Learning to Navigate in Mazes with Novel Layouts Using Abstract Top-down Maps

Linfeng Zhao[1], Aswin Shriram Thiagarajan[1], Lawson L.S. Wong[1]

***Abstract*— Learning navigation capabilities in different environments has long been one of the major challenges in decision-making. In this work, we focus on *zero-shot navigation ability* using given abstract 2-D top-down maps. Like human navigation by reading a *paper* map, the agent reads the map as an *image* when navigating in a novel layout, after learning to navigate on a set of training maps. We propose a model-based reinforcement learning approach for this multi-task learning problem, where it jointly learns a *hypermodel* that takes top-down maps as input and predicts the weights of the transition network. It disentangles the variations in map layout and goal location and enables longer-term planning ability for novel goals compared to reactive policies. We use the DeepMind Lab environment and customize layouts using generated maps. Our method can adapt better to novel environments in zero-shot and is more robust to noise.**

I. Introduction

If we provide a rough solution of a problem to a robot, can the robot learn to follow the solution effectively? In this paper, we study this question within the context of maze navigation, where an agent is situated within a maze whose layout has never been seen before, and the agent is expected to navigate to a goal without first training on or even exploring this novel maze. This task may appear impossible without further guidance, but we will provide the agent with additional information: an abstract 2-D top-down map, treated as an image, that illustrates the rough layout of the 3-D environment, as well as indicators of its start and goal locations ("abstract map" in Figure 1). This is akin to a tourist attempting to find a landmark in a new city: without any further help, this would be very challenging; but when equipped with a 2-D map of environment layout, the tourist can easily plan a path to reach the goal without needing to explore or train excessively.

Navigation is a fundamental capability of all embodied agents, both artificial and natural, and therefore has been studied under many settings. In our case, we are most concerned with *zero-shot navigation* in novel environments, where the agent cannot perform further training or even exploration of the new environment; all that is needed to accomplish the task is technically provided by the abstract 2-D map. This differs from the vast set of approaches based on simultaneous localization and mapping (SLAM) typically used in robot navigation [2], where the agent can explore and build an accurate *but specific* occupancy map of *each* environment *prior* to navigation. Recently, navigation approaches based on deep reinforcement learning (RL) approaches have also emerged, although they often require extensive training in the same environment [3], [4]. Some deep RL approaches are even capable of navigating novel environments with new goals or layouts without further training; however, these approaches typically learn the strategy of efficiently exploring the new environment to understand the layout and find the goal, then exploiting that knowledge for the remainder of the episode to repeatedly reach that goal quickly [5]. In contrast, since the solution is essentially provided to the agent via the abstract 2-D map, we require a more stringent version of zero-shot navigation, where it should *not* explore the new environment; instead, we expect the agent to produce a near-optimal path in its *first* (and only) approach to the goal.

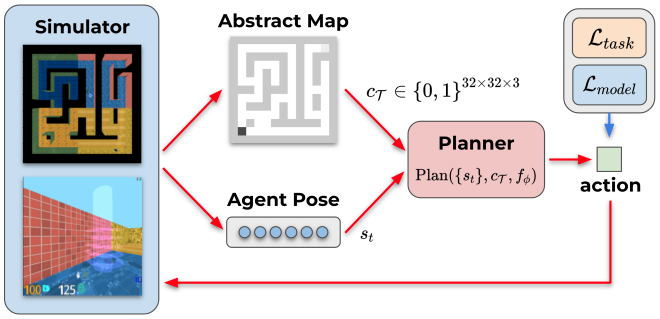


Fig. 1: We develop an agent that can perform zero-shot navigation on novel tasks $\mathcal{T}$ (unseen maps in the DeepMind Lab navigation simulator in blue box), without needing to first explore the new 3-D environment. Instead, the agent is given the *top-down view* as additional guidance: an abstract 2-D occupancy map, and a goal and start position (bottom left black dot and top right gray dot). Although the map provides a rough solution for navigation, the path cannot be directly followed due to the continuous nature of the agent's environment, as well as unknown map scale, inaccuracies in the map, and noisy localization.

The solution to navigation using the provided abstract map seems obvious: we should localize ourselves on the abstract map (image), plan a path, and simply follow it. However, this approach suffers from a key difficulty: determining the correspondence between 2-D image maps and 3-D environments. It is not obvious how to execute the abstract plan in practice because the state and action spaces are completely different, and may even be discrete in the abstract map but continuous in the real environment.

Instead, in this paper we explore an alternative approach that avoids explicitly localizing and planning on the abstract map. The key idea is to plan in a learned model that only considers the abstract map (and start/goal information) as contextual input, but does not directly plan on the map image

[1]Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115, USA. Correspondence zhao.linf@northeastern.edu.

A extended version available at http://lfzhao.com/map-nav [1].

itself. Specifically, we propose learning a *task-conditioned hypermodel* that uses the abstract map context to produce the *environment-specific parameters* (weights) of a latent-state transition dynamics model. We then perform planning by using sampling-based forward search on this task-specific dynamics model. Importantly, although the learned transition model operates in latent state space, it uses the agent's original action space, so that planned trajectories can be directly executed in the environment, without needing to solve the aforementioned correspondence problem. The hypermodel and the state encoder are learned in an end-to-end fashion, using loss functions that assess whether the learned components were able to support effective planning.

We refer to our method as the Map-conditioned Multi-task Navigator (**MMN**). We start with a model-based RL algorithm, MuZero [6], and introduce the above task-conditioned hypermodel based on HyperNetworks [7]. To tackle challenges in training, we additionally introduce an n-step generalization of Hindsight Experience Replay (HER) [8] and an auxiliary hypermodel loss. Additionally, we introduce a model-free RL baseline, named Map-conditioned Ape-X HER DQN (**MAH**). This method builds upon DQN [9] and augments the input with the provided abstract map, and uses standard single-step HER.

In experiments performed in DeepMind Lab [10], a 3-D maze simulation environment shown in Figure 1, we show that both approaches achieve effective zero-shot navigation in novel environment layouts, though the model-based **MMN** is significantly better at long-distance navigation. Additionally, whereas a baseline approach using deterministic path planning and reactive navigation quickly fails when the map is inaccurate or localization is noisy, our experiments suggest that **MMN** is significantly more robust to such noise.

II. Related work

Navigation is widely studied in robotics, vision, RL, and beyond; to limit the scope, we focus on zero-shot navigation in novel environments, which is most relevant to this work. This excludes traditional approaches based on SLAM [2], since those methods need to explicitly build a map before navigation, and the map can only be used for the corresponding environment and cannot be transferred to other layouts. Learning-based methods (e.g., [3], [4]) also require extensive training data from the same environment; they demonstrate generalization to new goals in the environment, but not transfer to new layouts. [5], [11], [12], [13] demonstrate agents that learn strategies to explore the new environment and potentially build maps of the environment during exploration; in contrast, we are interested in agents that do not need to explore the new environment. [12] learns to exploit semantic cues from its rich visual input, which is orthogonal to our work since we use the state directly. Other domains such as first-person-shooting games also involve agents navigating in novel environments [14], [15], [16], but since navigation is not the primary task in those domains, the agents may not need to actually reach the specified goal (if any). Most closely related to our work is [17], who also use 2-D occupancy maps as additional input and perform experiments in DeepMind Lab. Their approach is specific to map-based navigation, whereas our methodology aims to be less domain specific. [18] also use HyperNetworks, but they primarily focus on manipulation.

Our work is an instance of *end-to-end model-based planning* [19], [20], [6]. It has also been referred to as *implicit model-based planning* since the model is learned implicitly. It rolls out trajectories using a learnable transition model and jointly trains the *value and policy* networks along with the *transition* network. This is different from *decoupled* model learning and planning, such as *Dyna-style* [21]. One important distinction in end-to-end planning is whether the gradients are passed through the *planning computation*. For example, MuZero [6] uses sampling-based search method, Monte Carlo tree search (MCTS), that is hard to differentiate though. Other sampling-based approaches include [22], [23]. Another thread of work includes Value Iteration Networks and its variants [24], [25], [26], [27], which iteratively applies Bellman operators and is easily differentiable. They have also been used in end-to-end navigation, including CMP [28] and DAN [29]. However, they are limited to grid-like structure as the VIN backbone is 2-D convolution. Additionally, a body of work [30], [31], [32], [33], [34] studies learning structured latent models or representations useful for planning.

Our method is based on MuZero [6], but it has only been used on single map or goal because it learns purely from rewards. We additionally augment it with task conditioning (map and goal) to relax its dependence on task. [35] concurrently develop goal-relabeling for AlphaZero, but instead of jointly learning model and planning in MuZero, AlphaZero requires a given model, which limits its applicability to problems like maze navigation in continuous space.

III. Problem Statement

We consider a distribution of navigation tasks $\rho(\mathcal{T})$. Each task is different in two aspects: map layout and goal location. (1) *Abstract map*. The layout of each navigation task is specified by an abstract map. Specifically, an abstract map $m \in \mathbb{R}^{N\times N}$ is a 2-D occupancy grid, where cell with 1s (black) indicate walls and 0s (white) indicate nagivable spaces. A cell does not correspond to the agent's world, so the agent needs to learn to localize itself on an abstract 2-D map (i.e., to know which part of map it is currently at). We generate a set of maps and guarantee that any valid positions are reachable, i.e., there is only one connected component in a map. (2) *Goal position.* Given a map, we can then specify a pair of start and goal position. Both start and goal are represented as a "one-hot" occupancy grid $g \in \mathbb{R}^{2\times N\times N}$ provided to the agent. For simplicity, we use g to refer to both start and goal, and we denote the provided map and start-goal positions $c = (m, g)$ as the *task context*.

We formulate each navigation task as a goal-reaching *Markov decision process* (MDP), consisting of a tuple $\langle \mathcal{S}, \mathcal{A}, P, R_{\mathcal{G}}, \rho_0, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition probability function $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, $\rho_0 = \rho(s_0)$ is the initial state distribution,

and $\gamma \in (0, 1]$ is the discount factor. In the learning, we assume transitions are deterministic. For each task, the objective is to reach a subset of state space $\mathcal{S}_\mathcal{G} \subset \mathcal{S}$ indicated by a reward function $R_\mathcal{G} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We denote a task as $\mathcal{T} = \langle P, R_\mathcal{G}, \rho_0 \rangle$, since a map and goal specify the dynamics and reward function of a MDP, respectively. In the episodic goal-reaching setting, the objective is typically not discounted ($\gamma = 1$) and the reward is -1 for all non-goal states, i.e., $R_\mathcal{G}(s, a) = -\mathbb{I}[s \neq g], g \in \mathcal{S}_\mathcal{G}$.

We emphasize that although the abstract map's occupancy grid corresponds to the environmental layout, the correspondence between abstract "states" (grid cells) and agent states (pose and velocity) is not known in advance, and likewise for actions (grid-cell transitions vs. forward/backward/rotate). Furthermore, the learned correspondence may not be reliable due to inaccuracies in the abstract map and localization error.

IV. Learning to Navigate using Abstract Maps

This section presents an approach that can effectively use abstract maps (in image form) by end-to-end model-based planning based on MuZero [6]. We expect the agent to be able to efficiently *train* on multiple maps as well as *generalize* to new maps.

This poses several technical challenges. *(i)* A local change in map may introduce entirely different environment structure, so we need the model and planner to adapt to the task context in a different way than conditioning on state, and not directly condition on the entire task context. *(ii)* During training, we can only rely on a very small proportion of training tasks (e.g., 20 of 13×13 maps). This requires compositional generalization from existing map patches to novel combinations of patches. *(iii)* The reward signal is sparse, but model learning is done jointly and purely relies on reward signal. To this end, we first introduce the idea of using a *hypermodel* that learns to predict weights of transition model, instead of state output directly, to tackle *(i)* and *(ii)*. For the challenge *(iii)*, we use the idea from Hindsight Experience Replay (HER) [8] to reuse failure experience and also add an auxiliary loss of predicting transitions.

