

# Scene Memory Transformer for Embodied Agents in Long-Horizon Tasks

Kuan Fang<sup>1</sup>Alexander Toshev<sup>2</sup>Li Fei-Fei<sup>1</sup>Silvio Savarese<sup>1</sup><sup>1</sup>Stanford University<sup>2</sup>Google Brain

## Abstract

*Many robotic applications require the agent to perform long-horizon tasks in partially observable environments. In such applications, decision making at any step can depend on observations received far in the past. Hence, being able to properly memorize and utilize the long-term history is crucial. In this work, we propose a novel memory-based policy, named Scene Memory Transformer (SMT). The proposed policy embeds and adds each observation to a memory and uses the attention mechanism to exploit spatio-temporal dependencies. This model is generic and can be efficiently trained with reinforcement learning over long episodes. On a range of visual navigation tasks, SMT demonstrates superior performance to existing reactive and memory-based policies by a margin.*

## 1. Introduction

Autonomous agents, controlled by neural network policies and trained with reinforcement learning algorithms, have been used in a wide range of robot navigation applications [1, 2, 3, 23, 28, 32, 47, 50, 51]. In many of these applications, the agent needs to perform tasks over long time horizons in unseen environments. Consider a robot patrolling or searching for an object in a large unexplored building. Typically, completing such tasks requires the robot to utilize the received observation at each step and to grow its knowledge of the environment, *e.g.* building structures, object arrangements, explored area, *etc.* Therefore, it is crucial for the agent to maintain a detailed memory of past observations and actions over the task execution.

The most common way of endowing an agent’s policy with memory is to use recurrent neural networks (RNNs), with LSTM [5] as a popular choice. An RNN stores the information in a fixed-size state vector by combining the input observation with the state vector at each time step. The policy outputs actions for the agent to take given the updated state vector. Unfortunately, however, RNNs often fail to capture long-term dependencies [34].

To enhance agent’s ability to plan and reason, neural network policies with external memories have been pro-

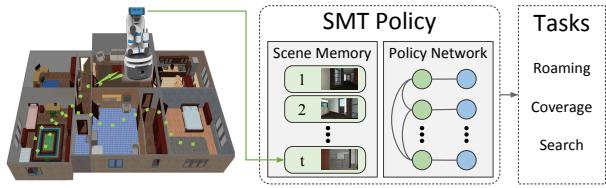


Figure 1. The Scene Memory Transformer (SMT) policy embeds and adds each observation to a memory. Given the current observation, the attention mechanism is applied over the memory to produce an action. SMT is demonstrated successfully in several visual navigation tasks, all of which has long time horizons.

posed [32, 49]. Such memory-based policies have been primarily studied in the context of robot navigation in partially observable environments, where the neural network learns to encode the received observations and write them into a map-like memory [16, 17, 19, 33, 44]. Despite their superior performance compared to reactive and RNN policies, existing memory-based policies suffer from limited flexibility and scalability. Specifically, strong domain-specific inductive biases go into the design of such memories, *e.g.* 2D layout of the environment, predefined size of this layout, geometry-based memory updates, *etc.* Meanwhile, RNNs are usually critical components for these memory-based policies for exploiting spatio-temporal dependencies. Thus they still suffer from the drawbacks of RNN models.

In this work, we present *Scene Memory Transformer (SMT)*, a memory-based policy using attention mechanisms, for understanding partially observable environments in long-horizon robot tasks. This policy is inspired by the Transformer model [43], which has been successfully applied to multiple natural language processing problems recently. As shown in Fig. 1, SMT consists of two modules: a scene memory which embeds and stores all encountered observations and a policy network which uses attention mechanism over the scene memory to produce an action.

The proposed SMT policy is different from existing methods in terms of how to utilize observations received in the previous steps. Instead of combining past observations into a single state vector, as commonly done by RNN policies, SMT separately embeds the observations for each time step in the scene memory. In contrast to most existing

memory models, the scene memory is simply a set of all embedded observations and any decisions of aggregating the stored information are deferred to a later point. We argue that this is a crucial property in long-horizon tasks where computation of action at a specific time step could depend on any provided information in the past, which might not be properly captured in a state vector or map-like memory. The policy network in SMT adopts attention mechanisms instead of RNNs to aggregate the visual and geometric information from the scene memory. This network efficiently learns to utilize the stored information and scales well with the time horizon. As a result, SMT effectively exploits long-term spatio-temporal dependencies without committing to a environment structure in the model design.

Although the scene memory grows linearly with the length of the episode, it stores only an embedding vector at each steps. Therefore, we can easily store hundreds of observations without any burden in the device memory. This overhead is justified as it gives us higher performance compared to established policies with more compact memories.

Further, as the computational complexity of the original model grows quadratically with the size of the scene memory, we introduce a memory factorization procedure as part of SMT. This reduces the computational complexity to linear. The procedure is applied when the number of the stored observations is high. In this way, we can leverage a large memory capacity without the taxing computational overhead of the original model.

The advantages of the proposed SMT are empirically verified on three long-horizon visual navigation tasks: roaming, coverage and search. We train the SMT policy using deep Q-learning [30] and thus demonstrate for the first time how attention mechanisms introduced in [43] can boost the task performance in a reinforcement learning setup. In these tasks, SMT considerably and consistently outperforms existing reactive and memory-based policies. Videos can found at <https://sites.google.com/view/scene-memory-transformer>

## 2. Related Work

**Memory-based policy using RNN.** Policies using RNNs have been extensively studied in reinforcement learning settings for robot navigation and other tasks. The most common architectural choice is LSTM [20]. For example, Mirowski *et al.* [28] train an A3C [29] agent to navigate in synthesized mazes with an LSTM policy. Wu *et al.* [46] use a gated-LSTM policy with multi-modal inputs trained for room navigation. Moursavian *et al.* [31] use an LSTM policy for target driven navigation. The drawbacks of RNNs are mainly two-fold. First, merging all past observations into a single state vector of fixed size can easily lose useful information. Second, RNNs have optimization difficulties over long sequences [34, 42] in backpropagation

through time (BPTT). In contrast, our model stores each observations separately in the memory and only aggregate the information when computing an action. And it extracts spatio-temporal dependencies using attention mechanisms, thereby it is not handicapped by the challenges of BPTT.

**External memory.** Memory models have been extensively studied in natural language processing for variety of tasks such as translation [43], question answering [39], summarization [27]. Such models are fairly generic, mostly based on attention functions and designed to deal with input data in the format of long sequences or large sets .

When it comes to autonomous agents, most of the approaches structure the memory as a 2D grid. They are applied to visual navigation [16, 17, 33, 48], interactive question answering [14], and localization [7, 19, 21]. These methods exhibit certain rigidity. For instance, the 2D layout is of fixed size and same amount of memory capacity is allocated to each part of the environment. Henriques *et al.* [19] designs a differentiable mapping module with 2.5D representation of the spatial structure. Such a structured memory necessitates write operations, which compress all observations as the agent executes a task and potentially can lose information which could be useful later in the task execution. On the contrary, our SMT keeps all embedded observations and allows for the policy to attend to them as needed at any step. Further, the memory operations in the above papers are based on current estimate of robot localization, where the memory is being modified and how it is accessed. In contrast, we keep all pose information in its original form, thus allow for potentially more flexible use.

A more generic view on memory for autonomous agents has been less popular. Savinov *et al.* [36] construct a topological map of the environment, and uses it for planning. Oh at al. [32] use a single-layer attention decoder for control problems. However, the method relies on an LSTM as a memory controller, which comes with the challenges of backpropagation through time. Khan *et al.* [24] apply the very general Differentiable Neural Computer [15] to control problems. While this approach is hard to optimize and is applied on very simple navigation tasks.

