# Memory-Augmented Reinforcement Learning for Image-Goal Navigation

Lina Mezghani[1,2]  Sainbayar Sukhbaatar[1]  Thibaut Lavril[1]  Oleksandr Maksymets[1]

Dhruv Batra[1,3]  Piotr Bojanowski[1]  Karteek Alahari[2]

[1]Facebook AI Research  [2]Inria[*]  [3]Georgia Institute of Technology

## Abstract

*In this work, we present a memory-augmented approach for image-goal navigation. Our key hypothesis is that, while episodic reinforcement learning may be a convenient framework for tackling this task, embodied agents, once deployed, do not simply cease to exist after an episode has ended. They persist and so should their memories. Our approach leverages a cross-episode memory to learn to navigate. First, we train a state-embedding network in a self-supervised fashion, and then use it to embed previously-visited states into the agent's memory. Our navigation policy takes advantage of the information stored in the memory via an attention mechanism. We validate our approach through extensive evaluations, and show that our model establishes a new state of the art on the challenging Gibson dataset. We obtain this competitive performance from RGB input alone, without access to additional information such as position or depth.*

## 1. Introduction

Embodied vision is an emerging field that tackles computer vision tasks coupled to control problems. Issues considered in this context go beyond the classical computer-vision setup of learning from pre-defined fixed datasets. This results in several interesting tasks, such as low-level control point-goal navigation [36], object-goal navigation [4] or even problems requiring natural language understanding, e.g., embodied question answering [15].

In this work, we focus on the problem of image-goal navigation [41], wherein an agent has to learn to navigate to a location that is specified by observations taken from there. For example, consider the agent shown in Figure 1. It is spawned at the blue location on the map, where it currently sees a sofa. The agent is asked to find the location in the house where it would see the washing machine shown in
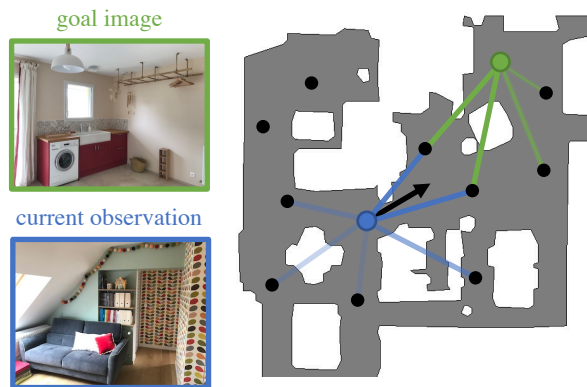


goal image

current observation

Figure 1: We tackle the problem of image-goal navigation. The agent (shown as the blue dot) is given an image from a goal location (green dot) which it must navigate to. To address this task, our agent stores a cross-episode memory of previously visited states (black dots), and uses a navigation policy that puts attention (lines) on this memory.

the goal image. In terms of difficulty, this task lies in between those of point-goal and object-goal navigation. Indeed, it does not require learning the association between visual inputs and manual labels, but it needs a higher-level understanding of scenes for navigating through them. This task can be deemed the embodied equivalent of the classical instance-level recognition problem in images. There are several facets to this task, which make it challenging.

The first challenge of this task is to design methods that are completely end-to-end. Classical methods for embodied agents, based on simultaneous localization and mapping (SLAM) [33] comprise multiple hand-crafted modules, and require a large amount of annotated data. Reinforcement learning stands as the most common framework for tackling navigation problems in an end-to-end manner. However, in the context of photorealistic simulation, existing methods have either shown results in a limited setting with synthetic data [17] or reported poor RL-based performance [11].

The second challenge in accomplishing this task is that

---

it requires a high-level understanding of the surrounding scene. Given the difficulty to interpret raw RGB observations, the agent needs to learn a more informative representation than pixels. This is required to build an internal structure of the surroundings such that the location of the goal can be identified. Learning useful representations is even more important when targeting generalization to unseen environments. Indeed, existing RL-based solutions tend to overfit due to limited size of the available datasets [38].

Finally, a requirement for navigating agents is to exploit past experience. There are two types of information that the agent should remember. First, within an episode, a navigating agent should remember the places it has already visited. This is crucial for efficient exploration [17,25]. Second, it is desirable for the agent to remember the locations it has visited during past episodes within the same scene, and how these locations are connected. This second type of memory is of significant importance for long-term planning [30].

In this work, we tackle these three challenges with a memory-augmented reinforcement learning approach that comprises two stages. In the first stage, the agent learns representations of the environment in a self-supervised fashion, similar to [25]. This allows the agent to have a high-level understanding of the surrounding scene. In the second stage, we use a distributed implementation of proximal policy optimization [36] to learn a policy for image-goal navigation in an end-to-end manner. In order to properly exploit information from past experiences, we endow the agent with an external memory that persists across navigation episodes. We condition the policy on the memory using an attention-based network, akin to the Transformer architecture [35]. We empirically evaluate our approach on the challenging image-goal navigation task on the Gibson [38] dataset, and our approach improves over the state of the art methods by a large margin.

## 2. Related Work

**SLAM-based Methods.** The task of navigation has been studied in the context of simultaneous localization and mapping (SLAM) in robotics [33]. Several SLAM methods comprise multiple hand-crafted modules to address strictly-defined problems in specific environments [14, 28, 34]. These modules have been progressively replaced with learning-based functions: some approaches [10] implement the localization module with a neural network, while others [18] replace the metric map with a latent mapping of the environment. Variants of latent mapping also include a topological map whose nodes contain geometric and semantic information about the environment, as well as a global planner that relies on it [11]. Other works replace SLAM entirely by deep models without explicit planning, and instead rely on a map or memory structure [3, 29, 40]. The major drawback of such methods is that they contain multiple modules that are often trained in a supervised fashion, requiring a large amount of annotated data. Moreover SLAM-based methods rely on the availability of position [29, 40] and depth [3, 11] sensors. In our work, we assume that we only have access to RGB input.

**RL-based Navigation.** Another popular class of methods involves training deep models with reinforcement learning to solve navigation tasks without an explicit world representation. They use end-to-end frameworks with modules that are less hand-crafted than SLAM-based methods, and have shown good performance on synthetic mazes [27] as well as real-world data [7,20,26,36]. Such methods have also been explored on indoor-scenes datasets, similar to our setup on image-goal [41] and object-goal [39] navigation tasks. They use an actor-critic model whose policy is a function of both the target and the current state. While episodic RL is a simple and elegant framework for navigation, it ignores the fact that useful information comes from previous episodes. In our work, we propose to augment the policy used in standard RL frameworks with an external memory and to attend on it with a specific module.

**Combining RL and Planning.** A few recent works have augmented RL-based methods with topological structures, like graphs [5, 12, 25, 30, 37] or memory buffers [6, 17, 23]. They store representations of the visited locations and exploit them at navigation time. The process of building these representations can be done offline [5, 25, 30], and requires human-generated data in some cases [30]. For example, the test phase in [30] contains a warm-up stage where the agent builds a graph memory from human trajectories. Alternatives to this manual annotation do exist, such as building a graph directly with reinforcement learning, using the value function of a goal-conditioned policy as edges weights [16], or a buffer of past observations and attending to them [17]. For now, these methods have only been tested on synthetic datasets, and have not proven to be scalable to high-dimensional visually-realistic setups. In the context of other RL tasks, data augmentation has recently been explored as a regularizer for pixel-based RL for DM Control and Atari [22, 24]. Although data augmentation has been used in robotics applications [21], we show that it helps in planning setups such as navigation in Gibson [38].

