# LEARNING WHEN AND WHAT TO ASK: A HIERARCHICAL REINFORCEMENT LEARNING FRAMEWORK

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Reliable AI agents should be mindful of the limits of their knowledge and consult humans when sensing that they do not have sufficient knowledge to make sound decisions. We formulate a hierarchical reinforcement learning framework for learning to decide *when* to request additional information from humans and *what* type of information would be helpful to request. Our framework extends partially-observed Markov decision processes (POMDPs) by allowing an agent to interact with an assistant to leverage their knowledge in accomplishing tasks. Results on a simulated human-assisted navigation problem demonstrate the effectiveness of our framework: aided with an interaction policy learned by our method, a navigation policy achieves up to a $7\times$ improvement in task success rate compared to performing tasks only by itself. The interaction policy is also efficient: on average, only a quarter of all actions taken during a task execution are requests for information. We analyze benefits and challenges of learning with a hierarchical policy structure and suggest directions for future work.

## 1 INTRODUCTION

To prevent catastrophic failures, AI agents should be taught to determine when they cannot make reliable decisions and to seek for advice from humans in those situations. In such cases, human assistants can potentially be more helpful (with lower effort) if the agent communicates specifically *what* knowledge it needs for making better decisions. For example, in an indoor object-finding task (Figure 1), if a robot could inform its human user that all it requires to accomplish the task is knowing what room the target object is in, the human user would concisely give the robot a room label (e.g,"*it is in the bedroom*"), instead of having to compose a verbose navigation instruction (e.g., "*go to the door on your left, turn right...*"). This would also lead to improved decision making on the part of the agent, because it receives exactly the information it needs and does not have to cope with redundant or misleading information.

One way agents can learn to express themselves to humans is to learn to mimic the sorts of questions humans might ask in similar circumstances, as has been explored in dialogue systems (Rao & Daumé III, 2018; Das et al., 2017; Li et al., 2016) or generating explanations (Camburu et al., 2018; Chen et al., 2021; Narang et al., 2020; Kim et al., 2018). A limitation of this approach is that the agent only mirrors human behavior, without forming an understanding of its own limitations and needs; essentially it learns to convey what a *human* may be concerned about, not what *it* is concerned about (Camburu et al., 2019). In this paper, we take a step towards designing agents that can ask questions that faithfully convey their intrinsic needs and uncertainties. We develop a hierarchical decision-making framework where an agent is equipped with specific, innate information-seeking intents that can be easily translated into human-intelligible requests for different types of information. Our framework extends standard partially-observed Markov decision processes (POMDPs), by introducing a human assistant into this framework, formulating how the agent can request additional information from the assistant to better accomplish tasks.

Under our framework, three natural information-seeking intents arise: in every step, the agent can ask for more information about (i) its current state, (ii) the goal state, or (iii) a subgoal state which, if reached, helps it make progress on the current task. Through interacting with the assistant who can answer such questions, and with the environment, receiving costs or rewards, the agent learns to decide when to convey an intent to the assistant and which intent to convey. As the intents are

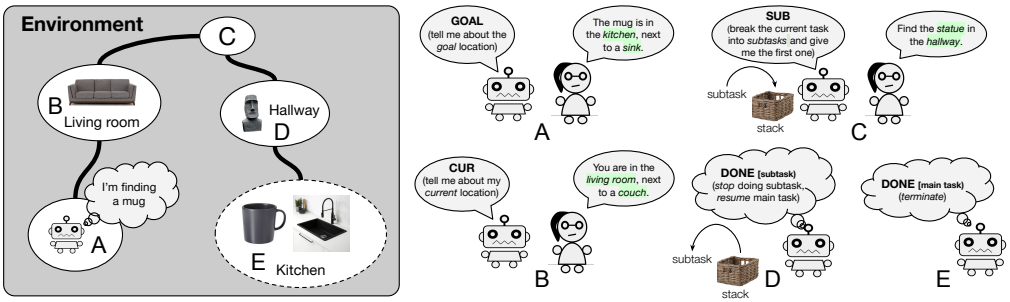



Figure 1: An illustration of our framework in a human-assisted navigation problem. An agent can only observe part of the environment and is asked to find a mug in the kitchen. An assistant communicates with the agent and can provide it with information about the environment and the task. Initially (A)it may request more information about the goal, but may not know enough about where it currently is. For example, at location B, due to limited perception, it does not recognize that it is in a living room and stands next to a couch. It can obtain for such information from the assistant. If the current task becomes too difficult (like at location C), the agent can require the assistant to provide a simpler subtask which, if accomplished, helps it amke progress on the main task. The agent maintains a stack of tasks and always executes the task at the top. When the agent receives a (sub)task, it pushes the (sub)task to the top of the stack. When it wants to stop executing a (sub)task, it pops the (sub)task from the stack (e.g., at location D). At location E, the agent empties the stack and terminates its execution.

integrated into the agent's learning and decision-making process, the selected intent in every step reflects what type of information the agent intrinsically finds most useful to obtain at that time. Our framework employs a stack data structure for managing goals, allowing the agent to conveniently construct deeper goal hierarchies, whereas most hierarchical decision-making frameworks (Le et al., 2018; Kulkarni et al., 2016; Sutton et al., 1999) work with only two levels of goal abstractions.