**Visual Navigation.** We apply SMT on a set of visual navigation tasks, which have a long history in computer vision and robotics [6, 10, 40]. Our approach falls into visual navigation, where the agent does not have any scene-specific information about the environment [8, 9, 37, 41, 45]. As in recent works on end-to-end training policies for navigation tasks [2, 23, 28, 47, 50, 51], our model does not need a map of the environment provided beforehand. While [23, 28] evaluates their models in 3D mazes, our model can handle more structured environments as realistic cluttered indoor scenes composed of multiple rooms. In contrast to [28, 50, 51] which train the policy for one or several known scenes, our trained model can generalize to unseen houses.

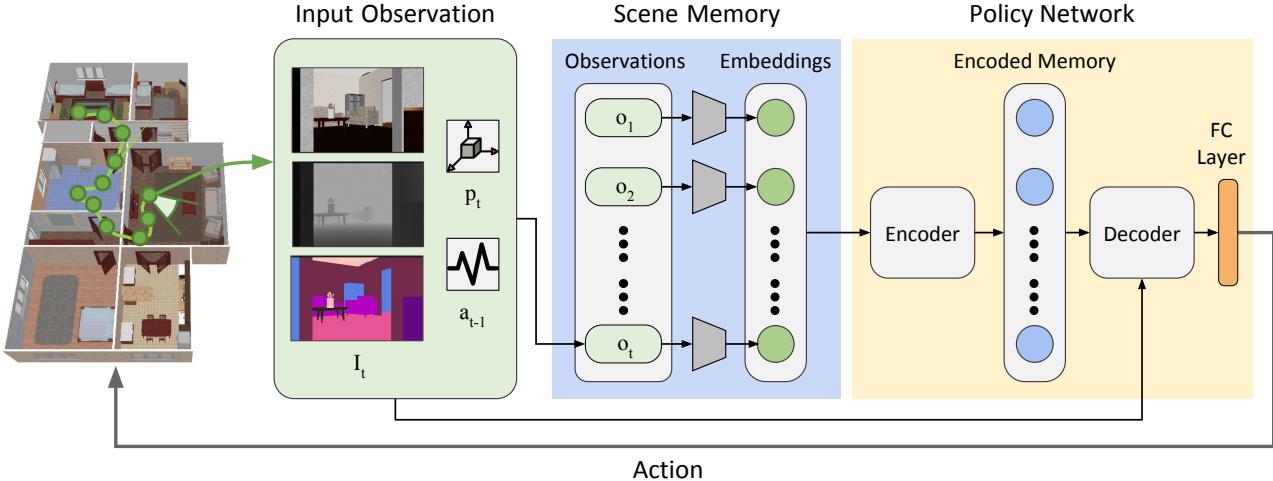


Figure 2. The **Scene Memory Transformer (SMT)** policy. At each time step  $t$ , the observation  $\mathbf{o}_t$  is embedded and added to the scene memory. SMT has access to the full memory and produces an action according to the current observation.

### 3. Method

In this section, we first describe the problem setup. Then we introduce the Scene Memory Transformer (SMT) and its variations as shown in Fig. 2.

#### 3.1. Problem Setup

We are interested in a variety of tasks which require an embodied agent to navigate in unseen environments to achieve the task goal. These tasks can be formulated as the Partially Observable Markov Decision Process (POMDP) [22]  $(\mathcal{S}, \mathcal{A}, \mathcal{O}, R(s, a), T(s'|s, a), P(o|s))$  where  $\mathcal{S}$ ,  $\mathcal{A}$ ,  $\mathcal{O}$  are state, action and observation spaces,  $R(s, a)$  is the reward function,  $T(s'|s, a)$  and  $P(o|s)$  are transition and observation probabilities.

The observation is a tuple  $\mathbf{o} = (I, p, a_{\text{prev}}) \in \mathcal{O}$  composed of multiple modalities.  $I$  represents the visual data consisting of an RGB image, a depth image and a semantic segmentation mask obtained from a camera sensor mounted on the robot.  $p$  is the agent pose w.r.t. the starting pose of the episode, estimated or given by the environment.  $a_{\text{prev}}$  is the action taken at the previous time step.

In our setup, we adopt a discrete action space defined as  $\mathcal{A} = \{\text{go\_forward}, \text{turn\_left}, \text{turn\_right}\}$ , a common choice for navigation problems operating on a flat surface. Note that these actions are executed under noisy dynamics modeled by  $P(s'|s, a)$ , so the state space is continuous.

While we share the same  $\mathcal{O}$  and  $\mathcal{A}$  across tasks and environments, each task is defined by a different reward function  $R(s, a)$  as described in Sec. 4.1. The policy for each task is trained to maximize the expected return, defined as the cumulative reward  $\mathbb{E}_{\tau}[\sum_t R(s_t, a_t)]$  over trajectories  $\tau = (s_t, a_t)_{t=1}^H$  of time horizon  $H$  unrolled by the policy.

#### 3.2. Scene Memory Transformer

The SMT policy, as outlined in Fig. 2, consists of two modules. The first module is the *scene memory*  $M$  which stores all past observations in an embedded form. This memory is updated at each time step. The second module, denoted by  $\pi(a|o, M)$ , is an attention-based *policy network* that uses the updated scene memory to compute an distribution over actions.

In a nutshell, the model and its interaction with the environment at time  $t$  can be summarized as:

$$\begin{aligned} o_t &\sim P(o_t|s_t) \\ M_t &= \text{Update}(M_{t-1}, o_t) \\ a_t &\sim \pi(a_t|o_t, M_t) \\ s_{t+1} &\sim T(s_{t+1}|s_t, a_t) \end{aligned}$$

In the following we define the above modules.

##### 3.2.1 Scene Memory

The scene memory  $M$  is intended to store all past observations in an embedded form. It is our intent not to endow it with any geometric structure, but to keep it as generic as possible. Moreover, we would like to avoid any loss of information when writing to  $M$  and provide the policy with all available information from the history. So we separately keep observations of each step in the memory instead of merging them into a single state vector as in an RNN.

The scene memory can be defined recursively as follows. Initially it is set to the empty set  $\emptyset$ . At the current step, given an observation  $\mathbf{o} = (I, p, a_{\text{prev}})$ , as defined in Sec. 3.1, we first embed all observation modalities, concat-

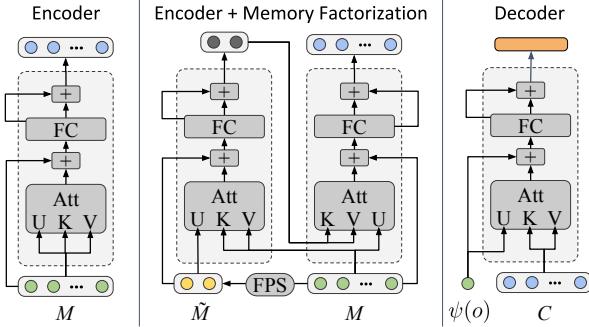


Figure 3. Encoder without memory factorization, encoder with memory factorization, and decoder as in Sec. 3.2.2.

nate them, and apply a fully-connected layer FC:

$$\psi(o) = \text{FC}(\{\phi_I(I), \phi_p(p), \phi_a(a_{\text{prev}})\}) \quad (1)$$

where  $\phi_I, \phi_p, \phi_a$  are embedding networks for each modality as defined in Sec. 3.4. To obtain the memory for the next step, we update it by adding  $\psi(o)$  to the set:

$$\text{Update}(M, o) = M \cup \{\psi(o)\} \quad (2)$$

The above memory grows linearly with the episode length. As each received observation is embedded into low-dimensional vectors in our design, one can easily store hundreds of time steps on the hardware devices. While RNNs are restricted to a fixed-size state vector, which usually can only capture short-term dependencies.

### 3.2.2 Attention-based Policy Network

The policy network  $\pi(a|o, M)$  uses the current observation and the scene memory to compute a distribution over the action space. As shown in Fig. 2, we first *encode* the memory by transforming each memory element in the context of all other elements. This step has the potential to capture the spatio-temporal dependencies in the environment. Then, we *decode* an action according to the current observation, using the encoded memory as the context.