**Exploration and Representation Learning.** Closely related to navigation, the task of exploration has also been extensively studied and has led to interesting breakthroughs in representation learning. In particular, learning to explore unseen environments has been seen through the spectrum of computer vision [9, 19], SLAM-based [8], and RL-based [13] approaches. Methods such as [25, 31] leverage a self-supervised representation learning stage to prepare the exploration phase. This pretraining stage allows to learn task-agnostic representations from the scene's locations so as to reduce the dimensionality of the problem.
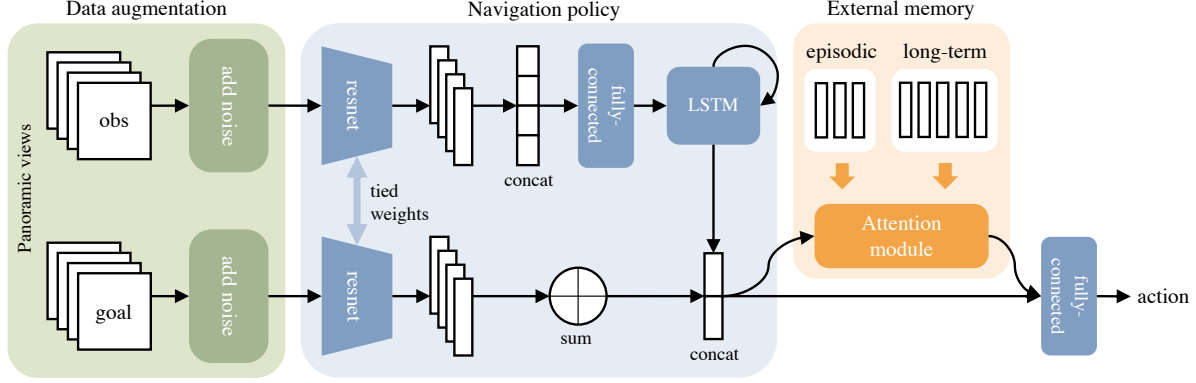
Figure 2: An overview of our agent model that consists of three parts: a data augmentation module (green) for better generalization, a navigation policy (blue) for picking actions, and an external memory (orange) for conditioning on previous observations.

For these works, the generalizability of the learned representations to unseen environments has not been established. Moreover, the use of these representations for navigation tasks was only limited to single-environment setups. Our work extends this line of study by first showing that a self-supervised pretraining phase allows to learn useful information that generalizes to unseen environments, as well as proposing a novel attention-based navigation policy that takes advantage of this information.

## 3. Problem Formulation

At the beginning of a navigation episode, an agent is given a target observation $x^*$, composed of an RGB image from the target location. At each timestep $t$, the agent performs an action $a_t$ and receives the next observation $x_{t+1}$ as well as a reward $r_t$ from the environment. The objective is to learn a navigation policy function $\pi(a_t|x_t, x^*)$ that brings the agent closer to the target location. We complete the definition of our setup with the following details.

**Action Space.** It comprises 4 actions: MOVE_FORWARD, TURN_LEFT, TURN_RIGHT and STOP. Please refer to Section 5.1 for numerical details.

**Success Criterion.** An episode is considered successful if the agent performs the stop action within a range of $l$ from the target location. In cases where the agent performs the stop action outside of this range, or if the maximum number of steps is exceeded before the agent performs the stop action, the episode is considered a failure.

**Observation Space.** The observation of the agent $x_t$ as well as the goal observation $x^*$ are the RGB images of the first-person view at those locations. Each RGB image is a panoramic sensor of size $v \times 3 \times 128 \times 128$. We compute this panoramic input by gathering observations from $v$ successive rotations of angle $(360/v)°$ from our agent's location.

Note that we do not have access to neither the agent's position nor any depth sensor information.

**Reward.** We follow the classic setup for image-goal navigation where the reward is split into three components: (i) *sparse success reward*: that rewards the agent for performing the stop action within the success range around the target location, (ii) *dense shaping reward*: that is equal to the decrease in distance to the goal, (iii) *dense slack reward*: that penalizes the agent for being alive at each step, and encourages shorter trajectories.

## 4. Approach

We now describe our approach for solving the image-goal navigation problem. As shown in Figure 2, our agent model has three parts: a data augmentation part for improving generalization, a navigation policy that learns to pick appropriate actions, and an external memory for leveraging past experiences.

### 4.1. Data Augmentation

To improve the generalization capacity of our agent to unseen environments, we apply random transformations on the observations of the simulator by using classic data augmentation techniques. We use two kinds of data transformations: (i) **random cropping** that increases the input image size and takes a random crop of the original size in it, and (ii) **color jitter** that randomly changes the brightness, contrast, saturation and hue levels of the image. An illustration for these transformations is shown in Figure 3. At navigation time, the agent receives the current and the goal observations from the simulator at each timestep. We apply both transformations sequentially to each of the $v$ views of the current and goal observations independently, producing $x_t$ and $x^*$ respectively. This process allows for more visual

| orig | color augment | orig | crop augment |

Figure 3: Illustration of data augmentation that we use to train our model. We consider both color jittering (left) and random crops (right). For a panoramic observation with $v$ views, the parameters of the augmentation are sampled independently.

diversity in the training data.

## 4.2. Navigation Policy

Once the current and goal observations pass through the data augmentation phase, we use them in the navigation policy module, which computes a probability distribution over all possible actions: $\pi(a_t|x_t, x^*)$.

First, the policy encodes each observation separately, as shown in Figure 2. We encode the current observation by feeding each of the $v$ views separately to the same convolutional neural network. The $v$ vectors resulting from this operation are concatenated and passed into a fully-connected network. This dimension-reduced output is then fed into a 2-layer LSTM along with a representation of previous actions, and its resulting vector $w_t^{\text{obs}}$ represents the embedding of the current observation at step $t$.

To encode the goal observation $x^*$, we process it through the same convolutional neural network as the current observation. However, the outputs corresponding to the different views are added together instead of being concatenated, so as to make the representation of the goal rotation-invariant. We denote by $w_t^{\text{goal}}$ the resulting feature vector at step $t$.

Next, we make a joint representation by concatenating the current and goal feature vectors, before passing it through a fully-connected network to output an action:

$$w_t^{\text{joint}} = \mathtt{cat}(w_t^{\text{obs}}, w_t^{\text{goal}}), \qquad (1)$$

$$\pi(a_t|x_t, x^*) = \mathtt{FC}(w_t^{\text{joint}}). \qquad (2)$$

## 4.3. External Memory

We add an external memory mechanism to the navigation policy to leverage past experiences. To this end, we first train a state-embedding network in a self-supervised fashion, as described in Sec. 4.3.1. This network, trained to detect nearby locations, allows us to build an external memory containing representations of past observations. To leverage

this memory, we add an attention module to the navigation policy so it can condition on this memory.

### 4.3.1 Training a State-Embedding Network

Before learning the navigation policy, we train a state-embedding network to learn representations of the environment's locations. The motivation for introducing this network is to encourage nearby locations in the environment to have similar representations, while pushing distant locations to have different ones. However, since we cannot access to the agent's position, the notion of distance between locations in the environment cannot be computed directly. As in [31], we will use the number of steps taken by an agent with a random policy to approximate this distance.