We demonstrate the effectiveness of our framework on a simulated human-assisted navigation problem. In this problem, we learn an interaction policy that controls how the agent communicates with an assistant and navigates in an environment. Equipped with the interaction policy learned by our method, the agent achieves a substantial $7\times$ increase in success rate on tasks that take place in environments it has not previously seen, versus performing those tasks without assistance. The agent achieves this level of performance while issuing only $4.8$ requests to the assistant on average, representing less than ¼ of the total number of actions taken in a task execution. The learned interaction policy exhibits reasonable behaviors like requesting goal clarification only once right at the beginning or requesting subgoals more often on more difficult tasks. We show that performance of the policy in unseen environments can be further improved by recursively asking for subgoals of subgoals. Finally, we discuss limitations of the policy model and feature representation, which suggest room for future improvements.

## 2 Motivation: Limitations of the Standard POMDP Framework

We consider an environment defined by a partially-observed Markov decision process (POMDP) $E = (\mathcal{S}, \mathcal{A}, T, c, \mathcal{D}, \rho)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, state-transition function $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, cost function $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, description space $\mathcal{D}$, and description function $\rho : \mathcal{S} \to \Delta(\mathcal{D})$.[1] Here, $\Delta(\mathcal{Y})$ denotes the set of all probability distributions over a set $\mathcal{Y}$. We refer to this environment as the *operation environment* because it is where the agent operates to accomplish tasks.

Each *task* in the environment is a tuple $(s_1, g_1, d_1^g)$ where $s_1$ is the start state, $g_1$ is the goal state, and $d_1^g$ is a limited description of $g_1$. Initially, a task $(s_1, g_1, d_1^g)$ is sampled from a task distribution $\mathfrak{T}$. An agent starts in $s_1$ and is only given $d_1^g$. It has to reach $g_1$ during within $H$ time steps. Let

[1] We use the term "description" in lieu of "observation" in the POMDP formulation to emphasize two properties of the information the agent has access to for making decisions: (i) the information can be in various modalities and (ii) the information can be obtained via not only perception, but also communication.

$g_t$ and $d_t^g$ be the goal and goal description being executed at time step $t$, respectively. In a standard POMDP, $g_t = g_1$ and $d_t^g = d_1^g$ for $1 \leq t \leq H$. But later, we will enable the agent to set new goals via communication with humans.

At any time step $t$, the agent does not know its true state $s_t$ but only receives a *description* $d_t^s \sim \rho(s_t)$ of the state. Generally, the description can include any information coming from any knowledge source (e.g., an RGB image and/or a verbal description describing the current view). Given $d_t^s$ and $d_t^g$, the agent then makes a decision $a_t \in \mathcal{A}$, transitions to the next state $s_{t+1} \sim T(s_t, a_t)$, and receives a cost $c_t \triangleq c(s_t, a_t)$. A special action $a_{\text{done}} \in \mathcal{A}$ is taken when the agent decides to terminate its execution. The goal of the agent is to reach $g_1$ with minimum cumulative cost $C(\tau) = \sum_{t=1}^{H} c_t$, where $\tau = (s_1, d_1^s, a_1, s_2, d_2^s, a_2, \ldots, s_H, d_H^s)$ is an execution of the task.

As the agent does not have access to its true state, it can only make decisions based on the (observable) partial execution $\tau_{1:t} = (d_1^s, a_1, \ldots, d_t^s)$. Kaelbling et al. (1998) introduce the notion of a *belief state* $b \in \Delta(\mathcal{S})$, which sufficiently summarizes a partial execution as a distribution over the state space $\mathcal{S}$. In practice, when $\mathcal{S}$ is continuous or high-dimensional, representing and updating a full belief state (whose dimension is $|\mathcal{S}|$) is intractable. We follow recent work (Hausknecht & Stone, 2015), using recurrent neural networks to learn compact representation of partial executions. We denote by $b_t^s$ a representation of the partial execution $\tau_{1:t}$ and by $\mathcal{B}$ the set of all possible representations.