**Attention Mechanism.** Both encoding and decoding are defined using attention mechanisms, as detailed by [43]. In its general form, the attention function Att applies  $n_1$  attention queries  $U \in \mathbb{R}^{n_1 \times d_k}$  over  $n_2$  values  $V \in \mathbb{R}^{n_2 \times d_v}$  with associated keys  $K \in \mathbb{R}^{n_2 \times d_k}$ , where  $d_k$  and  $d_v$  are dimensions of keys and values. The output of Att has  $n_1$  elements of dimension  $d_v$ , defined as a weighted sum of the values, where the weights are based on dot-product similarity between the queries and the keys:

$$\text{Att}(U, K, V) = \text{softmax}(U K^T) V \quad (3)$$

An attention block AttBlock is built upon the above function and takes two inputs  $X \in \mathbb{R}^{n_1 \times d_x}$  and  $Y \in \mathbb{R}^{n_2 \times d_y}$  of dimension  $d_x$  and  $d_y$  respectively. It projects  $X$  to the

queries and  $Y$  to the key-value pairs. It consists of two residual layers. The first is applied to the above Att and the second is applied to a fully-connected layer:

$$\text{AttBlock}(X, Y) = \text{LN}(\text{FC}(H) + H) \quad (4)$$

$$\text{where } H = \text{LN}(X W^U, Y W^K, Y W^V) + X$$

where  $W^U \in \mathbb{R}^{d_x \times d_k}$ ,  $W^K \in \mathbb{R}^{d_y \times d_k}$  and  $W^V \in \mathbb{R}^{d_y \times d_v}$  are projection matrices and LN stands for layer normalization [4]. We choose  $d_v = d_x$  for the residual layer.

**Encoder.** As in [43], our SMT model uses self-attention to encode the memory  $M$ . More specifically, we use  $M$  as both inputs of the attention block. As shown in Fig. 3, this transforms each embedded observation by using its relations to other past observations:

$$\text{Encoder}(M) = \text{AttBlock}(M, M) \quad (5)$$

In this way, the model extracts the spatio-temporal dependencies in the memory.

**Decoder.** The decoder is supposed to produce actions based on the current observation given the context  $C$ , which in our model is the encoded memory. As shown in Fig. 3, it applies similar machinery as the encoder, with the notable difference that the query in the attention layer is the embedding of the current observation  $\psi(o)$ :

$$\text{Decoder}(o, C) = \text{AttBlock}(\psi(o), C) \quad (6)$$

The final SMT output is a probability distribution over the action space  $\mathcal{A}$ :

$$\pi(a|o, M) = \text{Cat}(\text{softmax}(Q)) \quad (7)$$

$$\text{where } Q = \text{FC}(\text{FC}(\text{Decoder}(o, \text{Encoder}(M))))$$

where Cat denotes categorical distribution.

This gives us a stochastic policy from which we can sample actions. Empirically, this leads to more stable behaviors, which avoids getting stuck in suboptimal states.

**Discussion.** The above SMT model is based on the encoder-decoder structure introduced in the Transformer model, which has seen successes on natural language processing (NLP) problems such as machine translation, text generation and summarization. The design principles of the model, supported by strong empirical results, transfer well from the NLP domain to the robot navigation setup, which is the primary motivation for adopting it.

First, an agent moving in a large environment has to work with dynamically growing number of past observations. The encoder-decoder structure has shown strong performance exactly in the regime of lengthy textual inputs. Second, contrary to common RNNs or other structured external memories, we do not impose a predefined order or structure on the memory. Instead, we encode temporal and spatial information as part of the observation and let the policy learn to interpret the task-relevant information through the attention mechanism of the encoder-decoder structure.

### 3.2.3 Memory Factorization

The computational complexity of the SMT is dominated by the number of query-key pairs in the attention mechanisms. Specifically, the time complexity is  $O(|M|^2)$  for the encoder due to the self-attention, and  $O(|M|)$  for the decoder. In long-horizon tasks, where the memory grows considerably, quadratic complexity can be prohibitive. Inspired by [26], we replace the self-attention block from Eq. (4) with a composition of two blocks of similar design but more tractable computation:

$$\text{AttFact}(M, \tilde{M}) = \text{AttBlock}(M, \text{AttBlock}(\tilde{M}, M)) \quad (8)$$

where we use a “compressed” memory  $\tilde{M}$  obtained via finding representative centers from  $M$ . These centers need to be dynamically updated to maintain a good coverage of the stored observations. In practice, we can use any clustering algorithm. For the sake of efficiency, we apply iterative farthest point sampling (FPS) [35] to the embedded observations in  $M$ , in order to choose a subset of elements which are distant from each other in the feature space. The running time of FPS is in  $O(|M||\tilde{M}|)$  and the final complexity of AttFact is  $O(|M||\tilde{M}|)$ . With a fixed number of centers, the overall time complexity becomes linear. The diagram of the encoder with memory factorization is shown in Fig. 3.

### 3.3. Training

We train all model variants and baselines using the standard deep Q-learning algorithm [30]. We follow [30] in the use of an experience replay buffer, which has a capacity of 1000 episodes. The replay buffer is initially filled with episodes collected by a random policy and is updated every 500 training iterations. The update replaces the oldest episode in the buffer with a new episode collected by the updated policy. At every training iteration, we construct a batch of 64 episodes randomly sampled from the replay buffer. The model is trained with Adam Optimizer [25] with a learning rate of  $5 \times 10^{-4}$ . All model parameters except for the embedding networks are trained end-to-end. During training, we continuously evaluate the updated policy on the validation set (as in Sec. 4.1). We keep training each model until we observe no improvement on the validation set.

The embedding networks are pre-trained using the SMT policy with the same training setup, with the only difference that the memory size is set to be 1. This leads to a SMT with no attention layers, as attention of size 1 is an identity mapping. In this way, the optimization is made easier so that the embedding networks can be trained end-to-end. After being trained to convergence, the parameters of the embedding networks are frozen for other models.

A major difference to RNN policies or other memory-based policies is that SMT does not need backpropagation through time (BPTT). As a result, the optimization is more

stable and less computationally heavy. This enables training the model to exploit longer temporal dependencies.

### 3.4. Implementation Details

Image modalities are rendered as  $640 \times 480$  and sub-sampled by 10. Each image modality is embedded into 64-dimensional vectors using a modified ResNet-18 [18]. We reduce the numbers of filters of all convolutional layers by a factor of 4 and use stride of 1 for the first two convolutional layers. We remove the global pooling to better capture the spatial information and directly apply the fully-connected layer at the end. Both pose and action vectors are embedded using a single 16-dimensional fully-connected layer.

Attention blocks in SMT use multi-head attention mechanisms [43] with 8 heads. The keys and values are both 128-dimensional. All the fully connected layers in the attention blocks are 128-dimensional and use ReLU non-linearity.

A special caution is to be taken with the pose vector. First, at every time step all pose vectors in the memory are transformed to be in the coordinate system defined by the current agent pose. This is consistent with an ego-centric representation of the memory. Thus, the pose observations need to be re-embedded at every time step, while all the other observations are embedded once. Second, a pose vector  $p = (x, y, \theta)$  at time  $t$  is converted to a normalized version  $p = (x/\lambda, y/\lambda, \cos \theta, \sin \theta, e^{-t})$ , embedding in addition its temporal information  $t$  in a soft way in its last dimension. This allows the model to differentiate between recent and old observation, assuming that former could be more important than latter. The scaling factor  $\lambda = 5$  is used to reduce the magnitude of the coordinates.

## 4. Experiments

We design our experiments to investigate the following topics: 1) How well does SMT perform on different long-horizon robot tasks 2) How important is its design properties compared to related methods? 3) Qualitatively, what agent behaviors does SMT learn?

### 4.1. Task Setup

To answer these questions, we consider three visual navigation tasks: *roaming*, *coverage*, and *search*. These tasks require the agent to summarize spatial and semantic information of the environment across long time horizons. All tasks share the same POMDP from Sec. 3.1 except that the reward functions are defined differently in each task.