We let an agent with random policy explore the environment for T steps and denote by $(x_1, ..., x_T)$ the corresponding sequence of observations.[1] We then define a reachability label $y_{ij}$ for each pair of observations $(x_i, x_j)$ that depends on their distance in the sequence. More precisely,

$$y_{ij} = \begin{cases} 1 & \text{if} \quad |i - j| \leq k \\ 0 & \text{otherwise} \end{cases} \quad \text{for } 1 \leq i, j \leq T \quad (3)$$

where $k$ is a hyperparameter.

We train a siamese neural network $R$, the Reachability Network [31], to predict the reachability label $y_{ij}$ from a pair of observations $(x_i, x_j)$. $R$ is defined by a convolutional network $g$ to embed the observations and a fully-connected network $f$ to compare the embeddings, i.e.,

$$R(x_i, x_j) = f(g(x_i), g(x_j)). \qquad (4)$$

Note that this stage of training is unsupervised since no reward signal is needed. Also, we apply the same data augmentation techniques to observations during this Reachability Network training.

### 4.3.2 Episodic and Long-term Memories

Once we have the Reachibility Network that can differentiate observations from nearby and distant locations, the agent can collect a compact memory of its previously visited states. It will then have two distinct memory types: an episodic memory that contains elements from the current episode only, and a long-term memory that gathers information across multiple episodes. A visualisation of the episodic and long-term memories is shown in the appendix.

**Episodic Memory.** We follow a process similar to Savinov *et al.* [31] for building episodic memory. At timestep $t$, the agent has a memory buffer $M_{t-1}$ with embeddings from

---

[1] We ensure that the length of the computed sequence is $T$ by removing the STOP action from the action space.

observations seen at previous timesteps. Since storing every observation seen by the agent would be inefficient, we store only observations that are considered novel, i.e., distant from the current memory vectors. In other words, at each timestep, we use the network $R$ to compute a reachability score between the current observation $x_t$ and the memory buffer $M_{t-1}$. This score corresponds to the maximum value obtained when comparing the current observation to every vector in the memory, i.e.,

$$r(x_t, M_{t-1}) = \max\{\, f(g(x_t), m) \mid m \in M_{t-1}\}. \quad (5)$$

We then add the embedding of current observation to the memory if the reachability score is lower than a threshold, i.e., distant enough from the memory vectors. In other words,

$$M_t = \begin{cases} M_{t-1} \cup g(x_t) & \text{if } r(x_t, M_{t-1}) < \tau, \\ M_{t-1} & \text{otherwise,} \end{cases} \quad (6)$$

where $\tau$ is the *reachability threshold* hyperparameter. The episodic memory is reset after each episode.

**Long-Term Memory.** In contrast to the episodic memory that contains only information about the current episode, long-term memory includes information from previous episodes. Given a scene $s$ in the dataset, we denote by $L_t^s$ the state of the long-term memory at timestep $t$ in scene $s$. This memory is conserved over consecutive episodes, and is reset after $p$ episodes in the same scene.

Within an episode, the update rule is similar to that of episodic memory. The current observation $x_t$ is added to the long-term memory $L_{t-1}$ if their reachability score lower than a threshold, i.e.,

$$L_t^s = \begin{cases} L_{t-1}^s \cup g(x_t) & \text{if } r(x_t, L_{t-1}^s) < \tau, \\ L_{t-1}^s & \text{otherwise.} \end{cases} \quad (7)$$

### 4.3.3 Attention Module

The navigation policy can use both the episodic and long-term memories to move towards the target direction. Rather than using an explicit planner on the memories, we take advantage of the information stored in them implicitly, using an *attention module*.

The attention module has a multi-layer architecture similar to Transformers [35]. Each layer consists of a multi-head attention sublayer (`Attn`), followed by a feedforward sublayer (`FF`). See [35] for more details about these sublayers. Unlike Transformers, however, our attention module attends over a fixed set of vectors. In particular, the first $N$ layers attend to the episodic memory, while the last $N$ layers attend to the long-term memory:

$$\begin{aligned} z_t^l &= \text{FF}\left(\text{Attn}(z_t^{l-1}, M_t)\right) \quad \text{for } l \le N, \\ z_t^l &= \text{FF}\left(\text{Attn}(z_t^{l-1}, L_t^s)\right) \quad \text{for } N < l \le 2N. \end{aligned} \quad (8)$$

Here, $z_t^l$ is the output from the $l$-th layer, but the initial input $z_t^0$ is obtained by a linear transformation of the joint representation computed in Eq. (1): $z_t^0 = W w_t^{\text{joint}}$.

The output from the attention module is then concatenated with the joint representation in Eq. (2), so the final action is now computed by

$$\pi(a_t | x_t, x^*) = \text{FC}(\text{cat}(w_t^{\text{joint}}, z_t^{2N})). \quad (9)$$

## 5. Experimental Results

We seek to answer the following questions through our evaluation. (i) Does the external memory module improve the performance of the navigation agent? (ii) How does our method compare to the state of the art on the task? (iii) How do data augmentation, episodic and long-term memories impact the performance and generalization of the model?

### 5.1. Implementation Details

**Task Setup.** We conducted all of our experiments on the Habitat [32] simulator with the Gibson [38] dataset, which contains a set of visually-realistic indoor scenes. We use the standard 72/14 train/test scene split for this dataset. As stated earlier, *we do not use the agent's pose or depth sensor information*. The forward step range and turn angle are set to their standard values $(0.25\text{m}, 10°)$ for navigation episodes and $(1\text{m}, 30°)$ when training the reachability network. The maximum number of steps in an episode is 1k, and the success distance $l$ is 1m. In addition to the success rate, we also use success weighted by inverse path Length (SPL) [2] as an evaluation metric. SPL takes into account the length of the path that the agent has taken to the goal.

**Training the Reachability Network.** We generate one trajectory per train scene from an agent with a random policy. We allow 5k steps for each trajectory and remove the stop action from the action space. This results in a total of 360k steps from 72 scenes to train the reachability network. From each trajectory, we sample 1k positive pairs (within 10 timesteps) and 1k negative pairs, yielding a dataset of 144k image pairs. We implement the reachability network as a siamese network with ResNet18 for the embedding function $g$. Each of the $v$ views from the RGB observation is passed through the ResNet separately. We sum the resulting outputs to form the embedding vector of a panoramic observation. The comparison function $f$ is composed of two hidden layers of dimension 512. We train this network using SGD for 30 epochs with a batch size of 256, a learning rate of 0.01, a momentum of 0.9, a weight decay of $10^{-7}$, and no dropout. More details about the architecture of the network, the train and validation curves, as well as qualitative results, are shown in the appendix.

**Training Data for the Navigation Policy.** We generated 9k navigation episodes in each training scene. We generate

the training episodes following the protocol of Chaplot *et al.* [11] [2]. We split our navigation episodes into three levels of difficulty, based on the distance between the start and the goal locations: *easy* (1.5 - 3m), *medium* (3 - 5m), and *hard* (5 - 10m). For each scene, we sample 3k start-goal location pairs per level of difficulty, resulting in 648k train episodes. Similarly, we sample 100 episodes per test scene and per level of difficulty, resulting in 4.2k test episodes.