The agent maintains an *operation policy* $\hat{\pi} : \mathcal{B} \times \mathcal{D} \to \Delta(\mathcal{A})$ that maps a belief state $b^s$ and a goal description $d^g$ to a distribution over $\mathcal{A}$. The learning objective for solving a standard POMDP is to estimate an operation policy that minimizes the expected cumulative cost of performing tasks:

$$\min_{\pi} \mathbb{E}_{(s_1, g_1, d_1^g) \sim \mathfrak{T}, \tau \sim P_\pi(\cdot | s_1, d_1^g)} [C(\tau)] \tag{1}$$

where $P_\pi(\cdot \mid s_1, d_1^g)$ is the distribution over executions generated by a policy $\pi$ given start state $s_1$ and goal description $d_1^g$. In a standard POMDP, an agent performs tasks by executing its own operation policy without asking for any external assistance. Moreover, the description function $\rho$ and the goal description $d_1^g$ are assumed to be fixed during a task execution. As seen from Equation 1, given a fixed environment and task distribution, the expected performance of the agent is solely determined by the operation policy $\hat{\pi}$. Thus, the standard POMDP framework does not provide any mechanism for improving the agent's performance other than enhancing the operation policy.

## 3 Leveraging Human Knowledge to Better Accomplish Tasks

We introduce an *assistant* into the operation environment, who can provide information the environment's states. We consider a setting where the agent already possesses a decent-but-imperfect *pre-learned* operation policy $\hat{\pi}$. Our interest is to teach the agent to leverage the human's knowledge to make better decisions *without modifying the operation policy*.[2] Instead, we focus on learning an *interaction policy* $\psi_\theta$ (parametrized by $\theta$) that controls how the agent communicates with the human to gather additional information. The operation policy will be invoked by the interaction policy if the latter decides that the agent does not need new information and wants to take an operating action.

Let us first give an intuition for why obtaining additional information from the human can improve performance of the agent in our framework. Our key insight is that we can enhance the operation policy (without updating it) by manipulating its input. In most cases of interest, the performance of the pre-trained operation policy $\hat{\pi}$ over its input space is non-uniform: there are inputs $(b^s, d^g)$ on which the agent makes more accurate decisions than others. For example, in an object-finding navigation task, during pre-training, a human trainer may walk a robot around a house and describe the locations they walk by. Alternatively, the robot can be trained in a simulated environment with dense visual semantic annotations. The robot may learn to perform well in such rich-information conditions. When being deployed in a new environment, however, it may perform poorly due limited perception (e.g., it cannot recognize new object types). However, its capability of performing tasks under rich information remains. It would still find a spoon very quickly if it knew that the current room was a kitchen and there was cupboard next to it. The agent may obtain richer information by asking a human to provide new state descriptions.

[2]Even though updating the operation policy itself using this human knowledge is an another interesting research direction to explore, we leave it for future work.

**Communication with the Assistant.** The assistant is present all the time and knows the agent's current state $s_t$ and the goal state $g_t$. It is specified by two functions: a description function $\rho_A : \mathcal{S} \times \mathcal{D} \to \Delta(\mathcal{D})$ and a subgoal function $\sigma_A : \mathcal{S} \times \mathcal{S} \to \Delta(\mathcal{S})$. $\rho_A(d' \mid s, d)$ specifies the probability of giving $d'$ as the new description of state $s$ given a current description $d$. $\omega_A(g' \mid s, g)$ indicates the probability of proposing $g'$ as a subgoal given a current state $s$ and a goal state $g$.

At time step $t$, the assistant accepts three types of request from the agent:

(a) CUR: requests a new description of $s_t$ and receives $d^s_{t+1} \sim \rho_A(\cdot \mid s_t, d^s_t)$;
(b) GOAL: requests a new description of $g_t$ and receives $d^g_{t+1} \sim \rho_A(\cdot \mid g_t, d^g_t)$;
(c) SUB: requests a description of a subgoal $g_{t+1}$ and receives $d^g_{t+1} \sim \rho_A(\cdot \mid g_{t+1}, \emptyset)$ where $g_{t+1} \sim \omega_A(\cdot \mid s_t, g_t)$ and $\emptyset$ is an empty description.

**Interaction Policy.** The action space of the interaction policy $\psi_\theta$ consists of five actions: $\{$CUR, GOAL, SUB, DO, DONE$\}$. The first three actions correspond to making the three types of request that the assistants accepts. The remaining two actions are used to traverse in the operation environment

(d) DO: executes the action $a^{\text{do}}_t \triangleq \arg\max_{a \in \mathcal{A}} \hat{\pi}(a \mid b^s_t, d^g_t)$. The agent transitions to a new operation state $s_{t+1} \sim T(s_t, a^{\text{do}}_t)$;
(e) DONE: determines that the current goal $g_t$ has been reached.[3] If $g_t$ is a main goal ($g_t = g_1$), the task episode ends. If $g_t$ is a subgoal ($g_t \neq g_1$), the agent may choose a new goal to follow. Our problem formulation leaves it open on what goal should be selected next.