**Roaming:** The agent attempts to move forward as much as possible without colliding. In this basic navigation task, a memory should help the agent avoid cluttered areas and oscillating behaviors. The reward is defined as  $R(s, a) = 1$  iff  $a = \text{go\_forward}$  and no collision occurs.

**Coverage:** In many real-world application a robot needs to explore unknown environments and visit all areas of these

environments. This task clearly requires a detailed memory as the robot is supposed to remember all places it has visited. To define the coverage task, we overlay a grid of cell size 0.5 over the floorplan of each environment. We would like the agent to visit as many unoccupied cells as possible, expressed by reward  $R(s, a) = 5$  iff robot entered unvisited cell after executing the action.

**Search:** To evaluate whether the policy can learn beyond knowledge about the geometry of the environment, we define a semantic version of the coverage tasks. In particular, for six target object classes<sup>1</sup>, we want the robot to search for as many as possible of them in the house. Each house contains 1 to 6 target object classes, 4.9 classes in average. Specifically, an object is marked as found if more than 4% of pixels in an image has the object label (as in [46]) and the corresponding depth values are less than 2 meter. Thus,  $R(s, a) = 100$  iff after taking action  $a$  we find one of the six object classes which hasn't been found yet.

We add a collision reward of  $-1$  for each time the agent collides. An episode will be terminated if the agent runs into more than 50 collisions. To encourage exploration, we add coverage reward to the search task with a weight of 0.2.

The above tasks are listed in ascending order of complexity. The coverage and search tasks are studied in robotics, however, primarily in explored environments and are concerned about optimal path planning [13].

**Environment.** We use SUNCG [38], a set of synthetic but visually realistic buildings. We use the same data split as chosen by [46] and remove the houses with artifacts, which gives us 195 training houses and 46 testing houses. We hold out 20% of the training houses as a validation set for ablation experiments. We run 10 episodes in each house with a fixed random seed during testing and validation. The agent moves by a constant step size of 0.25 meters with `go_forward`. It turns by  $45^\circ$  degree in place with `turn_left` or `turn_right`. Gaussian noise is added to simulate randomness in real-world dynamics.

**Model Variants.** To investigate the effect of different model aspects, we conduct experiments with three variants: **SMT**, **SMT + Factorization**, and **SM + Pooling**. The second model applies SMT with AttFact instead of AttBlock. Inspired by [12], the last model directly applies a max pooling over the elements in the scene memory instead of using the encoder-decoder structure of SMT.

**Baselines.** We use the following baselines for comparison. A **Random** policy uniformly samples one of the three actions. A **Reactive** policy is trained to directly compute Q values using a purely feedforward net. It is two fully-connected layers on top of the embedded observation at every step. A **LSTM** policy [28] is the most common memory-based policy. A model with arguably larger capacity, called **FRMQN** [32], maintains embedded observations

<sup>1</sup>We use television, refrigerator, bookshelf, table, sofa, and bed.

| Method              | Reward       | Distance     | Collisions  |
|---------------------|--------------|--------------|-------------|
| Random              | 58.3         | 25.3         | 42.7        |
| Reactive [28]       | 308.9        | 84.6         | 29.3        |
| LSTM [28]           | 379.7        | 97.9         | <b>11.4</b> |
| FRMQN [32]          | 384.2        | 99.5         | 13.8        |
| SM + Pooling        | 366.8        | 96.7         | 20.1        |
| SMT + Factorization | 376.4        | 98.6         | 17.9        |
| SMT                 | <b>394.7</b> | <b>102.1</b> | 13.6        |

Table 1. **Performance on Roaming.** The average of cumulative reward, roaming distance and number of collisions are listed.

| Method              | Reward       | Covered Cells |
|---------------------|--------------|---------------|
| Random              | 94.2         | 27.4          |
| Reactive [28]       | 416.2        | 86.9          |
| LSTM [28]           | 418.1        | 87.8          |
| FRMQN [32]          | 397.7        | 83.2          |
| SM + Pooling        | 443.9        | 91.5          |
| SMT + Factorization | 450.1        | 99.3          |
| SMT                 | <b>474.6</b> | <b>102.5</b>  |

Table 2. **Performance on Coverage.** The average of cumulative reward and number of covered cells are listed.

| Method              | Reward       | Classes     | Ratio        |
|---------------------|--------------|-------------|--------------|
| Random              | 140.5        | 1.79        | 36.3%        |
| Reactive [28]       | 358.2        | 3.14        | 61.9%        |
| LSTM [28]           | 339.4        | 3.07        | 62.6%        |
| FRMQN [32]          | 411.2        | 3.53        | 70.2%        |
| SM + Pooling        | 332.5        | 2.98        | 60.6%        |
| SMT + Factorization | <b>432.6</b> | <b>3.69</b> | <b>75.0%</b> |
| SMT                 | 428.4        | 3.65        | 74.2%        |

Table 3. **Performance on Search.** The cumulative of total reward, number of found classes and ratio of found classes are listed.

in a fixed-sized memory, similarly as SMT. Instead of using the encode-decoder structure to exploit the memory, it uses an LSTM, whose input is current observation and output is used to attend over the memory.

For all methods, we use the same pretrained embedding networks and two fully-connected layers to compute Q values. We also use the same batch size of 64 during training. To train LSTM and FRMQN, we use truncated back propagation through time of 128 steps.

## 4.2. Comparative Evaluation

The methods are compared across the three different tasks: roaming in Table 1, coverage in Table 2, and search in Table 3. For each task and method we show the attained reward and task specific metrics.

**Effect of memory designs.** Across all tasks, SMT outperforms all other memory-based models. The relative performance gain compared to other approaches is most significant for coverage (14% improvements) and considerable for search (5% improvements). This is consistent with the

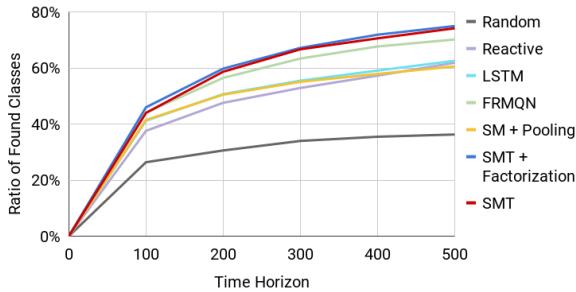


Figure 4. **Found classes by time steps.** For the search task, we show number of found target object classes across time steps.

notion that for coverage and search memorizing all past observations is more vital. On roaming, larger memory capacity (in SMT case) helps, however, all memory-based approaches perform in the same ballpark. This is reasonable in the sense that maintaining a straight collision free trajectory is a relatively short-sight task.

In addition to memory capacity, memory access via attention brings improvements. For all tasks SMT outperforms SM + Pooling. The gap is particularly large for object search (Table 3), where the task has an additional semantic complexity of finding objects. Similarly, having multi-headed attention and residual layers brings an improvement over a basic attention, as employed by FRMQN, which is demonstrated on both coverage and search.

The proposed memory factorization brings computational benefits, at no or limited performance loss. Even if it causes drop sometimes, the reward is better than SM + Pooling and other baseline methods.

**Implications of memory for navigation.** It is also important to understand how memory aids us at solving navigation tasks. For this purpose, in addition to reward, we report number of covered cells (Table 2) and number of found objects (Table 3). For both tasks, a reactive policy presents a strong baseline, which we suspect learns general exploration principles. Adding memory via SMT helps boost the coverage by 18% over reactive, and 17% over LSTM policies and 23% over simpler memory mechanism (FRMQN). We also observe considerable boosts of number of found objects by 5% in the search task.

The reported metrics above are for a fixed time horizon of 500 steps. For varying time horizons, we show the performance on search in Fig. 4. We see that memory-based policies with attention-based reads consistently find more object classes as they explore the environment, with SMT variants being the best. This is true across the full execution with performance gap increasing steadily up to 300 steps.

### 4.3. Ablation Analysis

Here, we analyze two aspects of SMT: (i) size of the scene memory, and (ii) importance of the different observation modalities and components.