A. Task-conditioned hypermodel

Our goal is to create a transition model that accurately handles various map inputs, enabling planning in 3D environments with arbitrary layouts. In a single-task training schema, a straightforward approach would be to learn a parameterized transition function $f_i(s, a)$ for each individual map. However, we aim to leverage shared knowledge between navigation tasks, where maps often exhibit common local patterns and require the ability to generalize to recombination of known patterns. For instance, in Figure 2, moving right on the center of the box in the left map shares computation with the right map. This phenomenon also applies to larger map areas and reward prediction. By enabling the agent to recognize these local computational patterns, it can transfer to new tasks by compositional generalization.

We propose to build a *meta* network h_ψ, or *hypermodel*, to learn the "computation" of the transition model f_ψ simultaneously for all maps with abstract 2-D maps as input. The transition model for task $\mathcal{T}$ (map-goal pair) is a function f_i that maps current (latent) state and action to a next (latent) state. We parameterize a transition function f_i as a neural network with its parameter vector ϕ_i. The set $\{f_i\}$ represents transition functions of all tasks belonging to a navigation schema (e.g., a certain size of map), and these tasks have similar structure. This set of transition functions/networks are characterized by the context variables $c = (m, g)$, i.e., the abstract 2-D map and goal.[1] This implies that parameter vectors ϕ_i live in a low-dimensional manifold. Thus, we define a mapping $h : \mathcal{C} \to \Phi$ that maps the context of a task to the parameter vector ϕ_i of its transition function f_i, predicting state s' and reward r. We parameterize h also as a network with parameter ψ:[2]

$$h_\psi : c \mapsto \phi, \quad f_\phi : s, a \mapsto s', r. \tag{1}$$

This can be viewed as soft weight sharing between multiple tasks. It efficiently maps low-dimensional structure in the MDP, specified by the map, to computation of the transition model. It may also be viewed as a structured *learned* "dot-product" between task context $c_\mathcal{T}$ and state and action s_t, a_t to predict the next state. The idea of predicting the weights of a main network using another *meta*-network is also known as HyperNetworks [7], [36].

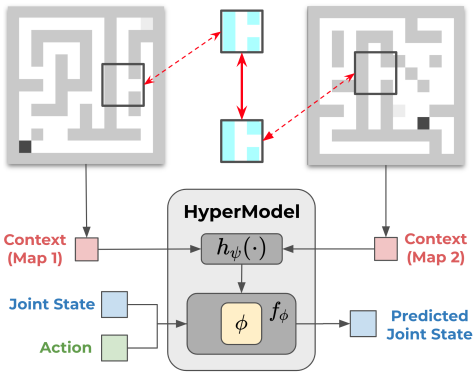


Fig. 2: Applying the hypermodel h_ψ on map m_1 and m_2 outputs two sets of transition network weights $\phi_1 = h_\psi(m_1, g_1)$ and $\phi_2 = h_\psi(m_2, g_2)$. Each transition network uses their weight ϕ_i to predict next state $f(s, a; \phi_i) = s'$ Since the maps may share local patterns at some scales (cropped 3×3 patches), they can be captured by the hypermodel h_ψ.

B. Planning using a learned hypermodel

Equipped with a map-conditioned model, we use it to search for actions according to the map layout and goal location: $(a^1, ..., a^k) = \text{Plan}(\{s_i\}, c, f_\phi)$. We follow MuZero [6] to use Monte-Carlo tree search (MCTS) to search

[1] Concretely, a task context $c \in \mathbb{R}^{4 \times N \times N}$ has four components: downsampled global occupancy map, cropped local occupancy map, and one-hot goal and start occupancy maps; N is downsampled size.

[2] We only predict weights of the transition model $f_\phi : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ which operates on a latent state space. The mapping from environment observations to latent states $e : \mathcal{O} \to \mathcal{S}$ is not predicted by a meta network. Since the latent space is low-dimensional, it is feasible to predict weight matrices of a transition network for it.

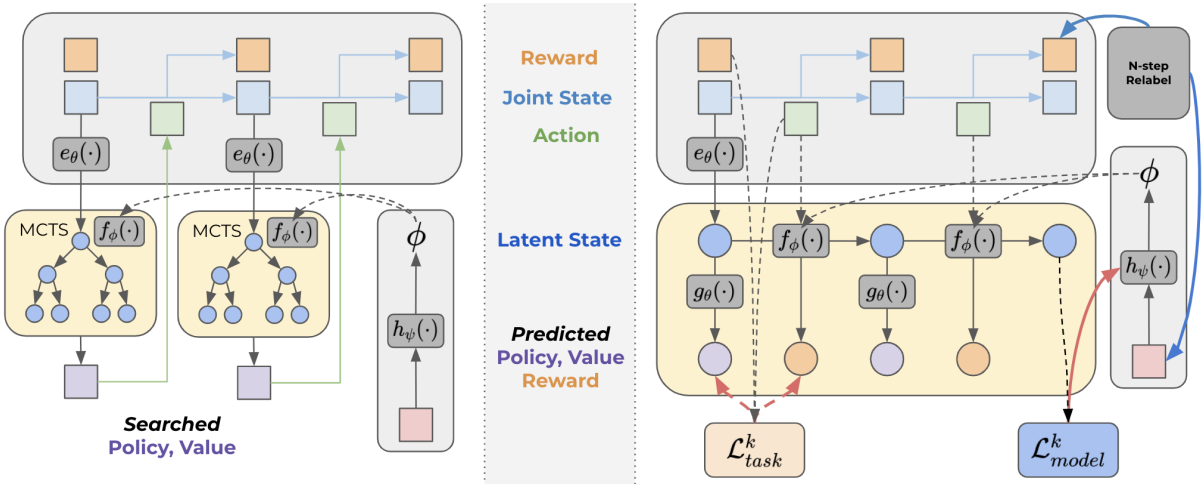


Fig. 3: We use yellow or circles to indicate predicted states or other quantities, and grey or squares from actual interactions. *(Left) Inference: search with learned model.* Applying MCTS with hypermodel to search for behavioral policy and value, and act with a sampled action. *(Right) Training: building learning targets.* Computing targets and backpropagating from loss. The blue line indicates n-step relabelling. We only illustrate backpropagation for one reward node for simplicity. The solid red line shows the gradient flow from auxiliary model loss to the meta-network's weight ψ. The dashed red line is the gradient from task loss.

with the learned hypermodel f_ϕ. The planner needs to act based on different task inputs, which necessitates a task-dependent value function that differs from the single-task setup in MuZero. Consequently, the planner $\text{Plan}(s_i, c, f_\phi)$ must strongly correlate its computation with the map and goal input $c = (m, g)$, which presents a challenge for model-free reactive agents.

As shown in Figure 3 (left), we begin by encoding the observed joint state o_t into a latent space s_t using the learned encoder $e_\theta(o_t)$. This serves as the root node of the search tree (top blue circle). To predict the next state given a latent state and a candidate action, we use the hypermodel f_ϕ. For each state (blue circle nodes), we use another network $g_\theta(s_t, c)$ to predict the policy π_t and value function v_t (not shown). These networks guide the search, where the value network estimates the future value and the policy network provides candidate actions for rollout in MCTS (blue circles), as described in [6]. During training, they are trained to minimize the loss with searched values and actions. Once a number of MCTS simulations are completed (yellow rounded boxes), we backup the statistics to the root node and sample an action (green boxes) from the searched action distribution (purple boxes). The trajectory and corresponding abstract map and goal $(c_\mathcal{T}, \{s_t, a_t, r_t, s_{t+1}\}_t)$ are saved to a centralized replay buffer for training.

At **zero-shot evaluation** time, given a new abstract map, we plan with the trained hypermodel: (1) given a map and goal $c_\mathcal{T} = (m_\mathcal{T}, g_\mathcal{T})$, at the beginning of the episode, compute the hypermodel weights $\phi = h(c; \psi)$ by applying the meta-network on the task context $c_\mathcal{T}$, (2) start MCTS simulations using the hypermodel $f(s, a; \phi)$ for latent state predictions, (3) get an action and transit to next state, and go to step (2) and repeat. Moreover, if we assume access to a *landmark oracle* on given maps, we can perform **hierarchical navigation** by generating a sequence of local subgoals $\{(m, g_i)\}_{i=1}^n$, and plan to sequentially achieve each landmark; see Section V-C for more details.

C. n-step Goal Relabelling: Denser Reward

Jointly training a planner with learned model can suffer from lack of reward signal, especially when the model training entirely relies on reward from *multiple tasks*, which is common in model-based agents based on *value gradients* [6], [20]. Motivated by this, we introduce a straightforward strategy to enhance the reward signal by implicitly defining a learning curriculum, named *n-step hindsight goal relabelling*. This generalizes the single-step version of *Hindsight Experience Replay* (HER) [8] to *n-step return relabeling*.

Motivation. As shown in Figure 3 (right), we sample a trajectory of experience $(c_\mathcal{T}, \{s_t, a_t, r_t, s_{t+1}\}_t)$ on a specific map and goal $c_\mathcal{T} = (m_\mathcal{T}, g_\mathcal{T})$ from the replay buffer. Observe that, if the agent does not reach the goal area $\mathcal{S}_\mathcal{G}$ (a 100×100 cell in the agent's 3-D environment, denoted by a 2-D position $g_\mathcal{T}$ on the abstract 2-D map), it will only receive reward $r_t = -1$ during the entire episode until timeout. In large maps, this hinders the agent to learn effectively from the current map $m_\mathcal{T}$. Even if the agent partially understands a map, it would rarely experiences a specific goal area on the map again.[3] This is more frequent on larger maps in which possible navigable space is larger.

Relabelling n-step returns. Motivated by single-step HER, we relabel failed goals to randomly sampled *future* states (visited area) from the trajectory, and associating states with the relabelled n-step return. Concretely, the *task-conditioned* bootstrapped n-step return is

$$G_t^\mathcal{T} \doteq r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n v_n^\mathcal{T}, \quad \left[v_n^\mathcal{T}, \pi_n^\mathcal{T}\right] = g_\theta(s_t, c_\mathcal{T}) \tag{2}$$

[3]In our *extremely* low data regime, the agent only has one start-goal pair on a small set of map. While on *low* data regime, the agent can train on randomly sampled pairs on the maps. See the Setup for more details.

where $v_n^{\mathcal{T}}$ is the state-value function *bootstrapping* n steps into the future from the search value and *conditional* on task context $c_{\mathcal{T}}$. This task-conditioned value function is *asymmetric* since $\mathbb{R}^{12} = \mathcal{S} \neq \mathcal{S}_g = \mathbb{R}^2$.

Steps. To relabel the task-conditioned bootstrapped n-step return, there are three steps, demonstrated by the blue lines from "N-step Relabel" box. (1) *Goal (red boxes).* Randomly select a reached state $s_t \in \mathbb{R}^{12}$ from the trajectories, then take the 2-D position $(x, y) \in \mathbb{R}^2$ in agent world and convert it to a 2-D goal support grid $g_{\mathcal{T}_S}$. Then, relabel the *goal* in task context $c_{\mathcal{T}_S} = (m_{\mathcal{T}}, g_{\mathcal{T}_S})$, keeping the abstracted map and start position unchanged. (2) *Reward (orange boxes).* Recompute the rewards along the n-step segment. In episodic case, we need to terminate the episode if the agent can reach the relabelled goal area $g_{\mathcal{T}_S}$, by marking "done" at the certain timestep or assigning zero discount after that step $\gamma_t = 0$ to mask the remaining segment. (3) *Value (purple circles).* Finally, we need to recompute the bootstrapping task-conditioned value $v_n^{\mathcal{T}_S}, \pi_n^{\mathcal{T}_S} = g_\theta(s_t, c_{\mathcal{T}_S})$.

Empirically, this strategy significantly increases the efficiency of our multi-task training by providing smoothing gradients when sampling a *mini-batch* of n-step targets from successful or failed tasks. It can also be applied to other multi-task agents based on n-step return.