By selecting among these actions, the interaction policy essentially decides *when* to ask the assistant for additional information, and *what* types of information to ask for. Our formulation does not specify the input space of the interaction policy, as this space depends on how the agent implements its goal memory (i.e. how it stores and retrieves the subgoals). In the next section, we introduce an instantiation where the agent uses a stack data structure to manage (sub)goals.

## 4 HIERARCHICAL REINFORCEMENT LEARNING FRAMEWORK

In this section, we first formulate the POMDP environment that this policy acts in, referred to as the *interaction environment* (§4.1). Our construction employs a *goal stack* to manage multiple levels of (sub)goals (§4.2). A goal stack stores all the goal descriptions the agent has received but has not yet terminated (by choosing the DONE action). It is updated in every step depending the action taken. We design a cost function (§4.4) that specifies a trade-off between the cost of taking actions (acting cost) and the cost of not completing a task (task error). This cost function is a part of our solution, not a part of the framework formulation, and can be customized for specific applications.

### 4.1 INTERACTION ENVIRONMENT

Given an operation environment $E = (\mathcal{S}, \mathcal{A}, T, c, \mathcal{D}, \rho)$, the interaction environment constructed on top of $E$ is a POMDP $\bar{E} = (\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{T}}, \bar{c}, \bar{\mathcal{D}}, \bar{\rho})$ with:

- State space $\bar{\mathcal{S}} = \mathcal{S} \times \mathcal{D} \times \mathcal{G}_L$ where $\mathcal{G}_L$ is the set of all goal stacks containing at most $L$ elements ($L$ is a hyperparameter). Each state $\bar{s} = (s, d^s, G) \in \bar{\mathcal{S}}$ is a tuple of an operation state $s \in \mathcal{S}$, its description $d^s \in \mathcal{D}$, and a goal stack $G \in \mathcal{G}_L$. Each element in the goal stack $G$ is a tuple $(g, d^g)$ of a goal state $g \in \mathcal{S}$ and its description $d^g \in \mathcal{D}$;
- Action space $\bar{\mathcal{A}} = \{$CUR, GOAL, SUB, DO, DONE$\}$;
- State-transition function $\bar{T} = T_s \cdot T_G$ where $T_s : \mathcal{S} \times \mathcal{D} \times \bar{\mathcal{A}} \to \Delta(\mathcal{S} \times \mathcal{D})$ and $T_G : \mathcal{G}_L \times \bar{\mathcal{A}} \to \Delta(\mathcal{G}_L)$;
- Cost function $\bar{c} : (\mathcal{S} \times \mathcal{G}_L) \times \bar{\mathcal{A}} \to \mathbb{R}$ (defined in § 4.4 to trade off acting cost and task error);
- Description space $\bar{\mathcal{D}} = \mathcal{D} \times \mathcal{G}^d_L$ where $\mathcal{G}^d_L$ is the set of all goal-description stacks of size at most $L$. The agent cannot access the environment's goal stack $G$, which contains true goal states. Instead, it can only observe the descriptions in $G$. We call this partial stack a *goal-description stack*, denoted by $G^d$;

[3] Note that the agent may falsely decide that a goal has been reached.

- Description function $\bar{\rho} : \bar{\mathcal{S}} \rightarrow \bar{\mathcal{D}}$, where $\bar{\rho}(\bar{s}) = \bar{\rho}(s, d^s, G) = (d^s, G^d)$. Unlike in the standard POMDP formulation, this description function is deterministic.

A belief state $\bar{b}_t$ summarizes a partial execution $(\bar{s}_1, \bar{a}_1, \cdots, \bar{s}_t)$. We formally define the interaction policy as $\psi_\theta : \bar{\mathcal{B}} \rightarrow \Delta(\mathcal{A})$, where $\bar{\mathcal{B}}$ is the set of all interaction belief states.