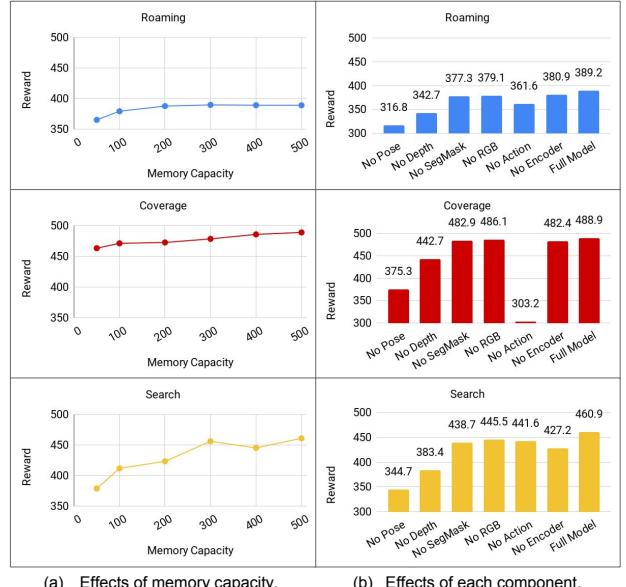


Figure 5. **Ablation Experiments.** (a) We sweep the memory capacity from 50 steps to 500 steps and evaluate the reward of trajectories of 500 steps. (b) We leave out one component at a time in our full model and evaluate the averaged reward for each task.

**Memory capacity.** While in the previous section we discussed memory capacity across models, here we look at the importance of memory size for SMT. Intuitively a memory-based policy is supposed to benefit more from larger memory over long time horizons. But in practice this depends on the task and the network capacity, as shown in Fig. 5 (a). All three tasks benefit from using larger scene memory. The performance of roaming grows for memory up to 300 elements. For coverage and search, the performance keeps improving constantly with larger memory capacities. This shows that SMT does leverage the provided memory.

**Modalities and components.** For the presented tasks, we have image observations, pose, previous actions. To understand their importance, we re-train SMT by leaving out one modality at a time. We show the resulting reward in Fig. 5 (b). Among the observation modalities, last action, pose and the depth image play crucial roles across tasks. This is probably because SMT uses relative pose and last action to reason about spatial relationships. Further, depth image is the strongest clue related to collision avoidance, which is crucial for all tasks. Removing segmentation and RGB observations leads to little effect on coverage and drop of 10 for roaming since these tasks are defined primarily by environment geometry. For search, however, where SMT needs to work with semantics, the drop is 15 and 20.

We also show that the encoder structure brings performance boost to the tasks. Especially in the search task, which is most challenging in terms of reasoning and planning, the encoder boosts the task reward by 23.7.

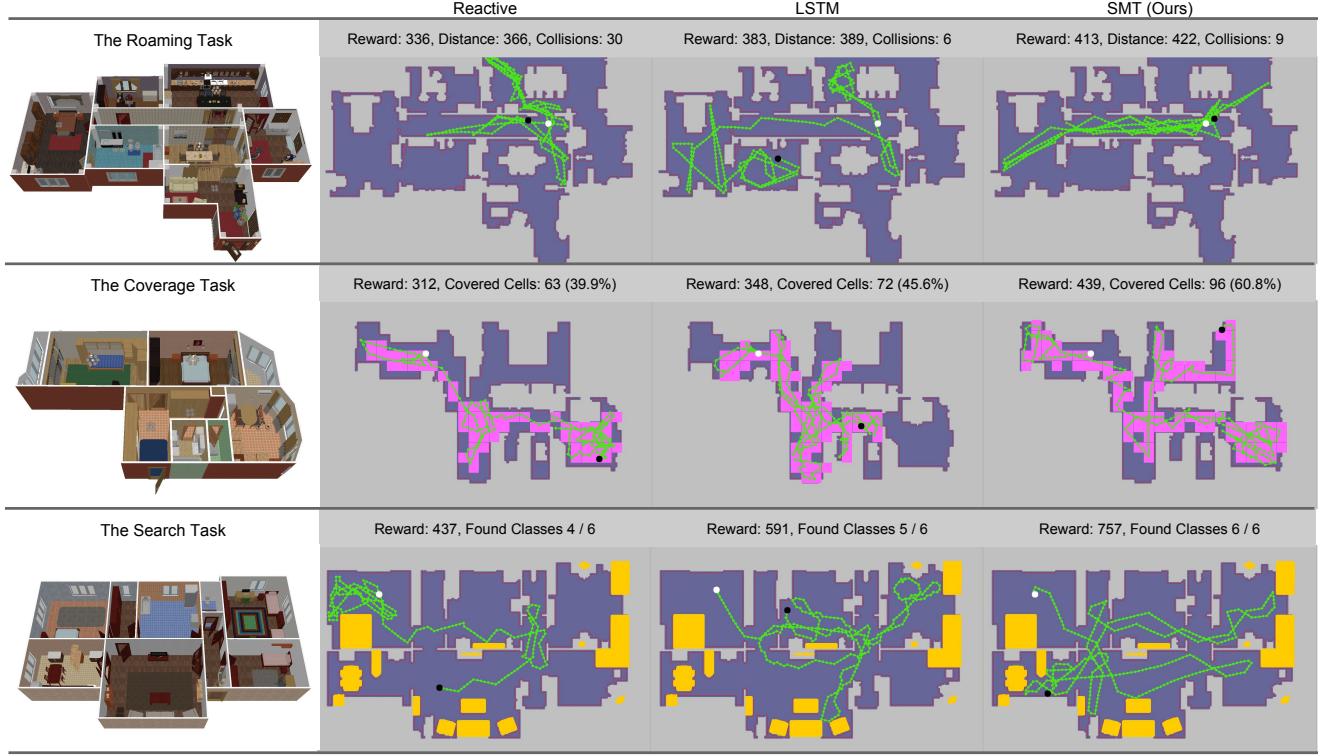


Figure 6. **Visualization of the agent behaviors.** We visualize the trajectories from the top-down view as green curves. Starting point and ending point of each trajectory are plot in white and black. Navigable area are masked in dark purple with red lines indicating the collision boundaries. For the coverage task, we mark the covered cells in pink. For the search task, we mark target objects with yellow masks.

#### 4.4. Qualitative Results

To better understand the learned behaviors of the agent, we visualize the navigation trajectories in Fig. 6. We choose reactive and LSTM policies as representatives of memoryless and memory-based baselines to compare with SMT.

In the roaming task, our model demonstrates better strategies to keep moving and avoid collisions. In many of the cases, the agent first finds a long clear path in the house, which lets it go straight forward without frequently making turns. Then the agent navigates back and forth along the same route until the end of the episode. As a result, SMT usually leads to compact trajectories as shown in Fig. 6, top row. In contrast, reactive policy and LSTM policy often wander around the scene with a less consistent pattern.

In the coverage task, our model explores the unseen space more efficiently by memorizing regions that have been covered. As shown in Fig. 6, middle row, after most of the cells inside a room being explored, the agent switches to the next unvisited room. Note that the cells are invisible to the agent, it needs to make this decision solely based on its memory and observation of the environment. It also remembers better which rooms have been visited so that it does not enter a room twice.

In the search task, our model shows efficient exploration

as well as effective strategies to find the target classes. The search task also requires the agent to explore rooms with the difference that the exploration is driven by target object classes. Therefore, after entering a new room, the agent quickly scans around the space instead of covering all the navigable regions. In Fig. 6, if the agent finds the unseen target it goes straight towards it. Once it is done, it will leave the room directly. Comparing SMT with baselines, our trajectories are straight and direct between two targets, while baseline policies have more wandering patterns.

#### 5. Conclusion

This paper introduces Scene Memory Transformer, a memory-based policy to aggregate observation history in robotic tasks of long time horizons. We use attention mechanism to exploit spatio-temporal dependencies across past observations. The policy is trained on several visual navigation tasks using deep Q-learning. Evaluation shows that the resulting policy achieves higher performance to other established methods.