D. Joint optimization: Multi-task Value Learning

Our training target has two components. The first component is based on the value gradient objective in MuZero [6], [20], using relabelled experiences from proposed n-step HER. It is denoted by $\mathcal{L}^k_{\text{task}}$ for step $k = 1, \ldots, K$. However, this loss is only suitable for single-task RL.

Thus, we propose an auxiliary model prediction loss, denoted by $\mathcal{L}^k_{\text{model}}$ in Figure 3 (right). The motivation is to regularize that the hypermodel $f_\phi(s, a, h_\psi(c_{\mathcal{T}}))$ should predict trajectory based on the information of given abstract map and goal $c_{\mathcal{T}}$. The objective corresponds to maximizing the mutual information between task context $c_{\mathcal{T}}$ and *predicted* trajectories $\hat{\tau}_{\mathcal{T}}$ from the hypermodel on sampled tasks $\mathcal{T} \sim \rho(\mathcal{T})$:

$$\max_{h_\psi} \mathbb{E}_{\mathcal{T}\sim\rho(\mathcal{T})} \left[I(c_{\mathcal{T}}; \hat{\tau}_{\mathcal{T}}) \right], \tag{3}$$

where $h_\psi(c_{\mathcal{T}}) = \phi$ is the meta network predicting the weight of transition network f_ϕ. Observe that: $I(\tau; c) = H(\tau) - H(\tau|c) \geq H(\tau) + \mathbb{E}_{\tau,c}[\log q(\tau|c)]$, we can equivalently optimize the RHS $\max_h \mathbb{E}_{\mathcal{T}} [\log q(\tau|c)] \iff \max_h \mathbb{E}_{(s,a,s')} [\log q(s'|s, a; h(c))]$ (subscripts omitted). This objective is equivalent to minimizing the loss between predicted states and true states from environment, for all transition tuples across all tasks. The final loss is given by the sum over multiple steps:

$$\mathcal{L}(\psi, \phi, \theta) = \sum_{k=1}^{n} \mathcal{L}^k_{\text{task}} + \mathcal{L}^k_{\text{model}}, \tag{4}$$

where $k = 1, ..., K$, and K is the length of training segment.

V. Experiments

In the experiments, we assess our method and analyze its performance on DeepMind Lab [10] maze navigation environment. We focus on zero-shot evaluation results, while more results are available in the extended version [1].

A. Experimental setup

We perform experiments on DeepMind Lab [10], an RL environment suite supporting customizing 2-D map layout. As shown in Figure 1, we generate a set of abstract 2-D maps, and use them to generate 3-D environments in DeepMind Lab. Each cell on the abstract map corresponds to 100 units in the agent world. In each generated map, all valid positions are reachable, i.e., there is only one connected component in the map. Given a sampled map, we then generate a start-goal position within a given distance range. Throughout each task, the agent receives the abstract map and start/goal location indicators, the joint state vector $o \in \mathbb{R}^{12}$ (consisting of position $\mathbb{R}^3$, orientation $\mathbb{R}^3$, translational and rotational velocity $\mathbb{R}^6$), and reward signal r. The action space is {forward, backward, strafe left, strafe right, look left, look right}, with an action repeat of 10. This means that, at maximum forward velocity, the agent can traverse a 100×100 block in two steps, but typically takes longer because the agent may slow down for rotations. In the experiments of Section V-D, the agent receives a noisy version of the state vector to simulate effects of imperfect localization.

a) Training settings: We train a set of agents on a variety of training settings, which have several key options: (1) *Map size*. We mainly train on sets of $13\times13, 15\times15, 17\times17, 19\times19, 21\times21$ maps. One cell in the abstract map is equivalent to a 100×100 block in the agent's world. (2) *Goal distance*. During training, we generate local start-goal pairs with distance between 1 and 5 in the abstract map. (3) *Map availability*. For each map size, we train all agents on the same set of 20 generated maps, with different randomly sampled start-goal pairs in each episode.

b) Evaluation settings: We have several settings for evaluation: (1) *Zero-shot transfer*. We mainly study this type of generalization, where the agent is presented with 20 unseen evaluation maps, and has to navigate between randomly generated start-goal pairs of varying distances. (2) *Goal distance* on abstract map. We consider both *local* navigation and *hierarchical* navigation. In the *local* case, we evaluate on a range of distances ($[1, 15]$) on a set of maps, while in the *hierarchical* case, we generate a set of landmarks with a fixed distance of 5 between them and provide these to agents sequentially. (3) *Perturbation*. To understand how errors in the abstract map and in localization affects performance, we evaluate agents with maps and poses perturbed by different strategies.

c) Evaluation metrics: We mainly report success rate and (approximate) SPL metric [37] with 95% confidence intervals (higher SPL is better). We further explain the metrics in the extended version [1]. We report results from fully trained agents to compare asymptotic performance; no training is performed on evaluation maps.

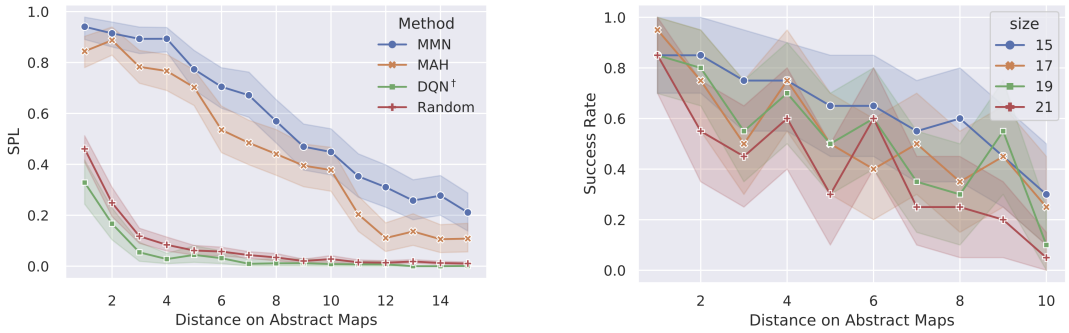


Fig. 4: **(Left)** Zero-shot evaluation performance on 13×13 maps. Local navigation with different distances between start and goal, from 1 to 15. **(Right)** Performance of our method on larger maps.

d) Methods: We compare our model-based approach against two model-free baselines and other methods. More details are available in the extended version [1].

1) *Map-conditioned Multi-task Navigator* (**MMN**), model-based. Our map-conditioned planner based on MuZero and improved with n-step HER and multi-task training.
2) *Map-conditioned Ape-X HER DQN* (**MAH**), model-free. A reactive baseline based on Ape-X DQN [38] and single-step HER [8], *conditioned* on map and goal.
3) *Single-task Ape-X HER DQN* (**DQN**†), model-free, based on Ape-X DQN [38] and single-step HER [8]. No task context c (map or goal) as input.
4) *Hand-crafted planner*. In the experiments of Section V-D, we create a planner with deterministic path planning and reactive navigation, to study robustness to noise in the abstract map and in localization.
5) *Random*, a reference of the navigation performance.

We also tested on *Map-Planner* [39], which combines graph search and DQN, but it is not suitable for map-conditioned setup and requires a different evaluation process. More details are in the extended version [1].

B. Zero-shot local navigation in novel layouts

For zero-shot generalization of *locally trained agents*, we *train* all four agents on 20 of 13×13 maps with randomly generated local start-goal pairs with distance $[1, 5]$ in each episode. We train the agents until convergence; MAH typically takes $3\times$ more training episodes and steps. We *evaluate* all agents on 20 of *unseen* 13×13 maps and generate 5 start-goal pairs for each distance from 1 to 15 on each map. The results are shown in Figure 4 left. MMN and MAH generally outperforms the other two baselines. MMN has better performance especially over longer distances, both in success rate and successful-trajectory length, even though it was only trained on distances ≤ 5. Since we compare fully trained agents, we found MMN performs asymptotically better than MAH. Additionally, as shown in Figure 4 right, we also train and evaluate MMN on larger maps from 15×15 to 21×21. Observed with similar trend to 13×13 maps, when trained with start-goal distance ≤ 5, the agent will find distant goal and larger maps more difficult.

C. Hierarchical navigation in novel layouts

This section performs hierarchical navigation experiment, which provides an additional *landmark oracle* to generate sequences of subgoals between long-distance start-goal pairs, and evaluate the performance of hierarchical navigation. The agent is trained on 13×13 maps, and evaluate on 20 of 13×13 unseen maps. On each map, we use the top-right corner as the global start position and the bottom-left corner as the global goal position, then plan a shortest path in the abstract 2-D map, and generate a sequence of subgoals with distance 5 between them; this typically results in 3 to 6 intermediate subgoals. Consecutive subgoal pairs are provided sequentially to the agent as local start-goal pairs to navigate. The navigation is considered successful only if the agent reaches the global goal by the end.

TABLE I: Hierarchical Navigation of distance between the *landmarks* in distances from 1 to 5, using SPL metric and success rate (SR, only for distance 5). Landmarks are generated sub-goals between fixed start-goal pairs on 20 maps.

Landmark Distance	1	2	3	4	5	5 (SR)
MMN	**0.61**	**0.59**	**0.68**	**0.45**	**0.63**	**0.80**
MAH	0.24	0.42	0.45	0.41	0.28	0.45
DQN†	0.00	0.00	0.00	0.00	0.00	0.00
Random	0.00	0.00	0.00	0.00	0.00	0.00

We evaluated MMN and MAH on the 20 evaluation maps. We provide the next subgoal when the current one is *reached* or until *timeout*. As shown in Table I, our model-based MMN outperforms the model-free counterpart by a large margin. MMN can reach 16 out of 20 global goals, which include all 9 successful cases of MAH. We visualize five trajectories of zero-shot hierarchical navigation in Figure 5. The model-based MMN is more robust to the intermediate failed subgoals by navigating to the new subgoal directly, where the model-free MAH gets stuck frequently.

D. Robustness to map and localization errors

To further study the robustness of our method and the importance of each component, we considered breaking three components in closed-loop map-based navigation: Map – (1) $\rightarrow$ Path – (2) $\rightarrow$ Environment – (3) $\rightarrow$ Map (repeat). In general, our learning-based agent is robust to these changes.

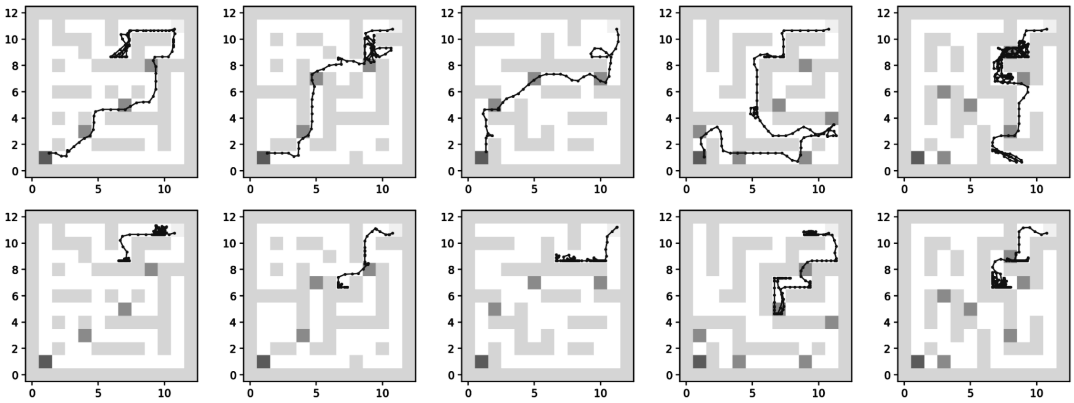


Fig. 5: Trajectories from hierarchical navigation in zero-shot on 13×13 maps. Since there is a fixed scaling factor from maps to environments, we can compute the corresponding location on the abstract map and visualize trajectories, while the agent *does not* know this domain knowledge. The top-right corner is the start, and the bottom-left is the goal. Other darker cells are generated subgoals with distance 5. The top row is for MMN and bottom row is for MAH. For the first 4 tasks (columns), MMN successfully reached the goals, while MAH failed. Both methods failed in the last task.