### 4.2 Goal Stack

A goal stack is an ordered set of tasks that the agent has not declared completion (by calling Done). The initial stack $G_1 = \{(g_1, d_1^g)\}$ contains the main goal $g_1$, and its description $d_1^g$. Let $G_t$ be the goal stack at time step $t$. Only the Goal, Sub, Done actions alter the stack. The Goal action replaces the top goal description with $d_{t+1}^g$, the new description given by the assistant. The Sub action pushes a new subtask $(g_{t+1}, d_{t+1}^g)$ to the stack. The Done action pops the top element.

$$G_t.\texttt{update}(a) = \begin{cases} G_t.\texttt{pop}().\texttt{push}(g_t, d_{t+1}^g) & \text{if } a = \textsc{Goal}, \\ G_t.\texttt{push}(g_{t+1}, d_{t+1}^g) & \text{if } a = \textsc{Sub}, \\ G_t.\texttt{pop}() & \text{if } a = \textsc{Done}, \\ G_t & \text{otherwise} \end{cases} \tag{2}$$

The Sub action is not available to the agent when the current stack contains $L$ elements, guaranteeing that goal stack always has at most $L$ elements. The goal-stack transition function $T_G$ is defined as $T_G(G_{t+1} \mid G_t, \bar{a}_t) = \mathbb{1}\{G_{t+1} = G_t.\texttt{update}(\bar{a}_t)\}$ where $\mathbb{1}\{.\}$ is an indicator function.

### 4.3 Transition of the Current Operation State and Its Description

To complete the definition of the state-transition function $\bar{T}$, we define the transition function $T_s$. This function is factored into two terms by the chain rule:

$$T_s(s_{t+1}, d_{t+1}^s \mid s_t, d_t^s, \bar{a}_t) = P(s_{t+1} \mid s_t, \bar{a}_t) \cdot P(d_{t+1}^s \mid s_{t+1}, d_t^s, \bar{a}_t) \tag{3}$$

Only taking the Do action may change the current operation state

$$P(s_{t+1} \mid s_t, \bar{a}_t) = \begin{cases} T\left(s_{t+1} \mid s_t, a_t^{\text{do}}\right) & \text{if } \bar{a}_t = \textsc{Do}, \\ \mathbb{1}\{s_{t+1} = s_t\} & \text{otherwise} \end{cases} \tag{4}$$

The description $d_t^s$ may vary when the agent moves to a new operation state (by taking the Do action) or requests a new description of $s_t$ (by taking the Cur action)

$$P(d_{t+1}^s \mid d_t^s, s_{t+1}, \bar{a}_t) = \begin{cases} \rho(d_{t+1}^s \mid s_{t+1}) & \text{if } \bar{a}_t = \textsc{Do}, \\ \rho_A(d_{t+1}^s \mid s_{t+1}, d_t^s) & \text{if } \bar{a}_t = \textsc{Cur}, \\ \mathbb{1}\{d_{t+1}^s = d_t^s\} & \text{otherwise} \end{cases} \tag{5}$$

### 4.4 Cost Function

The interaction policy needs to balance between two types of cost: the cost of taking actions (acting cost) and the cost of not completing a task (task error). The acting cost also subsumes the cost of communicating with the assistant because, in reality, such interactions consume time, human effort, and possibly trust. Assuming that the assistant is helpful, acting cost and task error usually conflict with each other; for example, the agent may lower its task error if it is willing to suffer a larger acting cost by increasing the number of requests to the assistant.

We employ a simplified model where all types of cost are non-negative real numbers of the same unit. Making a request of type $a$ is assigned a constant cost $\gamma_a$. The cost of taking the Do action is $c(s_t, a_t^{\text{do}})$, the cost of executing the $a_t^{\text{do}}$ action in the operation environment. Calling Done to terminate execution of the main goal $g_1$ incurs a task error $c(s_t, a_{\text{done}})$. We exclude the task errors of executing subgoals because the interaction policy is only evaluated on reaching the main goal

$$\bar{c}(s_t, G_t, \bar{a}_t) = \begin{cases} c(s_t, a_t^{\text{do}}) & \text{if } \bar{a}_t = \textsc{Do} \\ \gamma_{\bar{a}_t} & \text{if } \bar{a}_t \in \{\textsc{Cur}, \textsc{Goal}, \textsc{Sub}\}, \\ c(s_t, a_{\text{done}}) & \text{if } \bar{a}_t = \textsc{Done}, |G_t| = 1 \\ 0 & \text{if } \bar{a}_t = \textsc{Done}, |G_t| > 1, \end{cases} \tag{6}$$

The magnitudes of the costs naturally specify a trade-off between operation cost and task error. For example, setting the task errors much larger than other costs indicates that completing tasks is prioritized over taking few actions.