**Acknowledgement:** We thank Anelia Angelova, Ashish Vaswani and Jakob Uszkoreit for constructive discussions. We thank Marek Fišer for the software development of the simulation environment, Oscar Ramirez and Ayzaan Wahid for the support of the learning infrastructure.

## References

- [1] P. Anderson, A. X. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir. On evaluation of embodied navigation agents. *CoRR*, abs/1807.06757, 2018. [1](#)
- [2] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018. [1](#), [2](#)
- [3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017. [1](#)
- [4] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. [4](#)
- [5] B. Bakker. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*, pages 1475–1482, 2002. [1](#)
- [6] F. Bonin-Font, A. Ortiz, and G. Oliver. Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 53:263–296, 2008. [2](#)
- [7] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov. Active neural localization. *arXiv preprint arXiv:1801.08214*, 2018. [2](#)
- [8] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *ICCV*, 2003. [2](#)
- [9] F. Dayoub, T. Morris, B. Upcroft, and P. Corke. Vision-only autonomous navigation using topometric maps. In *Intelligent robots and systems (IROS), 2013 IEEE/RSJ international conference on*, pages 1923–1929. IEEE, 2013. [2](#)
- [10] G. N. DeSouza and A. C. Kak. Vision for mobile robot navigation: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 24(2):237–267, 2002. [2](#)
- [11] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2650–2658, 2015. [11](#)
- [12] S. M. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. C. Rabinowitz, H. King, C. Hillier, M. M. Botvinick, D. Wierstra, K. Kavukcuoglu, and D. Hassabis. Neural scene representation and rendering. *Science*, 360:1204–1210, 2018. [6](#)
- [13] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013. [6](#)
- [14] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. Iqa: Visual question answering in interactive environments. *arXiv preprint arXiv:1712.03316*, 1, 2017. [2](#)
- [15] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016. [2](#)
- [16] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *CVPR*, pages 7272–7281, 2017. [1](#), [2](#)
- [17] S. Gupta, D. F. Fouhey, S. Levine, and J. Malik. Unifying map and landmark based representations for visual navigation. *CoRR*, abs/1712.08125, 2017. [1](#), [2](#)
- [18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. [5](#)
- [19] J. F. Henriques and A. Vedaldi. Mapnet : An allocentric spatial memory for mapping environments. In *CVPR*, 2018. [1](#), [2](#)
- [20] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. [2](#)
- [21] R. Jonschkowski, D. Rastogi, and O. Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. *arXiv preprint arXiv:1805.11122*, 2018. [2](#)
- [22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101:99–134, 1998. [3](#)
- [23] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016. [1](#), [2](#)
- [24] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee. Memory augmented control networks. *arXiv preprint arXiv:1709.05706*, 2017. [2](#)
- [25] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2015. [5](#)
- [26] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh. Set transformer. *CoRR*, abs/1810.00825, 2018. [5](#), [11](#)
- [27] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia

- by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018. 2
- [28] P. W. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. *CoRR*, abs/1611.03673, 2016. 1, 2, 6
- [29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016. 2
- [30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. 2, 5
- [31] A. Mousavian, A. Toshev, M. Fiser, J. Kosecka, and J. Davidson. Visual representations for semantic target driven navigation. *arXiv preprint arXiv:1805.06066*, 2018. 2
- [32] J. Oh, V. Chockalingam, S. P. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. In *ICML*, 2016. 1, 2, 6
- [33] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *CoRR*, abs/1702.08360, 2017. 1, 2
- [34] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013. 1, 2
- [35] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 5
- [36] N. Savinov, A. Dosovitskiy, and V. Koltun. Semiparametric topological memory for navigation. *International Conference on Learning Representations*, 2018. 2
- [37] R. Sim and J. J. Little. Autonomous vision-based exploration and mapping using hybrid maps and rao-blackwellised particle filters. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2082–2089, 2006. 2
- [38] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. *CVPR*, 2017. 6
- [39] S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015. 2
- [40] S. Thrun. Simultaneous localization and mapping. In *Robotics and cognitive approaches to spatial mapping*, pages 13–41. Springer, 2007. 2
- [41] M. Tomono. 3-d object map building using dense object models with sift-based recognition features. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1885–1890, 2006. 2
- [42] T. H. Trinh, A. M. Dai, T. Luong, and Q. V. Le. Learning longer-term dependencies in rnns with auxiliary losses. In *ICML*, 2018. 2
- [43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. 1, 2, 4, 5, 11, 12
- [44] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18:620–634, 2010. 1
- [45] D. Wooden. A guide to vision-based map building. *IEEE Robotics & Automation Magazine*, 13:94–98, 2006. 2
- [46] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building generalizable agents with a realistic and rich 3d environment. *CoRR*, abs/1801.02209, 2018. 2, 6
- [47] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. *CoRR*, abs/1808.10654, 2018. 1, 2
- [48] J. Zhang, L. Tai, J. Boedecker, W. Burgard, and M. Liu. Neural slam. *arXiv preprint arXiv:1706.09520*, 2017. 2
- [49] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel. Learning deep neural network policies with continuous memory states. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 520–527. IEEE, 2016. 1
- [50] Y. Zhu, D. Gordon, E. Kolve, D. Fox, L. Fei-Fei, A. Gupta, R. Mottaghi, and A. Farhadi. Visual semantic planning using deep successor representations. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 483–492, 2017. 1, 2
- [51] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364, 2017. 1, 2

## A. Environment Details

In all experiments, we simulate a mobile base of the Fetch robot. The Fetch robot receives visual observations from a Primesense Carmine 1.09 short-range RGBD sensor mounted on its head. Accordingly, we render images of  $640 \times 480$  resolution. To simulate the operation range of the depth sensor, we only render depth values for points that are within 5 meters from the camera. We also provide a binary mask indicating which pixels have valid depth values and concatenate the mask with the depth image as its second channel. We also add zero-mean Gaussian noise of with a standard deviation of 0.05 meter to each pixel. The segmentation mask uses the class labels from NYU40 [11] with each pixel label encoded in the one-hot manner. We subsample the rendered images by a factor of 10, providing us RGB images of  $64 \times 48 \times 3$ , depth images of  $64 \times 48 \times 2$  and segmentation masks of  $64 \times 48 \times 40$ .

The environment dynamics is simulated for the Fetch robot operating on a planar surface. The robot moves forward and take turns by controlling the velocity of its two wheels, with a wheel radius of 0.065 meters and axis width of 0.375 meters. We add a zero-mean Gaussian with a standard deviation of 0.5 rad/s to both wheels to simulate the noisy dynamics. We check the collisions between the robot and the meshes of the environment. The robot will be reset to the previous pose when it collides by taking the action.

## B. Analysis of Memory Factorization

In memory factorization, it is crucial to choose representative centers that have a good coverage of all past observations. Therefore, the centers should be distant from each other in the feature space. Since the memory keeps growing across time, the centers are supposed to be dynamically updated during the task execution instead of remaining as static vectors for all episodes.

In this section, we compare the **farthest point sampling (FPS)** used in SMT with two alternative types of representative centers. We refer to **Window** as the baseline which uses the last  $|\tilde{M}|$  time steps in a fixed time window as representative centers. In this way, the centers are dynamically updated but only focus on the most recent history. We also implemented the static inducing points in [26], which we refer to as **Static**. The  $|\tilde{M}|$  inducing points are trained as neural network weights and remain static during test time. We compare the performance of the three types of centers on the validation set by setting  $|\tilde{M}|$  to be 100. As shown in Table. 4, FPS achieves comparable task performance with Static in the roaming task. And it outperforms the two baselines in coverage and search.

| Center Type | Roaming      | Coverage     | Search       |
|-------------|--------------|--------------|--------------|
| Window      | 378.0        | 451.6        | 438.7        |
| Static      | <b>383.9</b> | 457.96       | 445.9        |
| FPS         | 383.3        | <b>481.2</b> | <b>462.7</b> |

Table 4. Performance of using different types of representative center in memory factorization. Average rewards are listed.