**Navigation Policy Implementation.** At the beginning of each episode, the simulator generates the observation from the goal location as a $v \times 3 \times 128 \times 128$ panoramic RGB image and gives it to the agent as target observation. We use ResNet18 with shared weights for encoding the current and target observations in the policy. The size of the embedding space is 512. We concatenate the encoder's outputs for the $v$ views of the observation and feed them into an LSTM with two recurrent layers. Our attention module consists of 4 stacked layers of a 4-headed attention network. We set the buffer's capacity to 20 for the episodic memory and 100 for the long-term memory. We train the policy using DDPPO for 50k updates, with 2 PPO epochs, a forward of 64 steps, an entropy coefficient of 0.01, and a clipping of 0.2. We use the Adam optimizer with a learning rate of $2.5e - 4$.

**Data Augmentation.** For the training stages of both the reachability network and the navigation policy, we used random cropping with a minimum scale of 0.8 and color jittering with value 0.2 for brightness, contrast, saturation, and hue levels. These transformations are applied at two different levels when training the navigation policy: (i) when the agent samples the action $a_t$ from the policy, and (ii) during the forward-backward in PPO. Note that, for these two steps, the transformation applied to the images is independent and results in different input images.

## 5.2. Ablation Study and Analysis

As a first experiment, we perform an ablation study to empirically validate the design choices of our model. We evaluate the performance of the simple target-driven RL baseline, as well as the improvements brought by the data augmentation module, the episodic and long-term memories. To this end, we train four variants of our model: (i) **RL baseline**, the vanilla target-driven RL baseline to which we consecutively add (ii) **Data Aug.**, the data augmentation module, (iii) **Ep. Memory**, the episodic memory module, and (iv) **LT Memory**, the long-term memory module. In addition to the dataset containing *easy*, *medium*, and *hard* episodes, we generate an additional set of train and validation episodes "*extra*", for which the distance between start and goal locations is 10 - 15m. We train all models on this dataset for 500M steps for three random seeds and report
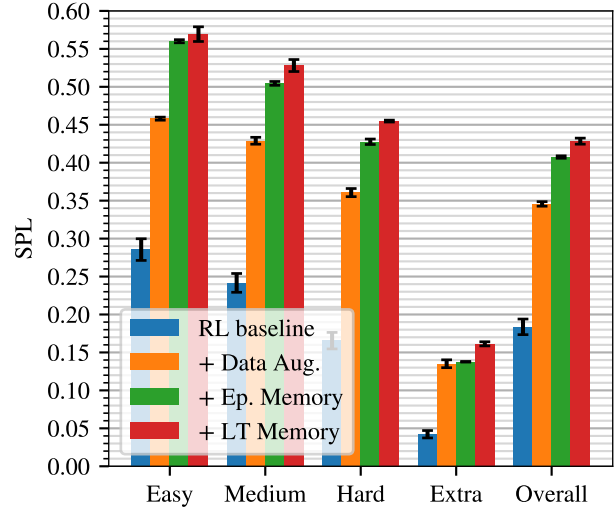
---

Figure 4: Ablation study of ou method. We present the SPL for four variants of our model. All models are trained for 500M steps on a combination of *easy*, *medium*, *hard* and *extra* episodes.

the average SPL for each level of difficulty. We present the results of this experiment in Figure 4.

First, we see that using data augmentation when training a RL-based navigation policy in this context improves the SPL significantly: the gap with the vanilla baseline is $+16\%$ *overall*. Second, we observe that the episodic and long-term memory-based policies improve over the very competitive data-augmented baseline ($+6\%$ and $+8\%$ *overall* respectively). The improvement of the long-term memory-based model over the episodic one is significant ($+2\%$ *overall*), and is slightly more important on *hard* episodes ($+3\%$) than *easy* ones ($+1\%$). This highlights the importance of having a memory that persists across episodes in the same scene, especially for goals that are far away. Finally, we see that on the *extra* split, the performance is much lower than on other splits, and that the model with long-term memory improves significantly over the one with episodic memory ($+2\%$).

## 5.3. Comparison with the state of the art

We consider several methods in this comparison: an adaptation of SPTM [30] to Habitat, as well as four approaches taken from [11].

**ResNet + GRU + IL** [11]: A simple baseline consisting of ResNet18 image encoder and a GRU-based policy trained with imitation learning.

**Target-Driven RL** [41]: A vanilla baseline trained with reinforcement learning, similar to our ablated variant.

**Active Neural SLAM** [8]: An exploration model based on metric maps, adapted to navigation.

**Neural Topological SLAM (NTS)** [11]: The previous

| Model | Past exp. | Easy | | Medium | | Hard | | Overall | |
|---|---|---|---|---|---|---|---|---|---|
| | | Succ | SPL | Succ | SPL | Succ | SPL | Succ | SPL |
| ResNet + GRU + IL [11] | No | 0.57 | 0.23 | 0.14 | 0.06 | 0.04 | 0.02 | 0.25 | 0.10 |
| Target-Driven RL [41] | | 0.56 | 0.22 | 0.17 | 0.06 | 0.06 | 0.02 | 0.26 | 0.10 |
| Active Neural SLAM [8] | | 0.63 | 0.45 | 0.31 | 0.18 | 0.12 | 0.07 | 0.35 | 0.23 |
| Neural Topological SLAM [11] | | **0.80** | 0.60 | 0.47 | 0.31 | 0.37 | 0.22 | 0.55 | 0.38 |
| SPTM [30] | Yes | 0.64 | 0.35 | 0.52 | 0.27 | 0.36 | 0.19 | 0.51 | 0.27 |
| **Ours** | | 0.78 | **0.64** | **0.71** | **0.58** | **0.62** | **0.48** | **0.70** | **0.57** |

Table 1: Comparison of our proposed model with several baselines and state-of-the-art approaches. We report success rate and SPL, over three levels of difficulty. We distinguish two different setups: with and without access to past experience. Our method establishes a new state of the art, e.g., doubling the SPL on *hard* episodes.

state-of-the-art method for navigation in unseen environments, based on neural SLAM and endowed with a topological graph for long-term planning.

**SPTM** [30]: A navigation model based on explicit construction of a graph over past experience. The method requires an exploration episode given *a priori*. To run this baseline, we adapt the available code [1] to Habitat. We train the reachability and locomotion networks on trajectories from the 72 training scenes. For evaluation we use a random exploration trajectory of 10k steps, and build a graph with 100k top-scoring edges. As this method does not implement a STOP action, we automatically end the episode when the agent gets within 1m of the goal.

The first four methods are designed for episodic navigation, which means that the validation episodes are performed on completely unseen scenes, without pre-exploration. Conversely, **SPTM** [30] is intended for setups where a pre-exploration phase is allowed on the test scenes before starting the actual navigation phase. Our model does not explicitly perform an exploration phase on the navigation episodes, but is allowed to have a memory that persists across episodes in the same scene. In that respect, our setup is related to that of **SPTM** [30] as it makes the use of past experience possible. When comparing to **NTS** [11], our model is advantaged by having access to cross-episode information. For completeness, we provide an ablation of our method in the appendix.