## 5 LEARNING WHEN AND WHAT TO ASK IN HUMAN-ASSISTED NAVIGATION

**Problem.** We apply our framework to modeling a human-assisted navigation (HAN) problem. In HAN, a human requests an agent to find an object in an indoor environment. Each task request asks the agent to go to a room of type $r$ and find an object of type $o$ (e.g., find a spoon in a kitchen). The agent is equipped with a camera, which the human can also see. We assume that the human is sufficiently familiar with the environment that they can recognize the agent's location by looking at its current view. While an agent is performing a task, it may request the human to provide additional information via telecommunication (e.g., a chat app). Specifically, it can ask for a description of the current location (CUR), the goal location (GOAL), or a subgoal location that is on the path from the current location to the goal location (SUB).[4]

Before issuing a task request, the human imagines a goal location (but do not reveal it to the agent). We are primarily interested in evaluating success in *goal-finding*, i.e. whether the agent can arrive at the human's intended goal location. Even though there could be multiple locations that match a request, the agent only succeeds if it arrives exactly at the chosen goal location. We also determine success in *request-fulfilling*, where the agent successfully fulfills a request if it navigates to any node that is within two meters of an object that matches the request.

**Operation Environment.** We construct the operation environments using the environment graphs provided by the Matterport3D simulator (Anderson et al., 2018). Each environment graph is generated from a 3D model of a house where each node is a location in the house and each edge connects two nearby unobstructed locations. Each operation state $s$ corresponds to a node in the graph. At any time, the agent's operation action space $\mathcal{A}$ consists of traversing to any of the nodes that are adjacent to its current node.

We employ a discrete bag-of-features representation for state descriptions.[5] Using this representation, we can easily vary the type and amount of information given to the agent. Specifically, we simulate two settings of descriptions: *dense* and *sparse*. At test time, the agent perceives sparse descriptions and request the assistant for dense descriptions. A dense description of a current location contains the room name at the location, and the features of $M$ objects restricted to be within $\delta$ meters of the location. The features of each object consists of (i) its name, (ii) horizontal and vertical angles (relative to the current viewpoint), and (iii) distance (in meters) from the object to the current location. A dense description of a goal follows the same representation scheme. In the sparse setting, the current-location description does not include the room name. Moreover, we removing the features of objects that are not in the top 100 most frequent objects, essentially emulating a flawed object detector module. The sparse goal description (the task request) has only features of the target object and the room name where the object is located at. Especially, if a subgoal location is adjacent or coincides with the agent's current location, instead of describing room and object features, the human specifies the ground-truth action to go to the subgoal (an action is specified by its horizontal and vertical angles, and travel distance).[6]

**Experimental Procedure.** We conduct our experiments in three phases. In the *pre-training* phase, we learn a navigation policy $\hat{\pi}$ with dense descriptions of the current location and the goal. In the *training* phase, the agent perceives a sparse description of its current location and is given a sparse goal description. We use advantage actor-critic (Mnih et al., 2016) to learn an interaction policy $\psi_\theta$ that controls how the agent communicates with the human and navigates in an environment. The

[4]Let $p_t$ be the shortest path from $s_t$ to $g_t$, and $p_{t,i}$ be the $i$-th node on the path. The subgoal location is chosen as $p_{t,k}$ where $k = \min(\lfloor |p|/2 \rfloor, l_{\text{max}})$, where $l_{\text{max}}$ is a pre-defined constant.

[5]While our representation of state descriptions simplifies the object/room detection problem for the agent, it does not necessarily make the navigation problem easier than with image input, as images may contain information that is not captured by our representation (e.g., object shapes and colors, visualization of paths).

[6]Here, we emulate a practical scenario where if a destination is visible in the current view, to save effort, a human would concisely tell an agent what to do rather than giving a verbose description of the subgoal.

Table 1: Main results on test sets. For success rate, we report both goal-finding (normal font) and request-fulfilling results (smaller grey font in parentheses). We also report the average number of different types of actions taken by the agent (across all environment types). Assistance dramatically improves success rates, especially for unseen objects ($4\times$) and environments ($6\times$), with a small number of additional actions ($\sim 2.8$).