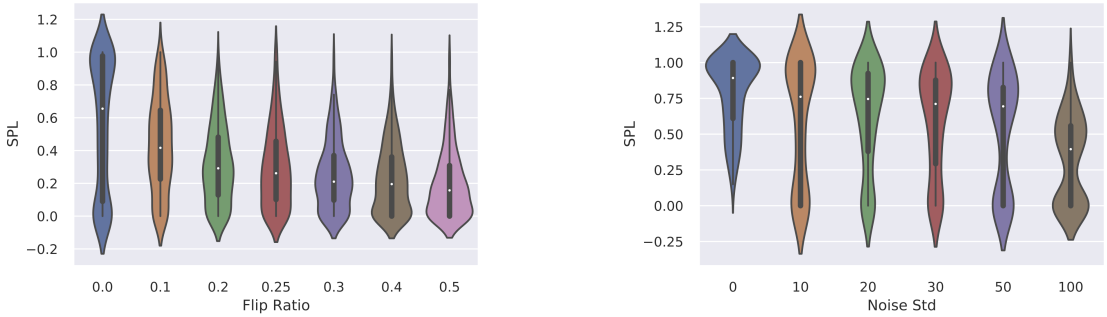


Fig. 6: Violin plots show the SPL of MMN with different map flip ratio **(left)** and localization noise level **(right)**. The two figures clearly show the negative impact of imperfect information, which also justify the importance of the guidance.

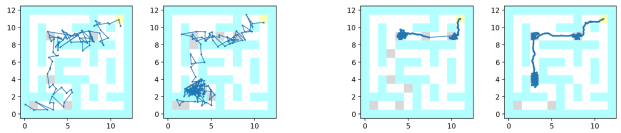


Fig. 7: (**left pair**) MMN visualized with *perturbed* locations; even though the provided state is noisy, MMN successfully reaches the goal. (**right pair**) Deterministic planner is unable to reach the goal when the provided state is noisy. (We only show the *unperturbed* locations in this case for clarity in visualization.) MMN still reaches the goal with 50 units of noise (0.5 cell), while the deterministic planner gets stuck at some subgoals or runs out of budget.

To illustrate the difficulty of the problem, we considered a hard-coded strategy (hand-crafted deterministic planner) based on perfect information of the environment (e.g., can plan on map) for comparison correspondingly: (1) known perfect maps and intermediate landmarks, (2) scaling factor (unavailable to MMN), and (3) world position on map. Since we assume that it has perfect localization and landmarks, the key step is to reach a landmark given current location, which consists of several procedures: (a) change the orientation to the target landmark, (b) move forward along the direction, and (c) stop at the target cell as soon as possible.

a) Perturbing planning: We try to break the implicit assumption of requiring perfect abstract map information. We adopt the hierarchical setting, but generating subgoals on perturbed maps, where some proportion of the map's occupancy information is flipped. In Figure 6 (left), as the perturbation level increases, MMN's performance gradually decreases, but it still navigates successfully with significant noise levels. More results in the extended version [1].

TABLE II: Success rate for perturbing action mapping, comparing with unperturbed MMN for reference.

Goal Distance	2	4	6	8	10
MMN (Perturbed)	0.80	0.85	0.71	0.40	0.36
MMN (Default)	0.91	0.90	0.71	0.58	0.43

b) Perturbing action mapping: We break the implicit requirement of known scaling between map and environment. We provide the agent with randomly transformed maps with random perspective transformation, where the ratio (in both x and y directions) is different. As shown in Table II, perturbed MMN's performance decreases gracefully compared to unperturbed one, which shows that our agent rely little on this knowledge or any perfect relation.

c) Perturbing location: We break the identifiability of agent position (a part of its joint state) by applying random noise to given position. We aim to show that our agent

does not rely on the position to understand the map, since providing position in the agent world has no relation with localizing on abstract maps and our learning-based method can adapt to the noise. In Figure 6 (right), even though MMN is trained without noise, it tolerates some amount of noise and maintains relatively high SPL even at 50 units of noise (corresponding to 0.5 cell width). In Figure 7, we visualize the trajectories of MMN and the deterministic planner to qualitatively demonstrate MMN's robustness to noise.

VI. Conclusion

In this work, we have presented an end-to-end model-based approach, MMN, for enabling agents to navigate in environments with novel layouts. By using provided abstract 2-D maps and start/goal information, MMN does not require further training or exploration (zero-shot). Compared to the map-conditioned model-free counterpart MAH, both approaches performed well in zero-shot navigation for short distances; for longer distances (with access to a landmark oracle), our model-based approach MMN performed significantly better. In future work, we will explore learned subgoal generator, extend this work to handle visual observation input, and perform navigation in rich visual environments.

References

[1] L. Zhao, A. S. Thiagarajan, and L. L. Wong, "Full version of learning to navigate in mazes with novel layouts using abstract top-down maps," http://lfzhao.com/map-nav.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, 2005.

[3] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," in *ICLR*, 2017.

[4] P. Mirowski, M. Grimes, M. Malinowski, K. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, A. Zisserman, R. Hadsell, *et al.*, "Learning to navigate in cities without a map," in *NeurIPS*, 2018.

[5] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *ICLR*, 2017.

[6] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *arXiv:1911.08265*, 2019.

[7] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv:1609.09106*, 2016.

[8] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *NeurIPS*, 2017.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. 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*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://www.nature.com/articles/nature14236

[10] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, *et al.*, "Deepmind lab," *arXiv:1612.03801*, 2016.

[11] T. Chen, S. Gupta, and A. Gupta, "Learning exploration policies for navigation," in *ICLR*, 2019.

[12] S. Gupta, V. Tolani, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," *International Journal on Computer Vision*, 2019.

[13] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural slam," in *ICLR*, 2020.

[14] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *AAAI*, 2017.

[15] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," in *ICLR*, 2017.

[16] V. Zhong, T. Rocktäschel, and E. Grefenstette, "RTFM: Generalising to new environment dynamics via reading," in *ICLR*, 2020.

[17] G. Brunner, O. Richter, Y. Wang, and R. Wattenhofer, "Teaching a machine to read maps with deep reinforcement learning," in *AAAI*, 2018.

[18] Y. Huang, K. Xie, H. Bharadhwaj, and F. Shkurti, "Continual model-based reinforcement learning with hypernetworks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 799–805.

[19] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *NeurIPS*, 2016.

[20] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *NeurIPS*, 2017.

[21] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep rl for model-based control," *arXiv:1802.09081*, 2018.

[22] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," *arXiv:1811.04551*, 2018.

[23] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *NeurIPS*, 2018.

[24] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value Iteration Networks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, Aug. 2017, pp. 4949–4953. [Online]. Available: https://www.ijcai.org/proceedings/2017/700

[25] L. Lee, E. Parisotto, D. S. Chaplot, E. Xing, and R. Salakhutdinov, "Gated Path Planning Networks," *arXiv:1806.06408 [cs, stat]*, June 2018, arXiv: 1806.06408. [Online]. Available: http://arxiv.org/abs/1806.06408

[26] L. Zhao, X. Zhu, L. Kong, R. Walters, and L. L. S. Wong, "Integrating Symmetry into Differentiable Planning," in *ICLR 2023*. ICLR, June 2022, arXiv:2206.03674 [cs] type: article. [Online]. Available: http://arxiv.org/abs/2206.03674

[27] L. Zhao, H. Xu, and L. L. S. Wong, "Scaling up and Stabilizing Differentiable Planning with Implicit Differentiation," in *ICLR 2023*, Feb. 2023. [Online]. Available: https://openreview.net/forum?id=PYbe4MoHf32

[28] S. Gupta, V. Tolani, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive Mapping and Planning for Visual Navigation," *arXiv:1702.03920 [cs]*, Feb. 2019, arXiv: 1702.03920. [Online]. Available: http://arxiv.org/abs/1702.03920

[29] P. Karkus, X. Ma, D. Hsu, L. P. Kaelbling, W. S. Lee, and T. Lozano-Perez, "Differentiable Algorithm Networks for Composable Robot Learning," *arXiv:1905.11602 [cs, stat]*, May 2019, arXiv: 1905.11602. [Online]. Available: http://arxiv.org/abs/1905.11602

[30] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," *arXiv:1702.08360*, 2017.

[31] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. Chadwick, T. Degris, J. Modayil, *et al.*, "Vector-based navigation using grid-like representations in artificial agents," *Nature*, vol. 557, no. 7705, pp. 429–433, 2018.

[32] M. Fortunato, M. Tan, R. Faulkner, S. Hansen, A. Badia, G. Buttimore, C. Deck, J. Leibo, and C. Blundell, "Generalization of reinforcement learners with working and episodic memory," in *NeurIPS*, 2019.

[33] G. Wayne, C.-C. Hung, D. Amos, M. Mirza, A. Ahuja, A. Grabska-Barwinska, J. Rae, P. Mirowski, J. Leibo, A. Santoro, *et al.*, "Unsupervised predictive memory in a goal-directed agent," *arXiv:1803.10760*, 2018.

[34] X. Ma, P. Karkus, D. Hsu, W. S. Lee, and N. Ye, "Discriminative particle filter reinforcement learning for complex partial observations," *arXiv:2002.09884*, 2020.

[35] L. Moro, A. Likmeta, E. Prati, and M. Restelli, "Goal-Directed Planning via Hindsight Experience Replay," Sept. 2021. [Online]. Available: https://openreview.net/forum?id=6NePxZwfae

[36] J. von Oswald, C. Henning, J. Sacramento, and B. F. Grewe, "Continual learning with hypernetworks," *arXiv:1906.00695*, 2019.

[37] P. Anderson, A. 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," 2018.

[38] D. Horgan, J. Quan, G. Barth-Maron, and M. Hessel, "DISTRIBUTED PRIORITIZED EXPERIENCE REPLAY," p. 19, 2018.

[39] Z. Huang, F. Liu, and H. Su, "Mapping state space using landmarks for universal goal reaching," in *NeurIPS*, 2019, pp. 1940–1950.

VII. Appendix Overview

We provide additional details in the appendix, including additional description of our approach, experimental setup, and experimental studies.

(Section VIII) We introduce the details of *n-step relabelling*, joint training, and implementation of hypermodels. **(Section IX)** We also explain our computation of SPL metric and the methods we compare with. **(Section X)** We provide additional results in *training* and *generalization*, including the results of [39] (`map-planner`). **(Section XI and XII)** We give a more complete study of *robustness* of our method compared to a manually-designed method, including additional visualization of the trajectories.

VIII. Further Algorithm Details

A. Details of n-step Relabelling

As shown in Figure 8, we sample a trajectory of experience $(c_{\mathcal{T}}, \{s_t, a_t, r_t, s_{t+1}\}_t)$ on a specific map and goal $c_{\mathcal{T}} = (m_{\mathcal{T}}, g_{\mathcal{T}})$ from the replay buffer. Observe that, if the agent does not reach the goal area $\mathcal{S}_{\mathcal{G}}$ (a 100×100 cell in the agent space denoted by a coordinate $g_{\mathcal{T}}$ on the abstract 2-D map), it will only receive reward $r_t = -1$ during the entire episode until timeout. In large maps, this hinders the agent to learn effectively from the current map $m_{\mathcal{T}}$. Even if the agent partially understands a map, it would rarely experiences a specific goal area on the map again.[4] This is more frequent on larger maps in which possible navigable space is larger.