We trained our model on the *easy*, *medium*, *hard* combined dataset for 500M steps for three random seeds, and evaluated it on the corresponding test set. As shown in Table 1, the performance obtained with our memory-augmented model is superior to that of previous work by a significant margin: $+19\%$ SPL on average compared to **NTS** [11] and $+30\%$ SPL vs. **SPTM** [30]. We obtain this strong performance while, unlike **NTS** [11], not using position information. We note that the success rate on *easy* episodes is lower for our method than **NTS** [11] ($-2\%$).

| Number of views ($v$) | 1 | 3 | 4 | 6 |
|---|---|---|---|---|
| SPL | 0.08 | 0.31 | 0.36 | 0.36 |
| Frames per sec. | 1890 | 2000 | 2080 | 2340 |

Table 2: Analysis of the SPL obtained with the **RL baseline** for various panoramic view configurations. We report the average SPL as well as the number of frames that we can process per second for the given configuration. An agent with a single view fails to learn a successful policy. Given the performance / speed tradeoff that we observe, we use 4 views in all of our experiments.

This is potentially due to the lack of position features or the use of discretized panoramic observations, both of which are particularly useful for nearby goals.

### 5.4. Analysis and qualitative visualisations

**Panoramic Observations.** As described in Sec. 3, we generate panoramic observations by equally spaced planar observations around the agent. In this experiment, we compare the performance of the vanilla **RL baseline** model trained with 1, 3, 4 and 6 views around the agent. We let the model train for 500M steps for three random seeds and report the average SPL obtained by this agent in Table 2.

The first thing we observe from Table 2, is that an agent trained with a single view, i.e., without panoramic observations, completely fails to learn a successful policy, obtaining only 0.08 SPL. This result is quite intuitive, as the relative localization with respect to the goal is made much easier by multiple views, removing the need for additional search if the goal is behind the agent. Better performance is obtained with either four or six views, with an SPL of 0.36. Second, we see that there is a tradeoff between the performance we obtain and the additional runtime required with more views. Because of that, in our experiments, we run all the variants
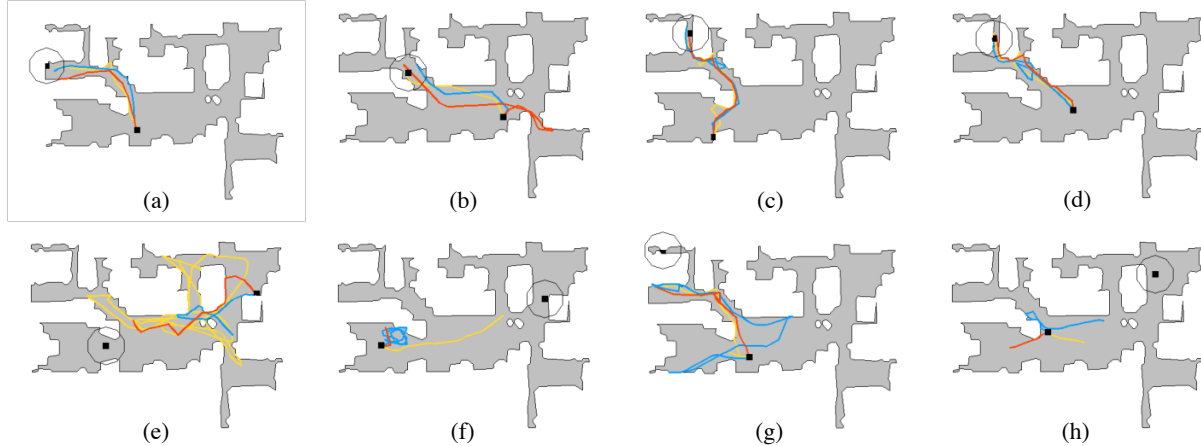
Figure 5: Qualitative Results. The top (resp. bottom) row shows trajectories from test episodes with highest (resp. lowest) SPL on the Eastville scene. Start and goal locations are shown in black, with the goal being circled by a line showing the success area. The results are shown for 3 seeds on the *hard* validation split, for our model with memory trained on the standard dataset.
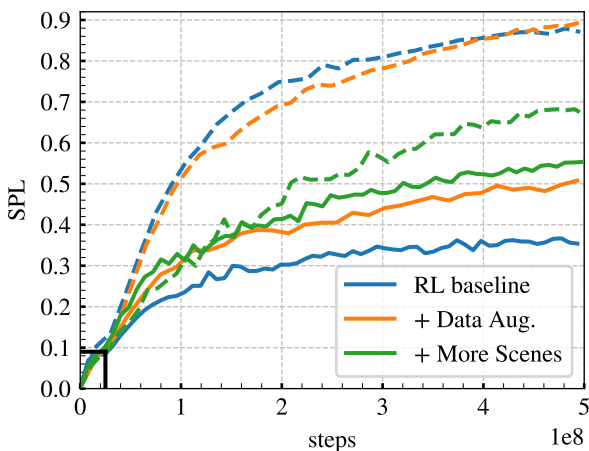


Figure 6: Performance measured in SPL as a function of training steps taken in the environment for the **RL baseline**, with data augmentation (+ Data Aug.) and by adding more training scenes (+ More Scenes). We report both the training SPL (dashed line) and the test SPL (solid line) on the same figure. The generalization gap is large and can be reduced by using classical computer vision data augmentation. It can be reduced significantly further by training the model on a larger set of scenes. The black mark at 25M steps correspond to the **Target-Driven RL** baseline as reported in [11].

of the models with four views.

**Data Augmentation and Overfitting.** Here, we evaluate the capacity of our navigation agent to generalize to unseen environments. We investigate how data augmentation

and adding more training scenes allows us to bridge the train / test gap observed on this task. To this end, we generate an extended training set by considering 150 additional scans, which are usually rated as being of poor quality. We perform this experiment for the vanilla **RL Baseline**, as well as the data-augmented version of it trained on the standard dataset (**Data Aug.**), and the extended one (**More Scenes**).

As seem from the train and test SPL for the three methods in Fig. 6, the generalization gap is huge (almost $65\%$ for the **RL baseline**). The use of data augmentation allows to remedy this problem reducing this gap to about $40\%$. Training the model on more scenes reduces this discrepancy even further to only $15\%$. We also observe that data augmentation not only improves the test performance but also helps faster convergence ($+2\%$ on the train SPL), as opposed to what is usually observed in supervised learning.

Notice that in the figure, we indicate the 25M step mark – this is the number of environment steps used to train the **Target-Driven RL** baseline reported in [11]. We see that the performance of a reinforcement-based model with such a limited number steps is quite poor, and that convergence is far from being achieved at this point. Though sample efficiency is an important challenge in robotics applications, the **RL baseline** should be trained sufficiently to leverage its full potential. Indeed, after 500M steps of training, it reaches 0.37 SPL *overall*, which is almost on par ($-1\%$) with the performance of the method introduced in [11].

**Qualitative Results.** Figure 5 shows a few success and failure cases from episodes of the test dataset. We see that our agent successfully learns to navigate to challenging locations, which are distant from the start location (5-a) and/or located at extremities of the scenes (5-c, 5-d). More-

over, our agent shows interesting skills, like bypassing ob-
stacles (5-c) or looking around in a room (5-b). From the
failure cases, we see that our agent has some undesired be-
haviour. For example, it can get stuck in a loop (5-f), stop
too early (5-h), or fail to reach some extremely challenging
goals (5-g).