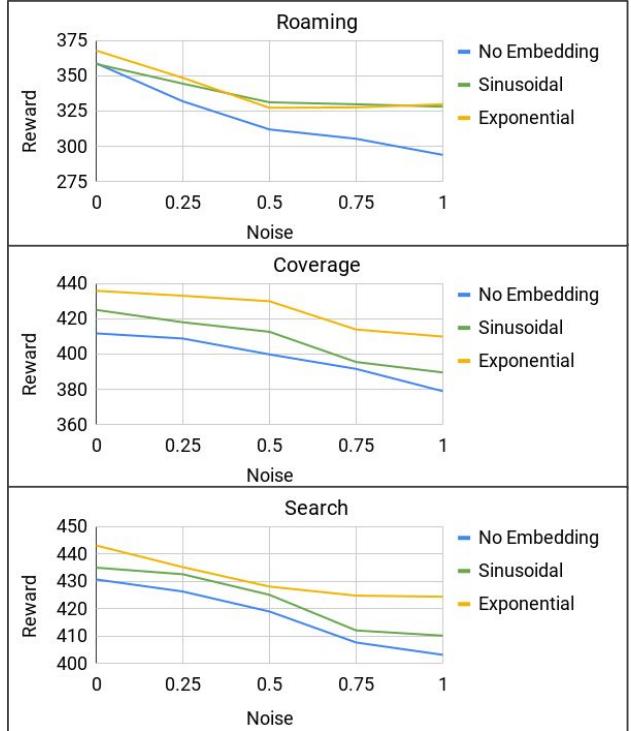


Figure 7. Robustness to noisy dynamics. We compare three positional embedding methods under noisy environment dynamics. The standard deviation of the noise is swept from 0.0 to 1.0.

## C. Robustness to Noisy Dynamics

In this section, we evaluate the robustness of our model to noisy environment dynamics. Instead of retrieving the ground truth poses  $p_t$  from the environment, we estimate the pose using the action  $a_t$ .  $a_t$  provides us translation and rotation of the agent w.r.t. the previous pose. Thus we can estimate the  $\hat{p}_{t+1}$  at each time step using  $a_t$  and the previous estimation  $\hat{p}_t$ . When there is no noise,  $\hat{p}_t$  is equivalent to  $p_t$ . With the Gaussian noise added at each time step, the noise added to  $\hat{p}_t$  will be a Gaussian process. Therefore, when computing the observation embedding using the relative poses, recent steps suffer less from the noisy dynamics.

In our design of SMT, we use a positional embedding of the time step similar to [43], but with exponential functions instead of sinusoidal functions. The positional embedding provides temporal information of each time step for the policy. Sinusoidal function is periodic and provides only relative temporal information. In contrast, the exponential func-

tion is monotonic and represents how recent each time step is. In the long-horizon tasks we are interested in, we believe relative temporal information is not sufficient for the agent to understand long-term dependencies.

To validate this assumption, we compare the exponential embedding with the two baselines. **No embedding** does not embed the positional embedding of the time step. **Sinusoidal** uses the same sinusoidal embedding function as in [43]. We sweep the standard deviation of the noise from 0.0 to 1.0 and evaluate the average rewards on the validation set. In practice, we found the temporal information not only improves the performance given clean observations, but also helps leverage the noisy environment dynamics across time. As shown in Fig. 7, the average rewards decrease with more noises in dynamics. Sinusoidal and exponential embeddings both mitigate the performance drop. In the roaming task, the two embedding methods have comparable effects. While in coverage and search, exponential embedding has the superior performance.

## D. More Visualization

We present more visualization of the agent behaviors for roaming in Fig. 8, for coverage in Fig. 9 and for search in Fig. 10. As in the main paper, we visualize the trajectories from the top-down view as green curves, with white and black dots indicating the starting and ending points. Navigable area are masked in dark purple with red lines as the collision boundaries. In the coverage task (Fig. 9), we mark the covered cells in pink. In the search task (Fig. 10), we mark target objects with yellow masks. These figures demonstrate similar behaviors as analyzed in the main paper. The same reactive and LSTM baselines as in the main paper are used to compare with the proposed SMT policy.

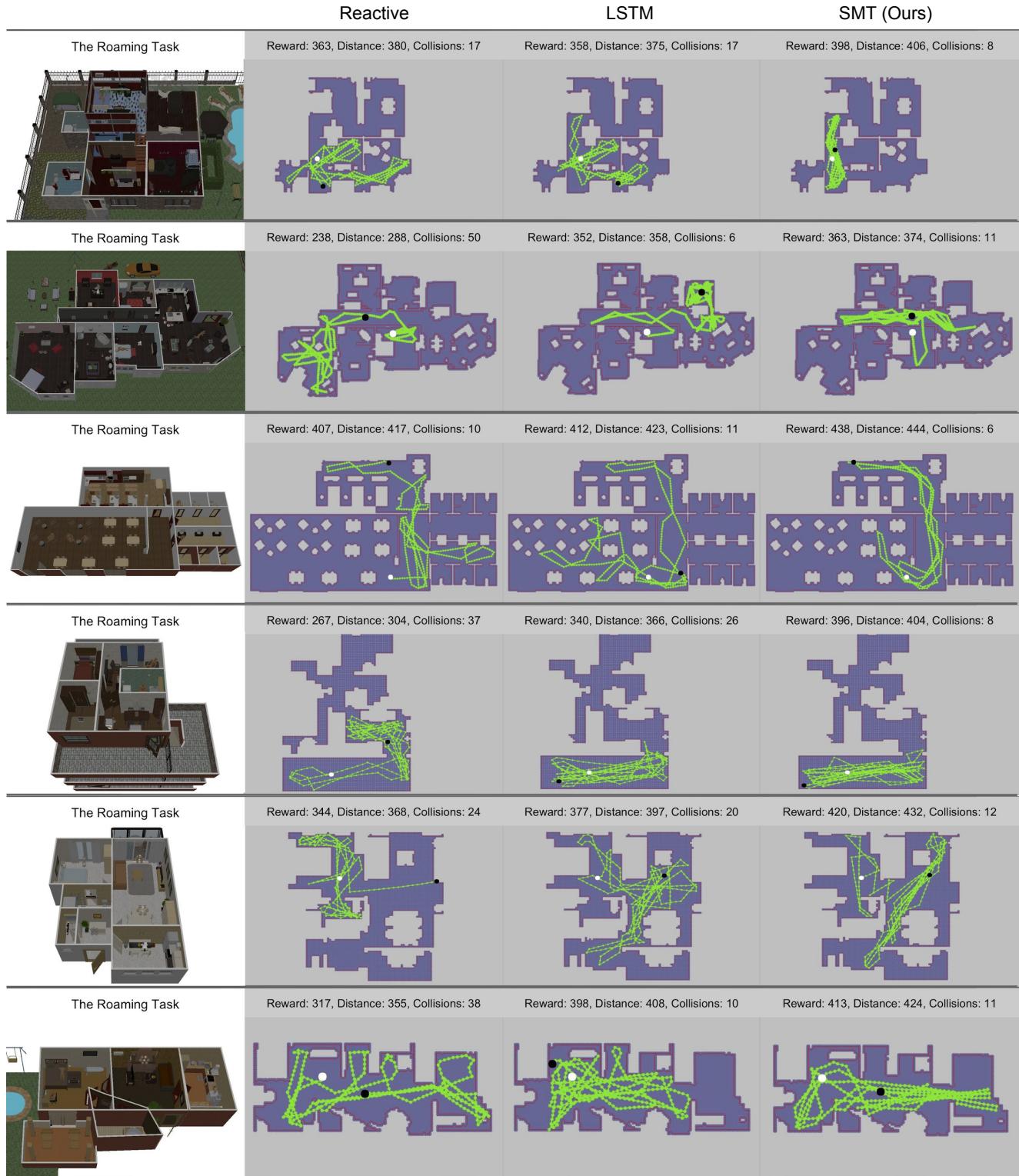


Figure 8. Visualization of the agent behaviors in the Roaming Task.