We adopt a multi-step strategy motivated by single-step HER, by relabelling failed goals to randomly sampled *future* states (visited area) from the trajectory. To relabel the task-conditioned bootstrapped n-step return, there are three steps. (1) *Goal.* Randomly select a reached state $s_t \in \mathbb{R}^{12}$ from the trajectories, then take the 2-D position $(x, y) \in \mathbb{R}^2$ in agent world and convert it to a 2-D goal support grid $g_{\mathcal{T}_S}$. Then, relabel the *goal* in task context $c_{\mathcal{T}_S} = (m_{\mathcal{T}}, g_{\mathcal{T}_S})$, keeping the abstracted map and start position unchanged. (2) *Reward.* Recompute the rewards along the n-step segment. In episodic case, we need to terminate the episode if the agent can reach the relabelled goal area $g_{\mathcal{T}_S}$, by marking "done" at the certain timestep or assigning zero discount after that step $\gamma_t = 0$ to mask the remaining segment. (3) *Value.* Finally, we need to recompute the bootstrapping task-conditioned value $v_n^{\mathcal{T}_S}, \pi_n^{\mathcal{T}_S} = g_\theta(s_t, c_{\mathcal{T}_S})$. Empirically, this strategy significantly increases the efficiency of our multi-task training by providing smoothing gradients when sampling a *mini-batch* of n-step targets from successful or failed tasks. It can also be applied to other multi-task agents based on n-step return.

B. Jointly Training Hypermodels

In the off-policy implementation, since we need to sample a mini-batch of trajectories, we have a *batch* of different contexts $[c_1, c_2, ..., c_n]$ during *multi-task* training, and need to generate a batch of weight $[\phi_1, \phi_2, ..., \phi_n]$ to compute each mini-batch gradient. To efficiently implement this, it is beneficial to use batch matrix multiplication in computing $[\phi_1, ..., \phi_n] = h_\psi([c_1, ..., c_n])$ and $[s'_1, ..., s'_n] = f([s_1, ..., s_n], [a_1, ..., a_n]; [s_1, \phi_2, ..., \phi_n])$ on a batch of randomly sampled transitions on different maps.

C. Additional Details of Hypermodels

We assume input is M-dimensional vectors in a mini-batch of size B, i.e., input tensor is $\mathbb{R}^{B \times M}$. The main network is a MLP with hidden units L_i in layer i. We denote the input as layer 0, i.e., $L_i = M$. Thus, the weight matrix from layer $i-1$ to layer i has size $W_i \in \mathbb{R}^{L_{i-1} \times L_i}$. We assume the task context is Z-dimentional and in a mini-batch of input each has a different task context associated with it, i.e., input tensor to the hypermodel is in $\mathbb{R}^{B \times Z}$. The hyperwork is also a MLP with hidden units K_i in each layer. The last hidden layer outputs a tensor $\mathbb{B} \times N$, where N is the dimension of output embedding for generating weights of the main network. The final generation layer has multiple branches. For generating weight tensor $W_i \in \mathbb{R}^{L_{i-1} \times L_i}$ of the main network *for all task contexts*, the branch outputs multi-task weight tensor $B \times L_{i-1} \times L_i$ and thus the mapping $h_{out} : B \times N \to B \times L_{i-1} \times L_i$ has weight tensor with dimensions $N \times (L_{i-1} \times L_i)$.

D. Architecture of Implementation

Aiming at fair comparability of our method MAH with the model-free method MAH, we implement them in a unified framework. Both ours (model-based) and model-free baselines have N actor workers, a single learner, and a centralized buffer worker. Each actor worker has a copy of the environment instance running single-threaded and take actions using either MCTS or a Q-network.

IX. Evaluation Details

A. Evaluation Metric: SPL

We used SPL (Successful Path Length) metric [37] as a measure to compare and validate the performance of our algorithm. In the computation of SPL, it requires the optimal path length. However, it is non-trivial to compute the length in agent's environment, since it is continuous and requires motion planning with kinedynamic constraints. Instead, we estimate the (approximate) SPL metric on the agent's environment in terms of number of steps travelled in the real world.

The shortest path length was estimated by mapping the distance travelled in the abstract map (in number of cells) to the agent's world (in steps), thus we can avoid to determine the shortest path directly in the agent's world. Instead, We can estimate the shortest path in a 2D abstract world (where we call it *landmark oracle*) by using known optimal path algorithms like Dijkstra's Algorithm, A*, etc. Once we obtained the data of distances in both the environment and the map by *manually* selecting *near-optimal* trajectories from MMN, we developed a linear regression to determine the dependency relationship. The selected trajectories are closed

[4] In our *extremely* low data regime, the agent only has one start-goal pair on a small set of map. While on *low* data regime, the agent can train on randomly sampled pairs on the maps. See the Setup for more details.

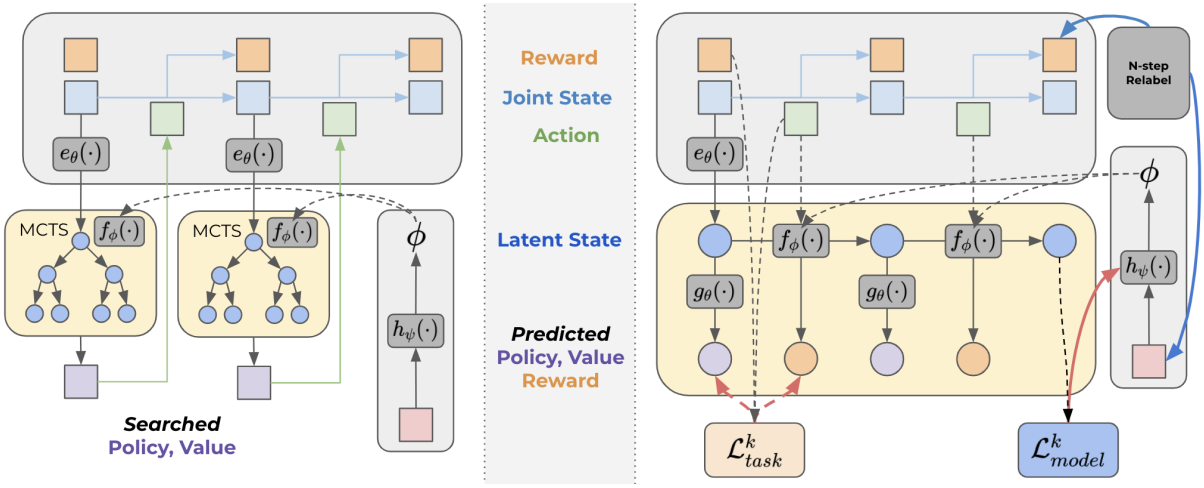


Fig. 8: *For completeness, we also include the same figure here.* We use yellow or circles to indicate predicted states or other quantities, and grey or squares from actual interactions. *(Left) Inference: search with learned model.* Applying MCTS with hypermodel to search for behavioral policy and value, and act with a sampled action. *(Right) Training: building learning targets.* Computing targets and backpropagating from loss. The blue line indicates n-step relabelling. We only illustrate backpropagation for one reward node for simplicity. The solid red line shows the gradient flow from auxiliary model loss to the meta-network's weight ψ. The dashed red line is the gradient from task loss.

to optimal, but the approximate SPL can also be viewed as an upper bound of optimal length or normalized performance (even our estimate is not accurate). Upon solving the linear regression, we got,

$$D_e = 2.090 \times D_a, \tag{5}$$

where D_e is distance (number of steps) travelled in agent's environment, and D_a is the path length in 2D abstract map.

Thus, the (approximate) SPL metric is given by

$$\frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)}, \tag{6}$$

where l_i is (estimated) shortest path length, p_i path length of some algorithm, and S_i success percentage. The SPL value is within the range $[0, 1]$.

Hence, we call this metric as *approximate* SPL. Higher (approximate) SPL means better performance.

B. Details about Compared Methods

We compare our model-based approach against two model-free baselines and other methods, as introduced in the main paper. More details are available in the extended version [1].

1) *Map-conditioned Multi-task Navigator* (**MMN**), model-based. Our map-conditioned planner based on MuZero and improved with n-step HER and multi-task training.
2) *Map-conditioned Ape-X HER DQN* (**MAH**), model-free. A reactive baseline based on Ape-X DQN [38] and single-step HER [8], *conditioned* on map and goal.
3) *Single-task Ape-X HER DQN* (**DQN**†), model-free, based on Ape-X DQN [38] and single-step HER [8]. No task context c (map or goal) as input.
4) *Hand-crafted planner*. In the experiments of Section V-D, we create a planner with deterministic path planning and reactive navigation, to study robustness to noise in the abstract map and in localization.
5) *Random*, a reference of the navigation performance.

In the implementation, we keep all components of MMN and MAH the same as much as possible, except that MAH only has a map-conditioned reactive policy network but no hypermodel. We use similar architecture for the Q-value network $Q(s, a, c_\mathcal{T})$, which is also conditioned on abstract 2-D maps and goals, as the policy and value prediction network $g_\theta(s, c_\mathcal{T})$ in our method. The task input to DQN† has been further masked, so the Q-value network is simply a single-task version $Q(s, a)$.

Thus, the main difference between our model-based approach MMN and the multi-task model-free variant MAH is that MAH entangles transfer on the map (dynamics) and goal (reward) levels, since the Q-value network needs to generalize value prediction jointly on different latent states s, goals g, and abstract 2-D maps m.

X. Additional Results

A. Discussion of Other Baseline

In the main paper, we describe a few algorithms we are comparing against with. In this section, we additionally provide an algorithm and its results. It needs different setup, so we include it here.

The [39] (`map-planner`) algorithm uses DQN with HER and sampling transitions from a sub-sample space. It was considered as one of the baselines to compare our model against but it had the following challenges.

We can only train each model on one map, whereas our model can train to do multi-task training on multiple maps. In order to make the map planner generalize on different maps we need to train each model on each of the maps, create a replay buffer for each map for evaluation and then use

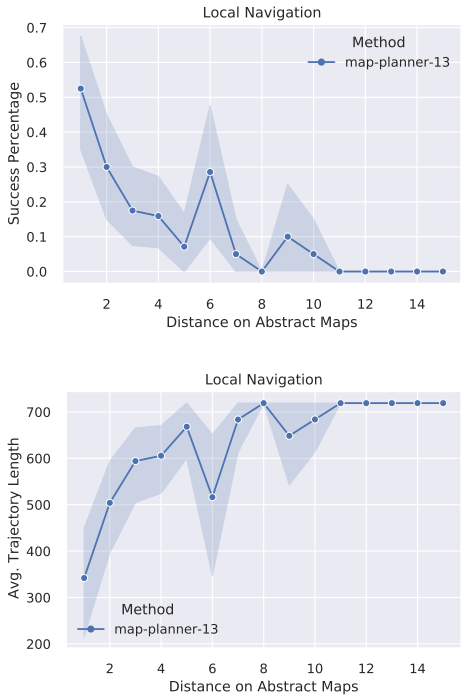
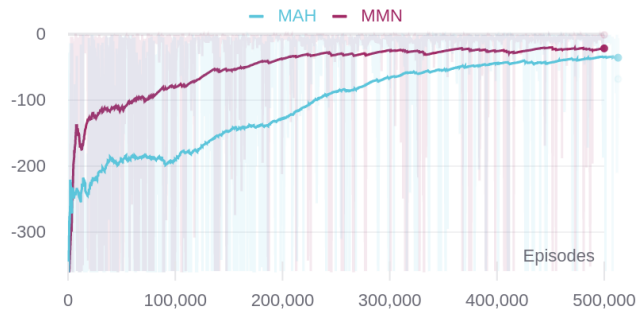
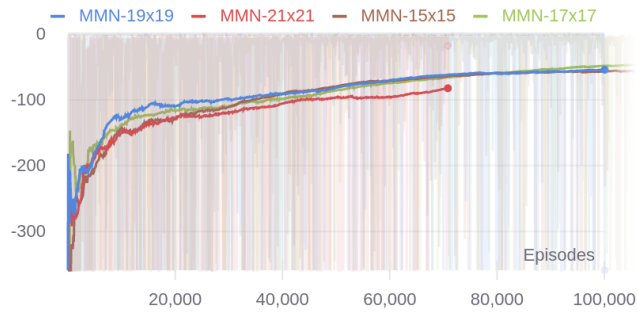


Fig. 9: Map Planner local navigation evaluation performance on 13×13 maps for different distances between start and goal, from 1 to 15. **(Left)** Success Percentage, **(Right)** Average Trajectory Length.