## 6. Conclusion

In this paper, we have presented a memory-endowed
agent that we train end-to-end with reinforcement learning.
This memory is accessed in the navigation policy using a
transformer-inspired neural network, with attention mod-
ules over past experiences. We evaluated our agent on the
challenging task of image-goal navigation, and have shown
that it surpasses previous work by a fair margin. In order
to showcase the performance of the agent in a very com-
plex setup, we extended the dataset proposed by [11] by
adding episodes with goals 10-15m away. In future work,
we plan to improve the training of the Reachability Network
and make the policy better exploit the memory.

# References

[1] https://github.com/nsavinov/SPTM. 7

[2] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 5

[3] Gil Avraham, Yan Zuo, Thanuja Dharmasiri, and Tom Drummond. Empnet: Neural localisation and mapping using embedded memory points. In *Int. Conf. Comput. Vis.*, 2019. 2

[4] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv preprint arXiv:2006.13171*, 2020. 1

[5] Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf. Learning to plan with uncertain topological maps. In *Proc. ECCV*, 2020. 2

[6] Edward Beeching, Christian Wolf, Jilles Dibangoye, and Olivier Simonin. EgoMap: Projective mapping and structured egocentric memory for deep RL. *arXiv preprint arXiv:2002.02286*, 2020. 2

[7] Marvin Chancán and Michael Milford. MVP: Unified motion and visual self-supervised learning for large-scale robotic navigation. *arXiv preprint arXiv:2003.00667*, 2020. 2

[8] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural SLAM. *arXiv preprint arXiv:2004.05155*, 2020. 2, 6, 7

[9] Devendra Singh Chaplot, Helen Jiang, Saurabh Gupta, and Abhinav Gupta. Semantic curiosity for active visual learning. *arXiv preprint arXiv:2006.09367*, 2020. 2

[10] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. *arXiv preprint arXiv:1801.08214*, 2018. 2

[11] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological SLAM for visual navigation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 1, 2, 6, 7, 8, 9, 13

[12] Kevin Chen, Juan Pablo de Vicente, Gabriel Sepulveda, Fei Xia, Alvaro Soto, Marynel Vázquez, and Silvio Savarese. A behavioral approach to visual navigation with graph localization networks. *arXiv preprint arXiv:1903.00445*, 2019. 2

[13] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. *arXiv preprint arXiv:1903.01959*, 2019. 2

[14] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Transactions on robotics and automation*, 17(2):125–137, 2001. 2

[15] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2018. 1

[16] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, 2019. 2

[17] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 1, 2

[18] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 2

[19] Dinesh Jayaraman and Kristen Grauman. Learning to look around: Intelligently exploring unseen environments for unknown tasks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 2

[20] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters (RA-L)*, 2020. 2

[21] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018. 2

[22] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020. 2

[23] Ashish Kumar, Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Visual memory for robust path following. In *Advances in Neural Information Processing Systems*, 2018. 2

[24] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020. 2

[25] Lina Mezghani, Sainbayar Sukhbaatar, Arthur Szlam, Armand Joulin, and Piotr Bojanowski. Learning to visually navigate in photorealistic environments without any supervision. *arXiv preprint arXiv:2004.04954*, 2020. 2

[26] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*, 2018. 2

[27] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. 2

[28] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 2

[29] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017. 2

[30] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018. 2, 6, 7, 13, 15

[31] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018. 2, 4, 12

[32] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9339–9347, 2019. 5, 15

[33] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002. 1, 2

[34] Nicola Tomatis, Illah Nourbakhsh, and Roland Siegwart. Combining topological and metric: A natural integration for simultaneous localization and map building. In *Proc. European Workshop on Advanced Mobile Robots (Eurobot)*. ETH-Zürich, 2001. 2

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, 2017. 2, 5

[36] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019. 1, 2

[37] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuandong Tian. Bayesian relational memory for semantic visual navigation. In *Int. Conf. Comput. Vis.*, 2019. 2

[38] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 2, 5

[39] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018. 2

[40] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural SLAM: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017. 2

[41] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Proc. ICRA*, 2017. 1, 2, 6, 7

## A. Analysis of the Reachability Network

We begin with additional details about training the Reachability Network and then present a few qualitative visualisations.

### A.1. Architecure & Training Details

We adapted the architecture of the Reachability Network presented in [31] to work with panoramic views in realistic environments. An illustration of this model is shown in Figure 7. The main differences with [31] are the use of a data augmentation module, that applies transformations to the RGB input of the network, and an additional layer to handle panoramic observations. This layer aggregates the ResNet output for each of the $v$ views into one feature vector. We experimented with two architectures for this layer: (i) **cat + FC**: where we concatenate the ResNet output for each of the $v$ views and feed this large vector into a one-layer fully-connected module, and (ii) **sum**: where we simply sum these $v$ vectors. Contrary to (i), (ii) has the interesting property of yielding embeddings that are rotation invariant—the resulting feature vector from a location of the environment will be the same for every direction of the agent.
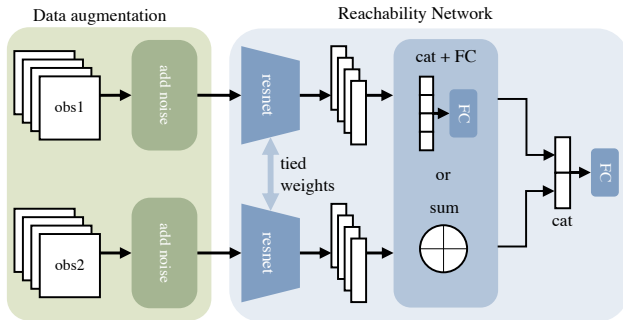


Figure 7: Architecture of the Reachability Network. We adapted the architecure from [31] by using a data augmentation module and a layer that handles panoramic observations. The output of the last fully-connected module is the similarity score between the two observations.

We compare the train and validation performance of these two design variants, and the influence of using data augmentation for training the Reachability Network. The results are shown in Figure 8. First, we observe that using data augmentation reduces overfitting in both the setups, and yields a better validation accuracy in these two cases. Second, we see that summing panoramic features allows to achieve better train and validation performance than concatenating them. One explanation for this is that the rotation invariance explained above is facilitating the learning. For all of our navigation policy learning experiments, we chose

the setup **sum + augmentation** for training the Reachability Network that encodes the memory vectors.
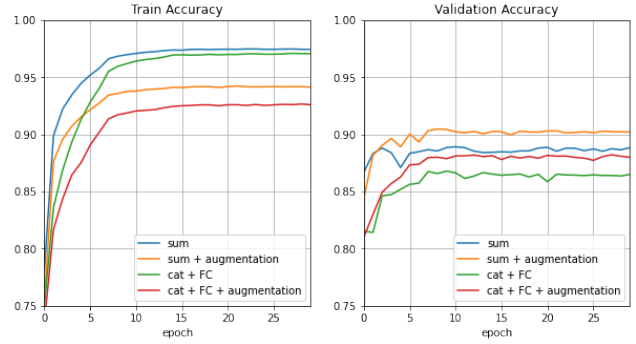


Figure 8: Training and validation curves for the Reachability Network. We tested four setups, that compare the use of data augmentation and the choice of the aggregation layer for panoramic features.