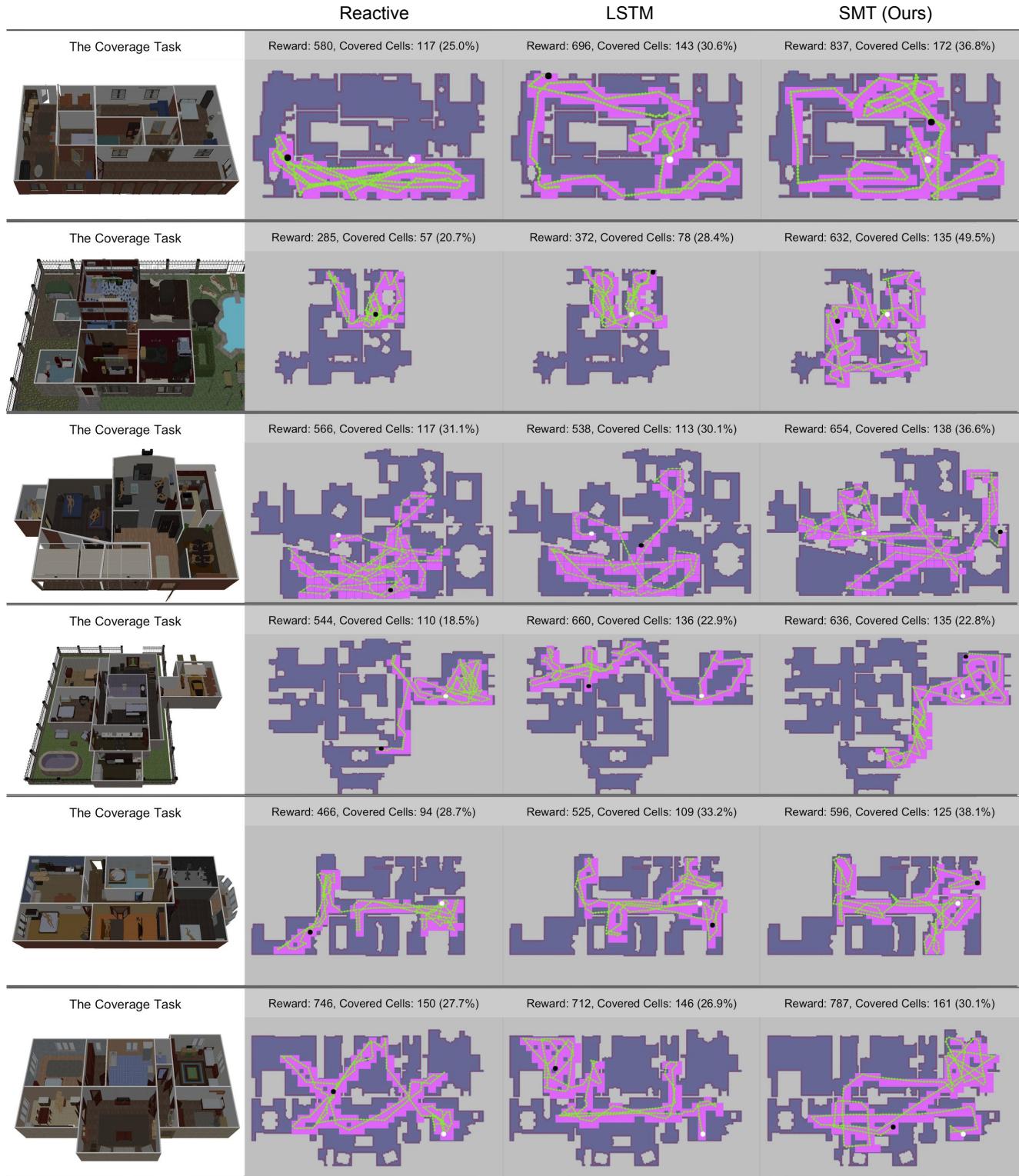


Figure 9. Visualization of the agent behaviors in the Coverage Task.

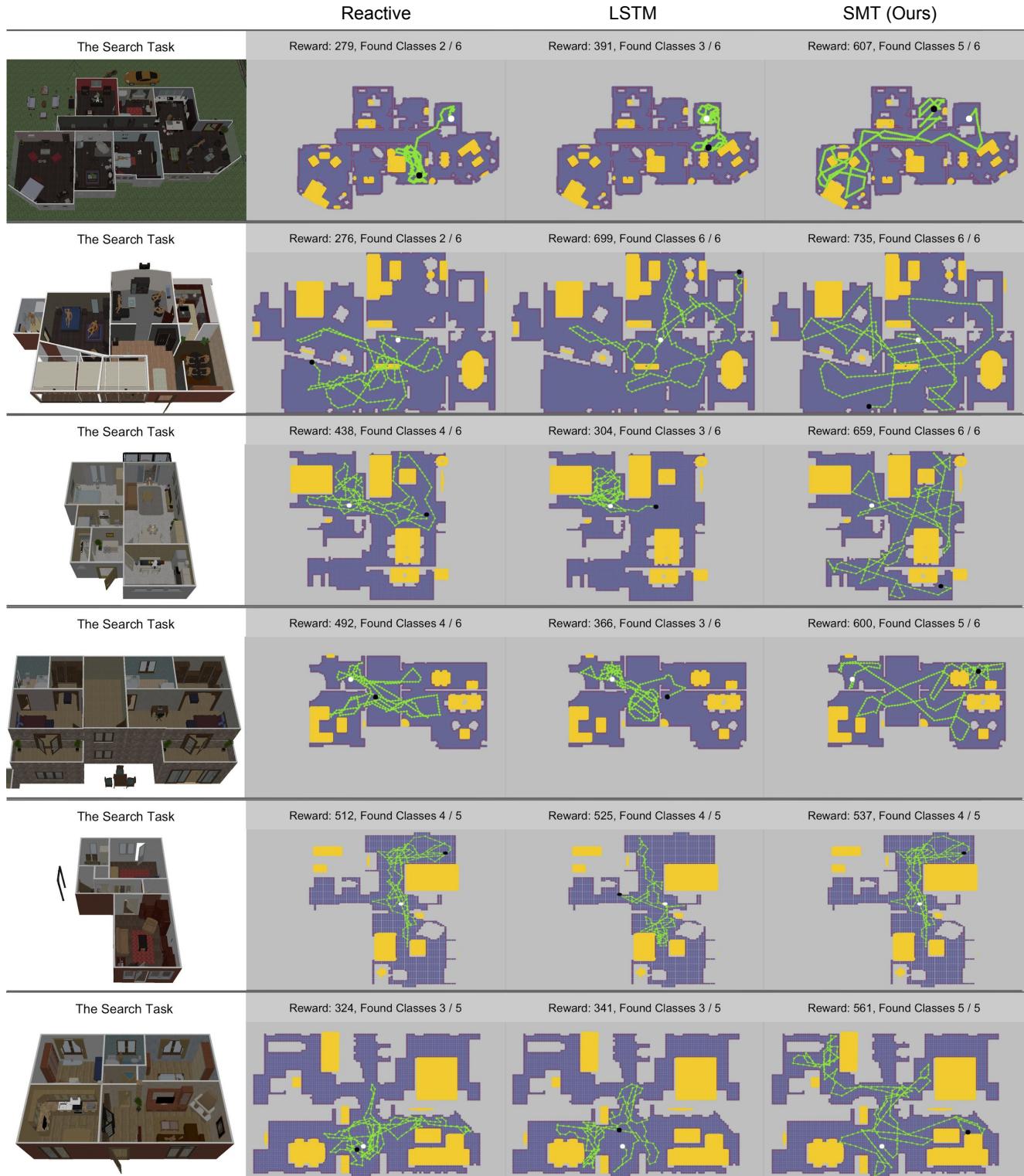


Figure 10. Visualization of the agent behaviors in the Search Task.