(a) Comparison of MMN and MAH



(b) Comparison on map sizes

Fig. 10: Multi-task training performance of MMN and MAH.

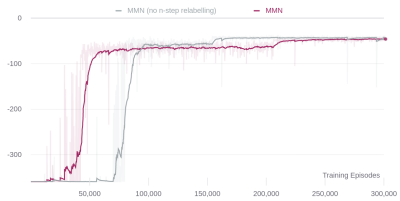


Fig. 11: Ablation study of n-step relabelling.

an ensemble of agents to predict the actions in an unseen environment. Thus, this method is inefficient and hard to generalize on and is not a zero-shot learning paradigm.

In contrast, our method learns a hyper-model that can generalize to novel map environments in zero-shot. This method although can take goal as input, but in a different map, it can only rely on local goal-conditioned value estimation (estimating the distance) to build a graph for each map, and plan on the graph.

B. Local Zero-shot Transfer on Larger Maps

We also evaluated MMN on larger maps from 15×15 to 21×21. We tested the zero-shot transfer performance on 20 unseen maps of corresponding sizes and generated start-goal pairs with distance $[1, 15]$. As shown in Figure 13 (Right), even though training performance is similar among varying sizes (not shown), zero-shot transfer on novel larger maps becomes increasingly harder, which shows the difficulty of learning directly from abstract 2-D maps.

C. Multi-task Training Performance

We demonstrate some representative results of the training performance of MMN and MAH. First, we compare the training on 20 of 13×13 maps with randomly generated goals at each episode, which is the most widely used training setting in our transfer evaluation. In Figure 10 (a), our model-based version MMN is much more sample efficient than the reactive MAH. There are two potential reasons: (1) model-based method is usually more sample efficient demonstrated in many single-task environments, and (2) our MMN is able to share knowledge between local patterns via the hypermodel. We also show the training results on larger map sizes with local start-goal pairs $[1, 5]$ in Figure 10(b). Although we found the evaluation performance decreases on larger maps, the local training performance w.r.t. episodes is similar. We include more results in the supplementary material.

We also study the ablation of n-step relabelling in Figure 11 in a special fixed goal setting on a 13×13 map. With the relabelling, our method is able to get signal earlier and learn faster.

D. Ablation Studies of Transfer Performance

a) Ablation study on evaluation hyperparameters.: We study two related hyperparameters in the zero-shot transfer: (1) number of *simulations*, where we change it to 30 from 100, and (2) *temperature* in action sampling, which is set to 0.1 from original value 0.25. We use one random goal for distance $[1, 15]$ on 20 of 13×13 maps. In Figure 12, we found decreasing number of simulations and increasing deterministicity do not produce significant difference.

b) Transfer with few-shot adaptation.: To examine if zero-shot transfer still have any room of improvement, we

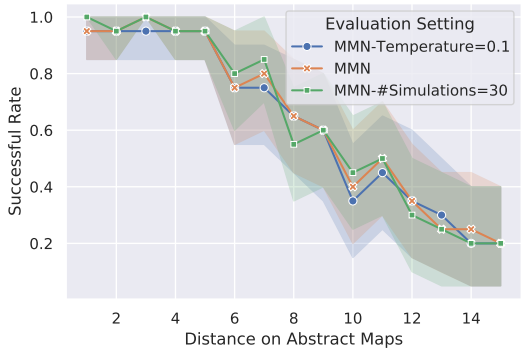


Fig. 12: Ablation study of different hyperparameters in evaluation.

also experiment on finetuning on same maps with different goals. Suprisingly, we found finetuning on novel maps does not result in improved performance. However, it is understandable since we have fully trained the agents during multi-task training, and the amount of data and learning steps in finetuning is insignificant compared to the training stage (about 10^5 vs. 10^3 steps, and 10^5 vs. 10^2 trajectories). We leave the further study of few-shot adaptation with abstract maps for future work.

XI. Additional Studies of Robustness

We provide a complete version of Section V-D, by considering three components in map-based navigation.

To further study the robustness of our method and the importance of each component, we considered breaking three components in closed-loop map-based navigation: Map – (1) $\rightarrow$ Path – (2) $\rightarrow$ Environment – (3) $\rightarrow$ Map (repeat). In general, our learning-based agent is robust to these changes.

To illustrate difficulty of the problem, we considered a **baseline** strategy (hand-crafted deterministic planner) based on perfect information of the environment (e.g. can plan on map) for comparison correspondingly: (1) known perfect maps and intermediate landmarks, (2) scaling factor (unavailable to MMN), and (3) world position on map. Since we assume that it has perfect localization and landmarks, the key step is to reach a landmark given current location, which consists of several procedures: (a) change the orientation to the target landmark, (b) move forward along the direction, and (c) stop at the target cell as soon as possible.

A. Perturbation 1: Planning

For *planning*, we try to break the implicit assumption of requiring perfect abstract map information. We adopt the hierarchical navigation setting, but generating subgoals on perturbed maps, where some proportion of the map's occupancy information is flipped. In Table I, we see that as the perturbation level increases, MMN's performance gradually decreases, but it can still navigate successfully with significant noise levels.

We use the flip strategy to perturb with a probability of 20% on randomly bit-flipping a block (from 0 to 1 or vice versa) and evaluate in the hierarchical navigation setting.

We try two different strategies for applying our algorithm: (1) generating landmarks on perturb map (flip, 20%) of distance 5, (2) a more path following like setting with landmark distance 1. In both settings, we input perturbed maps to the agent.

We show the result in Table I. The local path following (landmark distance 1) is less unstable than longer landmark-level hierarchical navigation. (We use landmark distance 1 to simulate the path following setting.) Both of them show significant degeneration of performance compared to our algorithm with tolerance to the map perturbation. So, our learning-based method is robust to the map topology changes and inaccuracy.

B. Perturbation 2: Localization

For *localizing*, we break the identifiability of agent position (a part of its joint state) by applying random noise. We aim to show that our learned agent does not rely on the position to understand the map, since providing position in the agent world has no relation with localizing on abstract maps and our learning-based method can adapt to the noise. As shown in the Figure 5, MMN had minimal performance drop, compared to a drastic performance drop for the baseline strategy.

We apply additive Gaussian noise (alternative: additive uniform noise) to the position of the joint state, where the mean is 0 and the standard deviation is 1 unit (one block on abstract map, 100×100 in agent's world). We use the same setting and maps as the local transfer experiment (Section 4.2) by comparing the planner on unseen 13×13 maps from distance 1 to 15. This experiment is to study the effects of implicit localization on maps. Results are shown in Figure 14 (Left) and (Middle).

C. Perturbation 3: Action Mapping

For *action mapping*, we break the implicit requirement of known scaling between map and environment. We provide the agent with randomly transformed maps, where the ratio (in both x and y directions) is different. Our experiments in local navigation showed that our agent does not rely on this knowledge or any perfect relation.

Our method also does not depend on the knowledge of the relationship between abstract maps and agent environments. We also try to break this relation, by applying some random (spatial) transformations on the map. Different from manipulating pixel values, we apply spatial transformations, such as perspective transformation. Results are shown in Figure 14 (Right).

D. Perturbation of Top-down Map Input

To further study the importance of the abstract map input, during evaluation on 13×13 maps, we perturb the task context input $c = (m, g)$ and provide an disrupted version to agents. We examine several perturbation strategies: *zero*, *flip*, *shuffle*, and *original* accurate map input. (1) In *zero* mode, we give a tensor of zeros as abstract map and start/goal inputs to agents. In this case, the model may malfunction and rely

on its policy and value prediction function to provide rough estimations. (2) In *shuffle* mode, we sample another map and start/goal pair within the set of evaluation maps. Thus, the entire structure should be entirely different and may largely affect the decision. (3) In *flip* mode, we randomly change the value of a cell with probability $p = 80\%$ (from wall to navigable space or vice versa). Note that flipping only changes the map m but not the goal g, thus the agent can still capture a rough direction in local navigation. We evaluate on distances between 1 and 8.

As shown in Figure 13 (Left), *shuffle* has the largest effect to the performance, since randomly choosing another map not only changes the map input m' to the agent, but also provides a misleading goal g'. Providing a *zero* map/goal has the second largest performance drop, since it does not mislead the agent with wrong map or goal, but does not provide the information that is necessary to complete the task without further learning and exploration. This demonstrates that both MMN and MAH are relying on the abstract map and start/goal input to do zero-shot navigation effectively. Surprisingly, the *flip* strategy turns out to have little performance decay. The reason for this may be that only the map is flipped, and since we evaluate on local start-goal pairs with distance $[1, 8]$, the flipping may not greatly affect the path connecting start-goal pairs, and the agent can rely on the unperturbed goal to navigate in the correct direction for short distances.

XII. Further Ablation Studies of Perturbation Setup

We performed a series of evaluation to test the performance of our model in different scenarios. The scenarios were chosen to test its generalization over unseen maze layouts. The experimentation constitutes two major phases the hierarchical and the local navigation tasks. In each task, to check if the agent is really following the map which is being given as input, we test the agent in perturbed maps and perturbed state space (uncertainty in the world).

The evaluation of the model was performed on 2 settings:

1) Hierarchical setting where the whole map of the task is divided into smaller tasks with sub goals generated to emulate the start and goal pairs of that small problem;
2) Local setting or the whole map.

Majority of the efforts spent on finding and choosing the right hyper-parameter to test the model and vary them to see the behavior of the agent in a given task (map + start and goal pairs).

The following section shows the results obtained through experimentation and shown are some of the optimistic runs of the model for each configuration. The images mainly consists of blue grids depicting the obstacles represented in the 2D occupancy grid, white grid representing the pathways where the agent can traverse, yellow/green the starting point and pink or grey as goals or sub-goals. The blue dots are agent location corresponding to the actual 3D environment and blue lines show the trajectory of the agent.

A. Hierarchical Navigation Evaluation

In hierarchical navigation, the agent must pass through all the landmarks (grey grid) to achieve the optimal path length. The start is always the grid in the top right corner and goal is always the bottom left corner. The landmarks are generated by Dijkstra's algorithm and based on the user's input distance between landmarks. We use the notations below.

- L_d – Distance between Landmarks
- L_T – Trajectory Length
- S_P – Success percentage = $\frac{\text{Number of times the agent reached the goal}}{\text{Total number of times experiment performed over different maps}}$
- Std – Standard deviation of the Gaussian noise (σ)
- M_S – Max number of steps

For consistent representation, we use a specific 13×13 map to visualize all the trajectories.

1) No Perturbation: The Figure 15, shows the performance of the agent when there is no perturbation and for different distance between the sub-goals (but constant throughout a task).

The results suggests that the agent has learned to generalize well for different distance between the sub-goals from its success rate. Another interesting note is that lesser the distance we expected a shorter trajectory but instead L_d = 1,3 has L_T more than when L_d = 5. This could be because the agent could be going faster and might have overshot the nearby landmark and must travel back.

2) Map Perturbation: The Figure 16, shows the performance of the agent when there is a perturbation to the map with a ratio of 0.2 and for different distance between the sub-goals (but constant throughout a task).

The violin plot 17, suggests that more the perturbation ratio, longer the trajectory lengths are. This also confirms the fact that agent is indeed dependent on the map input and the learning is utilized well. If there were some issues with this relationship, it could hint that agent is not utilizing the map properly. In that case, we might need to debug the training strategy.

3) Localization Perturbation: The Figure 18, shows the performance of the agent when there is perturbation to the state vector of the agent in the actual 3D environment and for different distance between the sub-goals.