### A.2. Qualitative Visualisations

We visualise the quality of the Reachability Network with the following experiment. First, we put the agent at a random location in the environment and sample an observation $x$ from there. Then, we randomly sample $N$ observations in the environment and for each of these observations, we compute their similarity score with observation $x$, using the Reachability Network. We present these results on a heat-map, where the colour at a location represents the corresponding similarity score. Some examples are shown in Figure 9. We see that the high similarity scores are at locations that are around the comparison observation, which implies that the Reachability Network performs well at learning representations that are similar for nearby locations, and dissimilar for representations that are far away. Since these experiments are shown on a validation environment, we note that the Reachability Network generalizes well to unseen environments.

We also visualise the state of the episodic and long-term memories for consecutive validation episodes in Figure 10. From this, we observe how these memories are filled through consecutive validation episodes. After 100 navigation episodes (10-d), the long-term memory is well filled and covers most of the environment. This allows the agent to reach challenging goals.

## B. Full ablation study

In this section, we provide the full ablation of our methods on the Image-Goal navigation task.

**Ablation study.** First, we present an ablation that complements results from Sec. 5.3. We compare the four ablated variants of our method described in Sec. 5.2 with the state
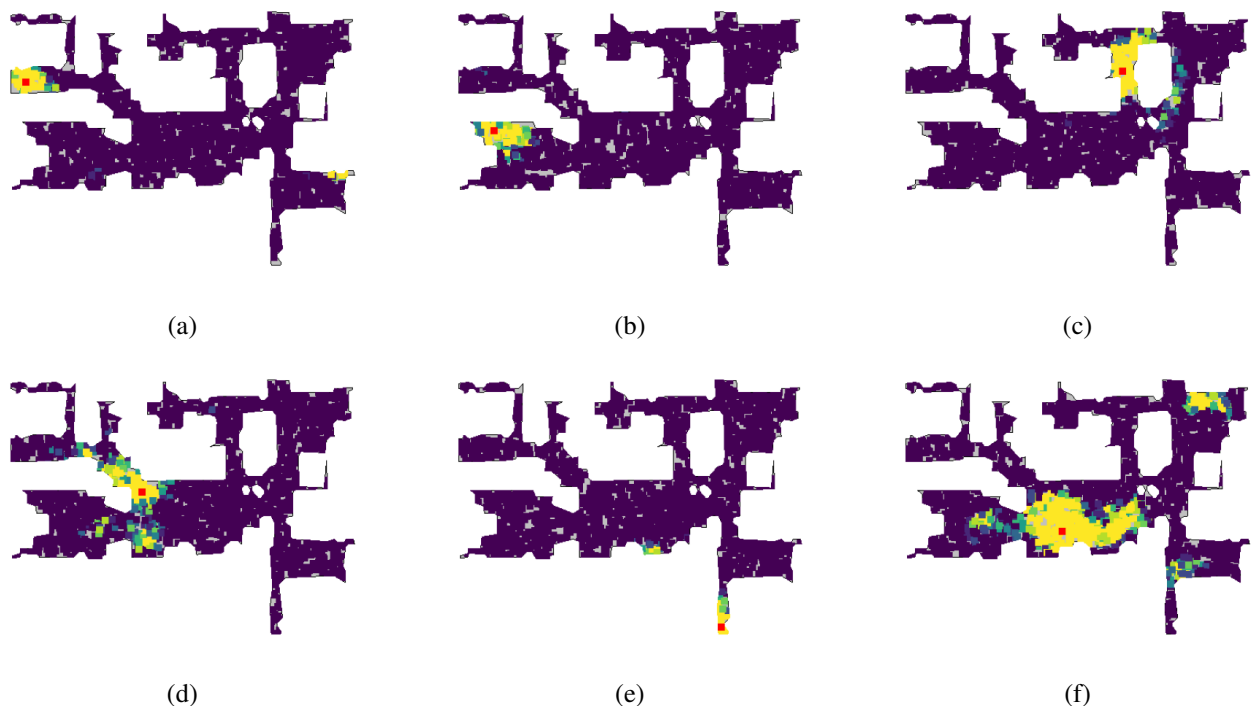
(a)　　　　　　(b)　　　　　　(c)

(d)　　　　　　(e)　　　　　　(f)

Figure 9: Heat-maps of the similarity score between the observation (the red point) and the observations at $N = 2000$ points sampled randomly in the environment. The colour at a location corresponds to the similarity score at that location: low values, close to 0, are in dark violet and high values, close to 1, in yellow. These visualisations were performed on Eastville: an environment of the validation set.

| Model | Past exp. | Easy | Medium | Hard | Overall |
|---|---|---|---|---|---|
| **NTS** [11] | ✗ | 0.60 | 0.31 | 0.22 | 0.38 |
| RL baseline | ✗ | 0.44 | 0.37 | 0.26 | 0.36 |
| + Data Aug. | ✗ | 0.59 | 0.53 | 0.43 | 0.52 |
| + Ep. Memory | ✗ | **0.64** | **0.58** | **0.48** | **0.57** |
| **SPTM** [30] | ✓ | 0.35 | 0.27 | 0.19 | 0.27 |
| + LT Memory | ✓ | **0.64** | **0.58** | **0.48** | **0.57** |

Table 3: Comparison of the four ablated variants of our method (RL baseline, + Data Aug., + Ep. Memory, + LT Memory) with the state of the art (**NTS** [11] and **SPTM** [30]). We distinguish two different setups: with and without access to past experience. The metric shown is the SPL.

of the art methods for Image-Goal navigation: **NTS** [11] and **SPTM** [30]. We distinguish two different setups, that differ in the use of past experience. The results are shown in Table 3. As a first observation, we see that our method outperforms the state of the art on the two setups. Indeed, the version with Episodic Memory shows a large improvement over **NTS** [11] (+19% SPL *overall*), and this gap is even

more important on *hard* episodes (+26% SPL). For the second setup, with methods that uses past experience, we also observe a large improvement of our method with respect to the state of the art. Finally, we observe that the versions of our method with Episodic Memory and Long-term Memory perform similarly on this dataset. One possible explanation is that the episodes in this dataset (proposed by Chaplot *et al*.) are too easy to show the real improvement brought by the Long-Term Memory. This observation was a motivation for adding harder training and validation episodes in the *extra* dataset.

