>10.9</td>
</tr>
<tr>
<td></td>
<td>ANS</td>
<td>3.6</td>
<td>3.5</td>
<td>3.0</td>
<td>7.7</td>
<td>7.8</td>
<td>6.3</td>
<td>13.0</td>
<td>12.7</td>
</tr>
<tr>
<td></td>
<td>EPC (W.E)</td>
<td>5.9</td>
<td>5.9</td>
<td>6.0</td>
<td>10.6</td>
<td>10.6</td>
<td>10.5</td>
<td>16.6</td>
<td>15.7</td>
</tr>
<tr>
<td></td>
<td>EPC (S.E)</td>
<td>6.0</td>
<td>5.9</td>
<td>5.9</td>
<td>10.3</td>
<td>10.2</td>
<td>10.0</td>
<td>15.8</td>
<td>15.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Gibson-S</th>
<th></th>
<th>Gibson-L</th>
<th></th>
<th>MP3D</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Methods</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
</tr>
<tr>
<td></td>
<td>Reactive (scratch)</td>
<td>6.4</td>
<td>4.0</td>
<td>6.2</td>
<td>3.8</td>
<td>6.3</td>
<td>3.9</td>
<td>12.1</td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>RNN (scratch)</td>
<td>6.7</td>
<td>4.2</td>
<td>6.4</td>
<td>4.1</td>
<td>6.3</td>
<td>4.0</td>
<td>13.8</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>SMT (scratch)</td>
<td>9.2</td>
<td>5.4</td>
<td>8.8</td>
<td>5.1</td>
<td>8.6</td>
<td>5.1</td>
<td>17.8</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td>SMT (MidLevel)</td>
<td>7.6</td>
<td>4.6</td>
<td>7.4</td>
<td>4.4</td>
<td>7.4</td>
<td>4.4</td>
<td>15.2</td>
<td>5.7</td>
</tr>
<tr>
<td></td>
<td>SMT (MoCo)</td>
<td>9.1</td>
<td>5.3</td>
<td>8.8</td>
<td>5.3</td>
<td>8.8</td>
<td>5.3</td>
<td>18.8</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td>SMT (Video)</td>
<td>8.7</td>
<td>5.1</td>
<td>8.3</td>
<td>5.0</td>
<td>8.2</td>
<td>5.0</td>
<td>16.4</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>ANS</td>
<td>10.0</td>
<td>5.9</td>
<td>9.2</td>
<td>5.5</td>
<td>6.6</td>
<td>4.1</td>
<td>20.6</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td>EPC (W.E)</td>
<td>11.9</td>
<td>6.5</td>
<td>11.3</td>
<td>6.3</td>
<td>11.2</td>
<td>6.3</td>
<td>24.0</td>
<td>7.6</td>
</tr>
<tr>
<td></td>
<td>EPC (S.E)</td>
<td>12.0</td>
<td>6.6</td>
<td>11.9</td>
<td>6.5</td>
<td>11.8</td>
<td>6.5</td>
<td>24.5</td>
<td>7.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Gibson-S</th>
<th></th>
<th>Gibson-L</th>
<th></th>
<th>MP3D</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Methods</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
<td>N-D,P</td>
<td>NF</td>
<td>N-D</td>
</tr>
<tr>
<td></td>
<td>Reactive (scratch)</td>
<td>0.4</td>
<td>1.5</td>
<td>0.3</td>
<td>1.0</td>
<td>0.3</td>
<td>1.1</td>
<td>0.2</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>RNN (scratch)</td>
<td>2.2</td>
<td>6.8</td>
<td>2.3</td>
<td>6.2</td>
<td>2.1</td>
<td>5.6</td>
<td>1.6</td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td>SMT (scratch)</td>
<td>2.3</td>
<td>5.0</td>
<td>2.1</td>
<td>4.9</td>
<td>1.7</td>
<td>3.8</td>
<td>1.4</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>SMT (MidLevel)</td>
<td>4.8</td>
<td>10.4</td>
<td>4.3</td>
<td>10.5</td>
<td>4.5</td>
<td>9.9</td>
<td>2.3</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>SMT (MoCo)</td>
<td>8.9</td>
<td>20.1</td>
<td>7.8</td>
<td>18.1</td>
<td>7.7</td>
<td>17.4</td>
<td>5.5</td>
<td>12.7</td>
</tr>
<tr>
<td></td>
<td>SMT (Video)</td>
<td>7.1</td>
<td>18.1</td>
<td>6.1</td>
<td>16.3</td>
<td>4.8</td>
<td>12.5</td>
<td>5.0</td>
<td>12.0</td>
</tr>
<tr>
<td></td>
<td>ANS</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>EPC (W.E)</td>
<td>12.6</td>
<td>28.9</td>
<td>12.2</td>
<td>27.2</td>
<td>11.0</td>
<td>24.0</td>
<td>9.7</td>
<td>21.1</td>
</tr>
<tr>
<td></td>
<td>EPC (S.E)</td>
<td>13.9</td>
<td>31.5</td>
<td>12.5</td>
<td>28.2</td>
<td>10.9</td>
<td>23.7</td>
<td>8.9</td>
<td>19.8</td>
</tr>
</tbody>
</table>

Table A4: Comparing robustness to sensor noise on downstream tasks in Gibson and Matterport3D. We inject noise into the depth sensor and/or the odometer sensor. The depth-noise model consists of disparity-based quantization, high-frequency noise, and low-frequency distortion (Choi et al., 2015). The odometry-noise model is based on data collected from a LoCoBot, and accumulates over time leading to a drift in pose (Ramakrishnan et al., 2020; Chaplot et al., 2020b). Note: NF denotes noise free sensing, N-D denotes noisy depth (and noise-free pose), and N-D,P denotes noisy depth and pose.
Figure A5: We highlight the downstream task performance as a function of episode time on both Matterport3D and Gibson. The legend shown near the top two rows also applies to the bottom two rows. Note that the maximum episode length on Gibson scenes is $T = 500$. For MP3D, we use a maximum episode length of $T = 1000$ for all tasks except RoomNav where we use $T = 500$ (following Narasimhan et al. (2020)).