| Agent | Success Rate % ↑ | | | Avg. number of actions ↓ | | | |
|---|---|---|---|---|---|---|---|
| | Unseen Start | Unseen Object | Unseen Environment | CUR | GOAL | SUB | DO |
| **No assistant and interaction policy $\psi_\theta$** | | | | | | | |
| ($d^s$: current-state description, $d^g$: goal description) | | | | | | | |
| Sparse $d^s$ and $d^g$ | 43.4 (50.4) | 16.4 (23.2) | 3.0 (6.8) | - | - | - | 13.1 |
| Sparse $d^s$, dense $d^g$ | 67.2 (68.4) | 56.6 (58.2) | 9.7 (12.3) | - | - | - | 12.6 |
| Dense $d^s$, sparse $d^g$ | 77.9 (86.0) | 30.6 (40.3) | 4.1 (7.5) | - | - | - | 12.0 |
| Dense $d^s$ and $d^g$ | 97.8 (98.1) | 81.7 (83.3) | 9.4 (11.9) | - | - | - | 11.0 |
| **With assistant and interaction policy $\psi_\theta$** | | | | | | | |
| Rule-based $\psi_\theta$ (baseline) | 78.8 (78.8) | 68.5 (68.5) | 12.7 (12.7) | 2.0 | 1.0 | 1.7 | 11.3 |
| RL-learned $\psi_\theta$ (ours) | 85.8 (86.8) | 78.2 (79.6) | 19.8 (22.5) | 2.1 | 1.0 | 1.7 | 11.2 |
| + Perfect nav. on sub-goals (skyline) | 94.3 (95.8) | 95.1 (96.1) | 92.6 (94.3) | 0.0 | 0.0 | 6.3 | 7.3 |

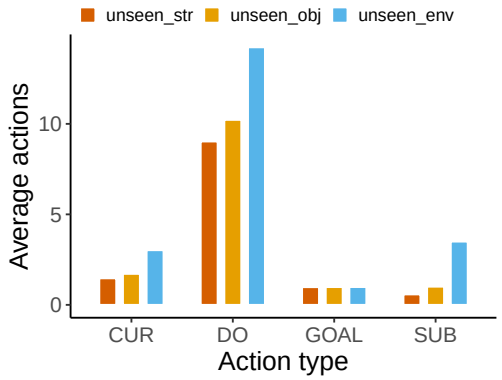
interaction policy is trained in environments that are previously seen as well as unseen during pre-training. Finally, in the *evaluation* phase, the interaction policy is tested on three conditions: seen environment and target object type but starting from a new room (UNSEENSTR), seen environment but new target object type (UNSEENOBJ), and new environment (UNSEENENV). The pre-trained navigation policy $\hat{\pi}$ is fixed during the training and evaluation phases. We create 82,104 examples for pretraining, 65,133 for training, and approximately 2,000 for each validation or test set. Details about the training procedure and the dataset are included in the Appendix.

## 6 RESULTS AND ANALYSES

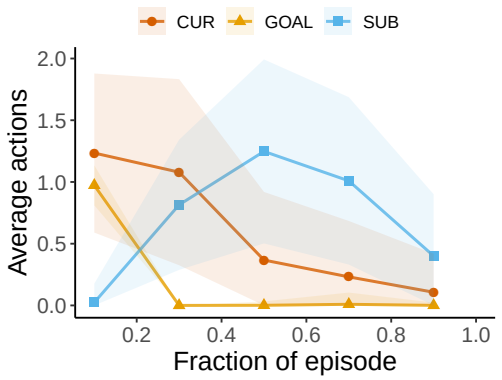
**Settings.** In our main experiments, we set: the cost of taking a CUR, GOAL, SUB, or DO action to be 0.01 (we will consider other settings subsequently); the cost of calling DONE to terminate the main goal (i.e. the task error) equal the (unweighted) length of the shortest-path from the agent's location to the goal; and the goal stack's size ($L$) to be 2. We compare our RL-learned interaction policy with a rule-based baseline that first takes the GOAL action and then randomly selects actions. In each episode, we enforce that the rule-based policy can take at most $\lfloor X_a \rfloor + y$ actions of type $a$, where $y \sim \text{Bernoulli}(X_a - \lfloor X_a \rfloor)$ and $X_a$ is a constant. We tune each $X_a$ on the validation sets so that the rule-based policy has the same average count of each action as the RL-learned policy. We also construct a skyline where the interaction policy is also learned by RL but with a navigation policy that executes subgoals perfectly. As discussed in § 5, we are primarily interested in goal-finding success rate and will refer to this metric briefly as success rate.

**Main Results (Table 1).** To inspect the potential benefits of asking for additional information, we compute how much the navigation policy $\hat{\pi}$ improves when it is supplied with dense information about the current and/or goal states. As seen, success rate of the navigation policy is lifted dramatically when both the current-state and goal descriptions are dense ($\sim 2\times$ increase on UNSEENSTR, $\sim 5\times$ on UNSEENOBJ, and $\sim 3\times$ on UNSEENENV). We find that dense information about the current state is more helpful on UNSEENSTR, while dense information about the goal is more valuable on UNSEENOBJ and UNSEENENV. This is reasonable because on UNSEENSTR, the agent has been trained to find similar goals. In contrast, the initial goal descriptions in UNSEENOBJ and UNSEENENV are completely new to the agent, thus gathering more information about them is necessary.