The violin plot 19, is quite interesting to look at. The pattern is aligning with what we expected, which is more the noise longer the trajectory length and lesser the SPL. The images show the expected behavior in a particular 13×13 map.

4) Max Steps: Here, the L_d = 5, is kept as constant for effective comparison of this hyper-parameter.

The violin plot 21, suggests max steps of 25 and 50 are good bets for evaluation. The standard deviation of step 25 is a good sign of consistency. We can try to incorporate 2 versions in training, currently it is 50 max steps.

B. Local Navigation Evaluation

In Local navigation, the agent must have to reach the goal from start in an optimal way. The start and goal are randomly generated. There are no landmarks.

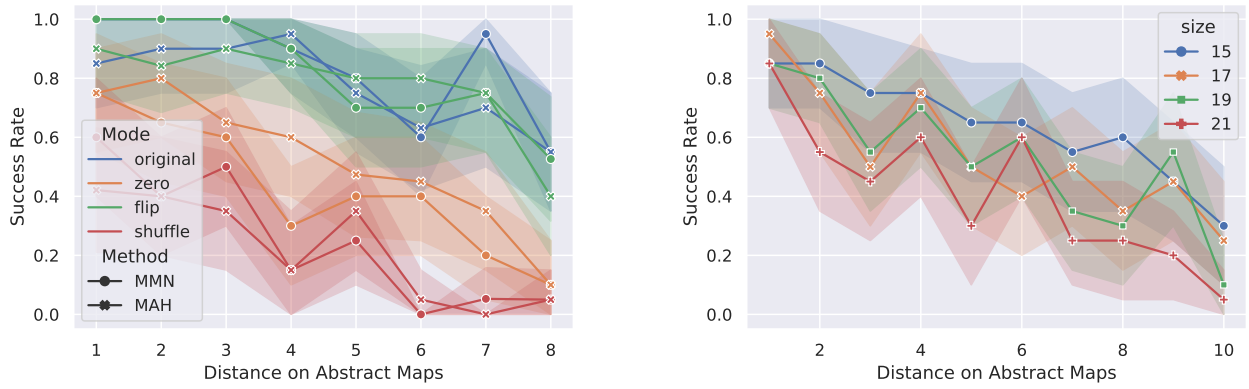


Fig. 13: **(Left)** Study of performance on different perturbation strategies. **(Right)** Performance on larger maps.

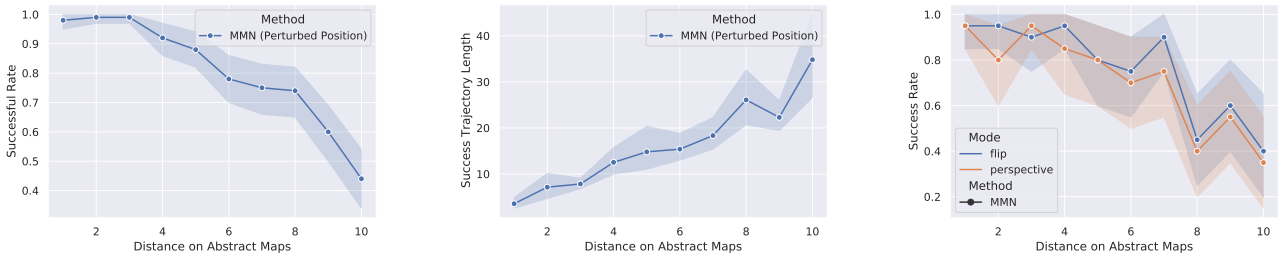


Fig. 14: **(Left)** Success rate and **(Middle)** successful trajectory length on local navigation with noisy position information. **(Right)** Success rate on local navigation with perturbed action mapping.

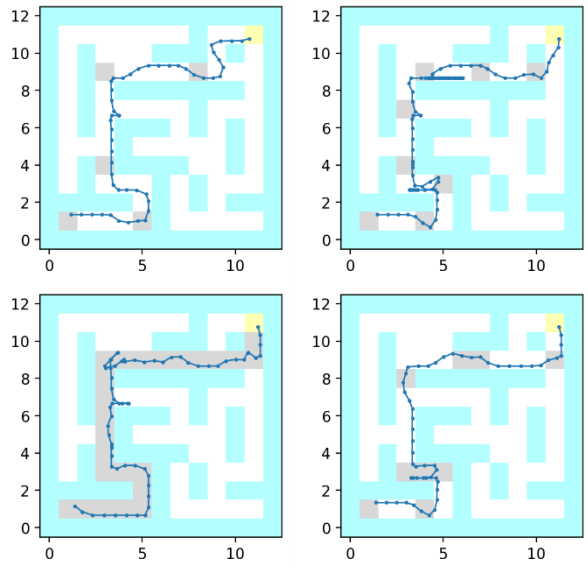


Fig. 15: Hierarchical No Perturbation

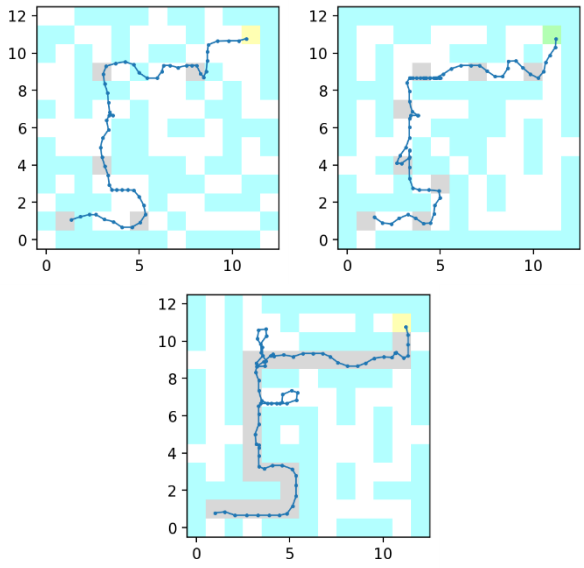


Fig. 16: Hierarchical Map Perturbation

- L_d – Distance between Start and Goal
- S_P – Success percentage $= \frac{\text{Number of times the agent reached the goal}}{\text{Total number of times experiment performed over different maps}}$
- Std – Standard deviation of the Gaussian noise (σ)

For consistent representation, we use a specific 13×13 map to visualize all the trajectories.

1) No Perturbation: The Figure 22, shows the performance of the agent when there is no perturbation and for different distance between the start and goal.

The above images confirm our theory that even though the

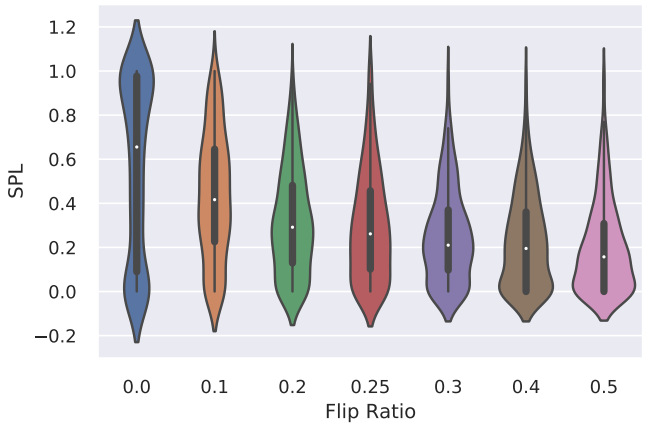


Fig. 17: Violin Plot of SPL for different perturbation ratios. The figure clearly explains the impact of map perturbation ratio on the SPL metric. More the noise lesser the SPL gets.

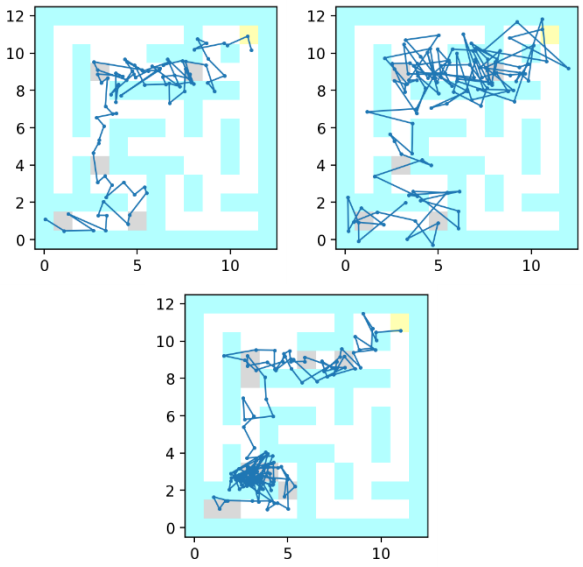


Fig. 18: Hierarchical Local Perturbation

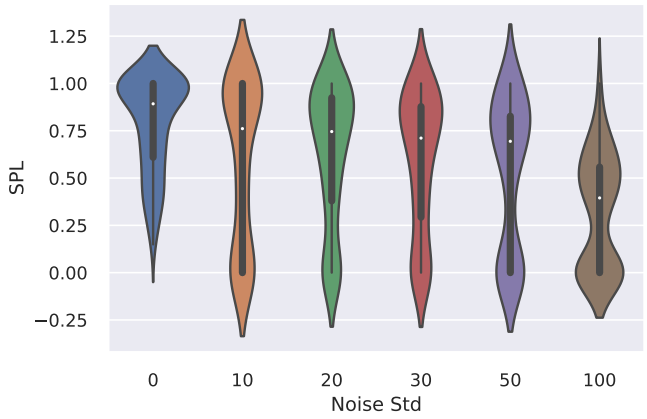


Fig. 19: Violin Plot of SPL for different perturbation standard deviation of noise. The figure clearly explains the impact of perturbation of the state vector on the SPL. More the noise lesser the SPL gets.

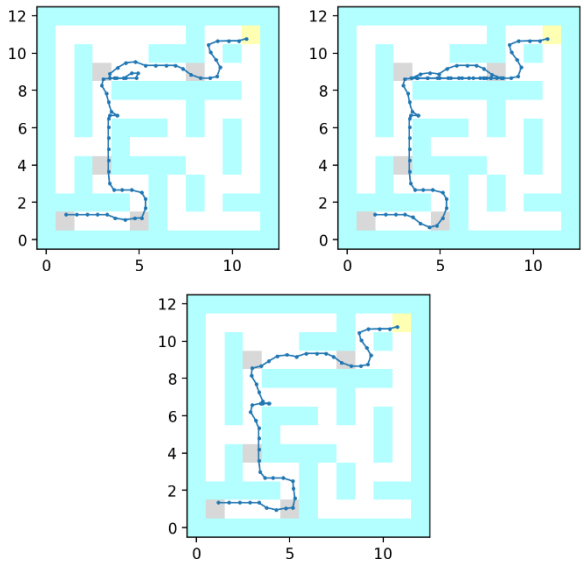


Fig. 20: Different Max Steps

agent has learnt to generalize well for different distances of start-goal pairs, for distances larger than the one seen during training it might struggle. That is why the success rate in L_d = 8 is lesser than the others.

2) Localization Perturbation: The Figure 23, shows the performance of the agent when there is perturbation to the state vector of the agent in the actual 3D environment and for different distance between the sub-goals.

The Figure 23, goes well with the hypothesis that more the noise to the state vector longer the trajectory length and the agent struggles to navigate freely.

C. Deterministic Planner

This deterministic planner is a simple agent that follows the landmarks in a straight path at a slow pace (velocity in 3D environment). This agent can serve as an intermediate benchmark for our models and still can be optimized using motion planning algorithms with Kino-dynamic constraints to obtain an optimal trajectory for SPL metric calculation.

The Figure 24, we were experimenting with different step sizes to see on which case the agent takes the shortest (optimal) path to the goal. Here step size is the rate at which the agent moves in the 3D environment. The smaller the step size slower it is travelling and larger the step size faster it travels. We found step size 3 seems to produce optimal results hence used it for the rest of analysis. The Figure 25, deals with same step size but different distance between the landmarks. This agent also struggles to navigate when distances are large (> 5).