**Optimizing the learning rate.** Learning rate tuning is an important step for training our models. Figure 11 shows the *overall* SPL for the four ablated variants of our method and several values of the learning rate when trained on the standard dataset ( 11-a) and on the *extra* one ( 11-b). Each method was trained for three random seeds. We observe that the methods that contain external memory benefit a lot from decreasing the learning rate, as opposed to the data-augmented baseline.
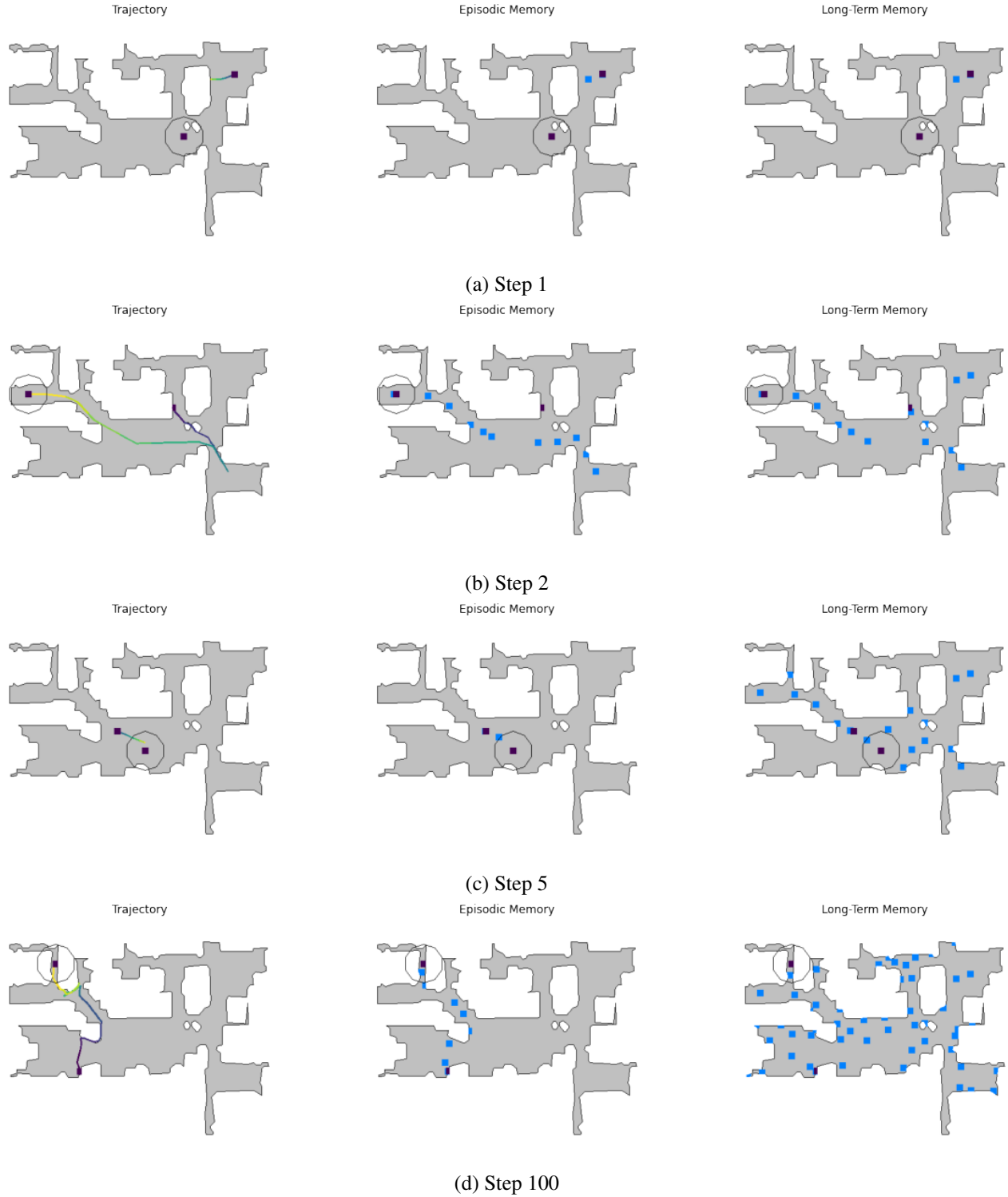
Figure 10: Visualisation of the agent's trajectory, episodic and long-term memories for first, second, fifth and 100th episode in the Eastville environment. The start and goal locations are shown in black, goal location being circled by a line showing the success area. The blue points represent the location of the episodic and long-term memory vectors. The episodic memory is reset after each episode, while the long-term memory remains for 100 episodes in the same scene.
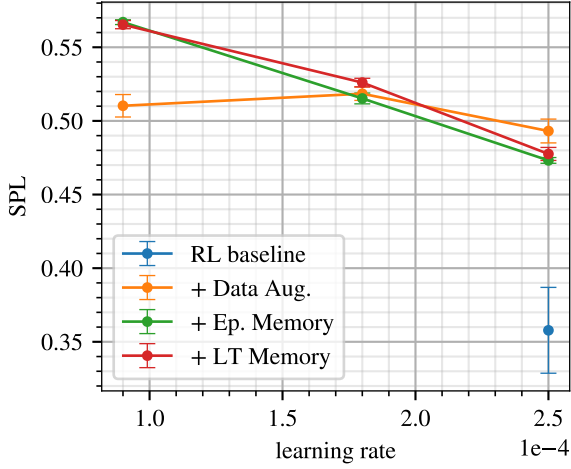
(a)
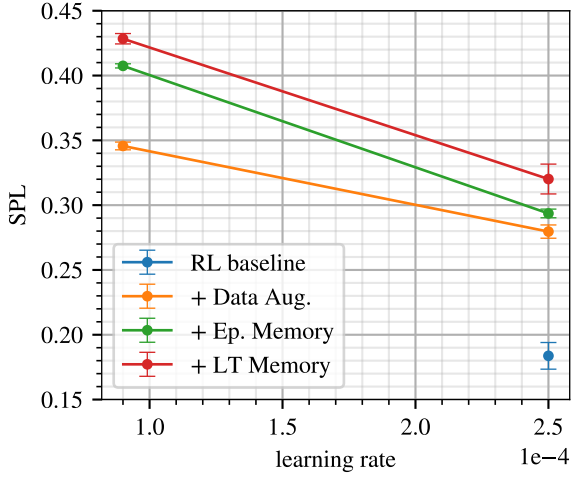


(b)

Figure 11: Overall SPL for several values of the learning rate. We plot the results for the four ablated variant of our method, when trained on the standard dataset (a) and on the extra one (b). Each experiment was ran for three random seeds. Methods with external memory benefit from a lower learning rate.

## C. SPTM [30] implementation details

As mentionned in the main paper, we compare our method to SPTM. In order to compare to this method, we need to adapt the code to the Habitat [32] environment. We use RGB observations of size $160 \times 120$ pixels. We train the Reachability and Locomotion networks with a learning rate of 1e-4 for 300 "epochs", with a batch size of 64. To evaluate it, we use an exploration trajectory obtained with a random policy rolled-out for 10k steps.
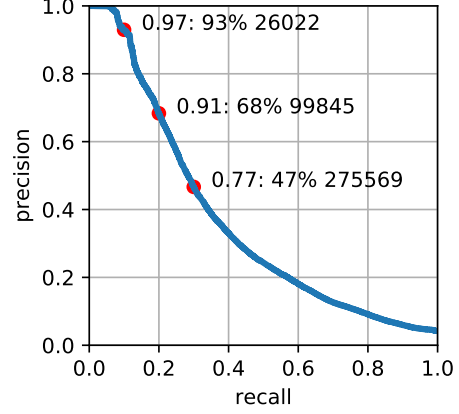


Figure 12: PR curve for edge prediction for **SPTM** [30]. We choose the number of edges to keep for navigation based on this evaluation. In our experiments, we kept the 100k top scoring edges.

In order to select the number of edges to infer, we look at the precision and recall of the Reachability Network trained in this context. We label all edges as a positive if it connects two nodes that are physically less than 1m apart. Given this label, and the scoring obtained from the Reachability Network, we plot the precision-recall curve that we present in Fig. 12.

On this plot, we see that if we set the threshold for the Reachability Network to 0.9, we obtain a precision of 68% for a recall of 20%. In that case we pick 99845 edges to be added to the graph. This precision and recall tradeoff leads to good performance, so we decided to keep 100k top scoring edges in our experiments.