1) Map Perturbation: The Figure 26, suggests that the agent does not care about the abstract map. This is because, the agent interacts with the deep mind lab environment directly and with the inputs from the environment itself (like interacting with the OpenAI Gym environment). Hence,

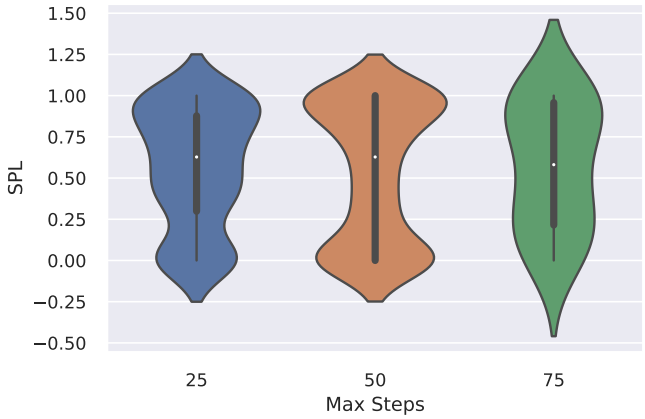


Fig. 21: Violin Plot of SPL for different max-steps to reach the sub-goal. The figure clearly explains the impact of max-steps on the SPL. The optimal max-steps is 50 which produces more SPL.

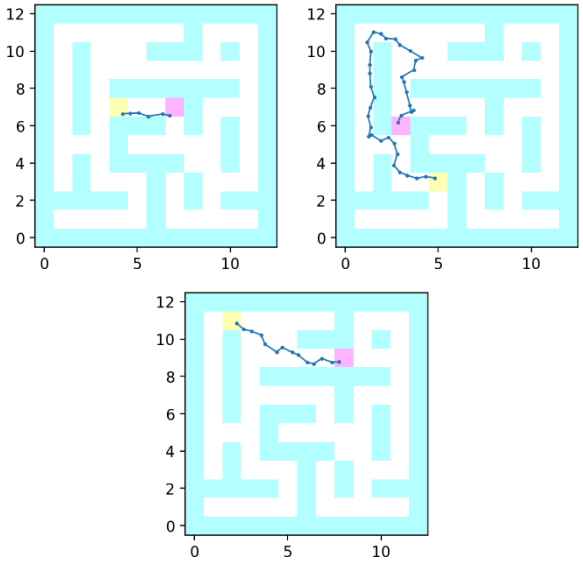


Fig. 22: Local No Perturbation

perturbation really will not affect it. Hence, we might need a deterministic planner that interacts with the abstract 2D map for a more realistic comparison.

2) Localization Perturbation: The figures 27 and 28, suggests that the agent is severely affected by the noise added to the state vector. This is again because the agent is directly interacting with the 3D environment, because of which it is struggling a lot when the only perception is perturbed.

D. Notable Behavior

The above image is an interesting behavior observed in the hierarchical navigation with no perturbation. The agent seems to work fine by passing through all the landmarks when the landmarks are scattered in the 'L' shaped pattern. Otherwise, the agent misses few landmarks but eventually reaches the end goal. This could be because of the training phase when the distance between landmarks are closer (≤ 5) and most of the times they would be in a straight line.

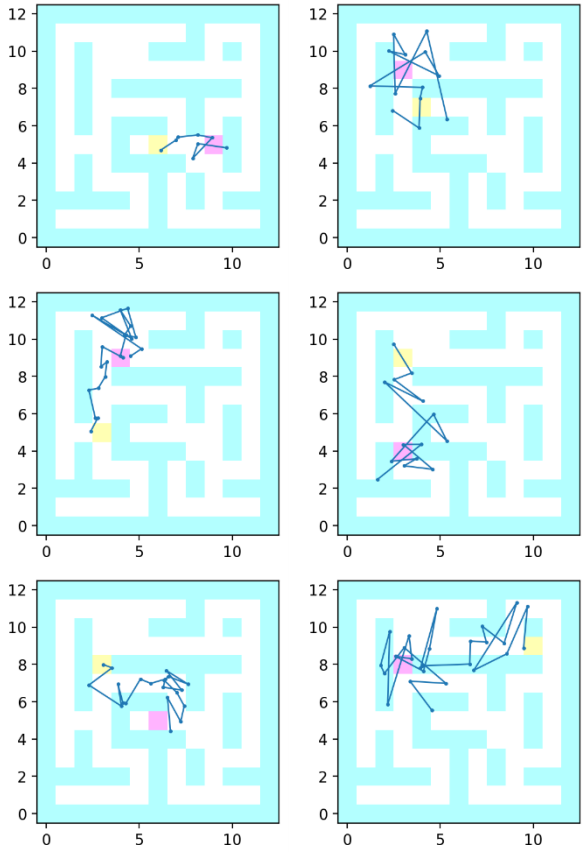


Fig. 23: Localization Perturbation - Std = 100

Maybe having more complex paths and longer distances during training phase might alleviate this behavior.

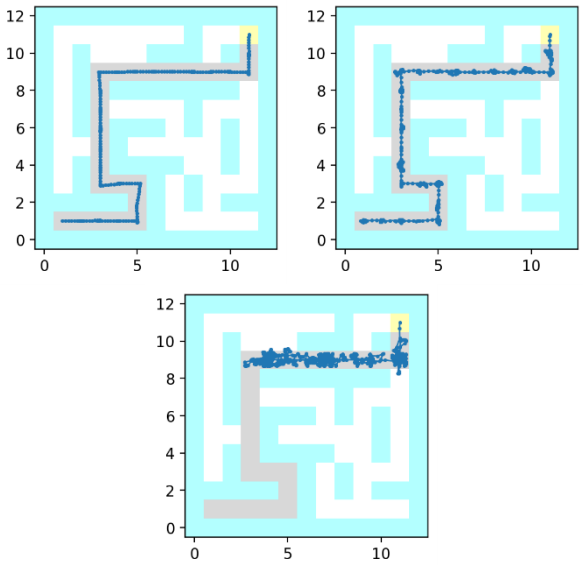


Fig. 24: Deterministic Planner - No Perturbation - Step Size Tuning

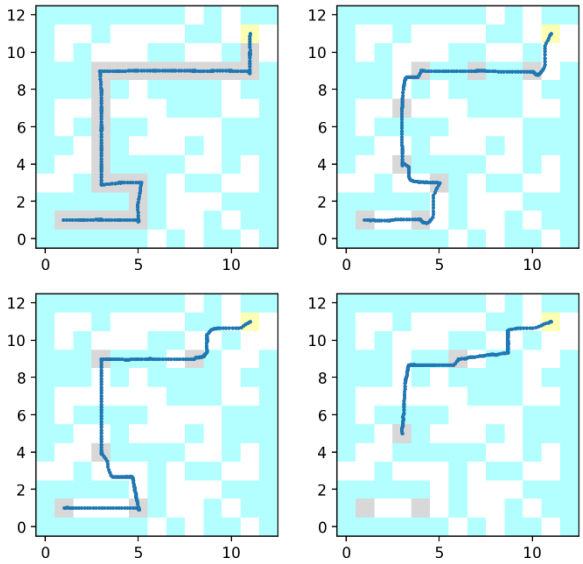


Fig. 26: Deterministic Planner - Map Perturbation

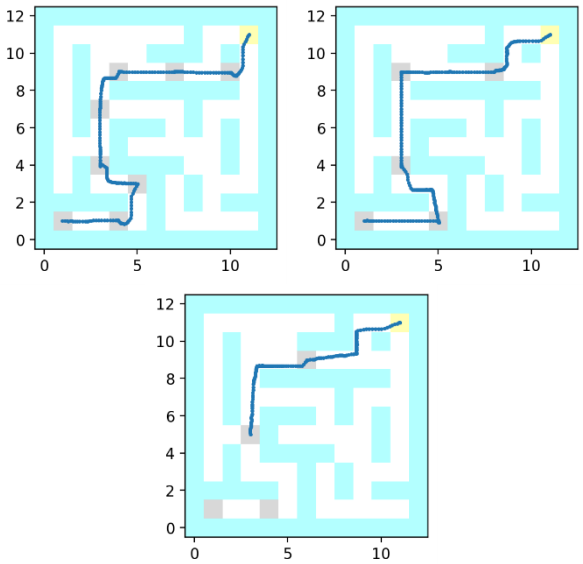


Fig. 25: Deterministic Planner - No Perturbation - Different L_d

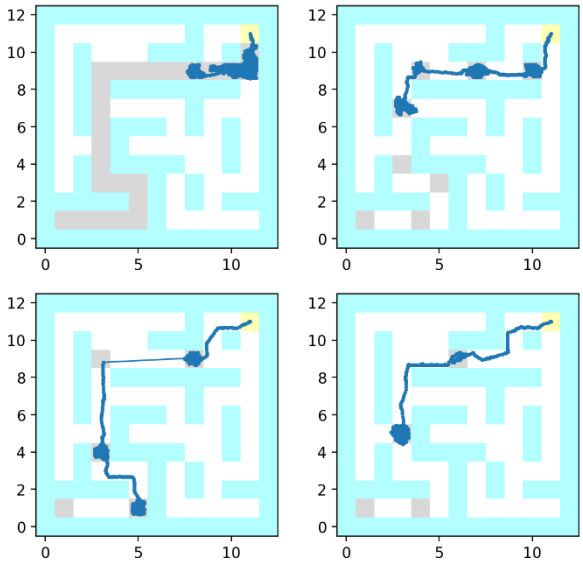


Fig. 27: Deterministic Planner - Localization Perturbation - Std = 50

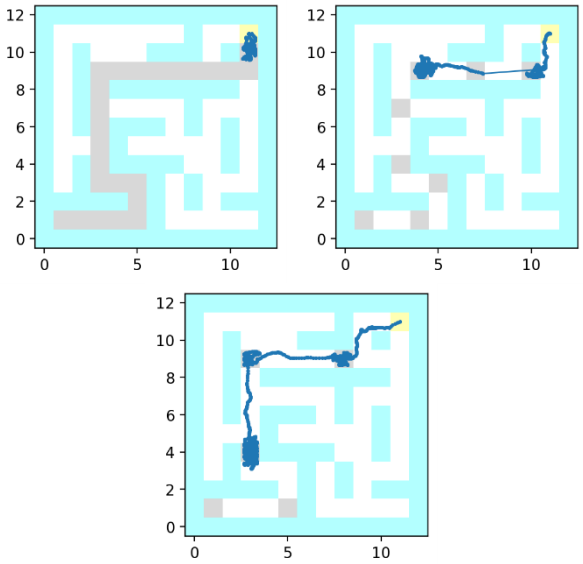


Fig. 28: Deterministic Planner - Localization Perturbation - Std = 100

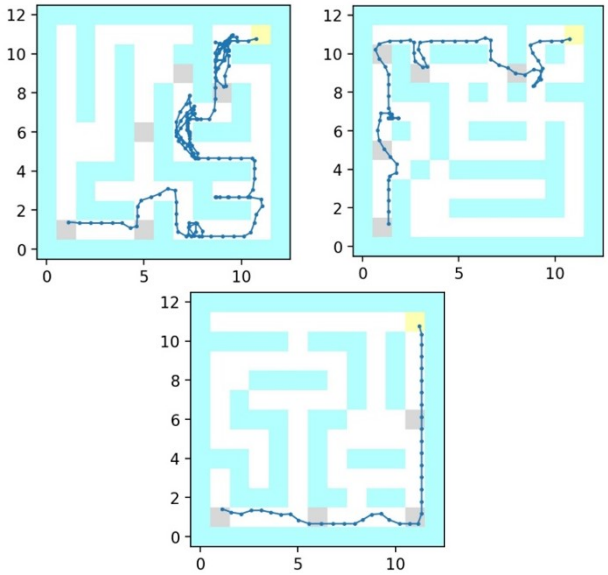


Fig. 29: Notable Behavior