Aided by our RL-learned interaction policy, the agent observes a substantial $\sim 2\times$ increase in success rate on UNSEENSTR, $\sim 5\times$ on UNSEENOBJ, and $\sim 7\times$ on UNSEENENV, compared to when performing tasks using only the navigation policy. In unseen environments, with its capability of requesting subgoals, the agent impressively doubles the success rate of the navigation policy that has access to dense descriptions. The RL-learned interaction policy is significantly more effective

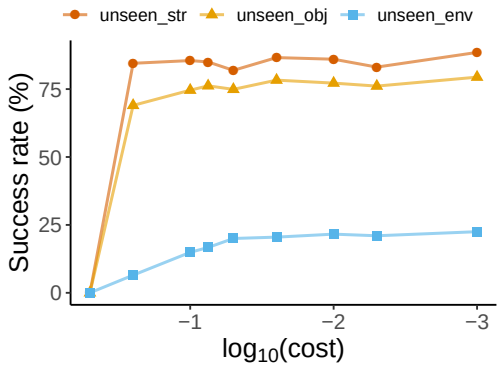



(a) How frequently does the interaction policy execute different actions, on average across different types of environments? Subgoals are requested much more in unseen environments.

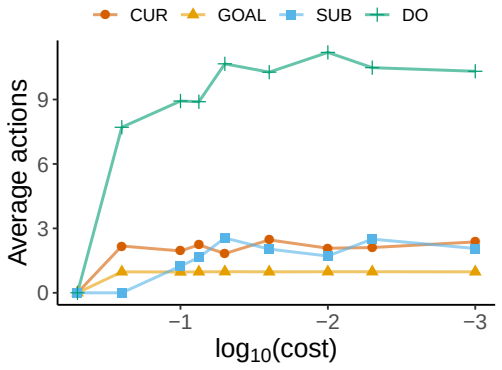



(b) Over the course of a trajectory, how does the frequency of different types of actions change in unseen environments? Subgoals are requested in the middle, goal information at the beginning.

Figure 2: Analyzing the behavior of the RL-learned interaction policy (on validation environments).




(a) Effect of cost on success rate.




(b) Effect of cost on action counts.

Figure 3: Analyzing the effect of simultaneously varying the cost of the CUR, GOAL, SUB, DO actions (on validation environments), thus trading off success rate versus number of actions taken.

than the rule-based baseline. The policy is also efficient: on average, it makes about 4.8 requests to the assistant (less than 1/4 of all the actions it takes during an episode).

On UNSEENSTR and UNSEENOBJ, the RL-learned policy has not closed the gap with the navigation policy that performs tasks with dense descriptions. Our investigation finds that limited information often causes the policy to not request information about the current state (i.e. taking CUR) and make incorrect decisions. Encoding uncertainty in the current-state description (e.g. Finkel et al. (2006)) is a plausible future direction for tackling this issue. Finally, results obtained by replacing the learned navigation policy with one that behaves optimally on subgoals shows that further improving performance of the navigation policy on short-distance goals would effectively enhance its performance on long-distance goals.

**Behavior of the RL-learned interaction policy.** Figure 2a characterizes behaviors of the RL-learned interaction policy in three evaluation conditions. We expect that tasks in the UNSEENSTR are the easiest and those in UNSEENENV are the hardest. As the difficulty of the evaluation condition increases, the interaction policy issues more CUR, SUB, and DO actions. The average number of GOAL actions does not vary, showing that the interaction policy has correctly learned that making more than one goal-clarifying request is unnecessary. Figure 2b illustrates the distribution of each action along the length of an episode in the validation UNSEENENV dataset. The GOAL action, if taken, is always taken only once and immediately in the first step. The number of CUR action

Table 2: Success rates and numbers of actions taken with different stack sizes (on validation). Larger stack sizes significantly aid success rates in unseen environments, but not in seen environments

| | Goal-finding success rate (%) ↑ | | | Average number of actions ↓ | | | |
|---|---|---|---|---|---|---|---|
| Stack size | Unseen Start | Unseen Object | Unseen Environment | CUR | GOAL | SUB | DO |
| 1 (no subgoals) | 92.2 | 78.4 | 12.5 | 5.1 | 1.9 | 0.0 | 10.7 |
| 2 | 86.9 | 77.6 | 21.6 | 2.1 | 1.0 | 1.7 | 11.2 |
| 3 | 83.2 | 78.6 | 33.5 | 1.3 | 1.0 | 5.0 | 8.2 |

gradually decreases over time. The agent makes most SUB requests in the middle of an episode, after its has attempted but failed to accomplish the main goals. We observe similar patterns on the other two validation sets.

**Effects of varying action cost.** As mentioned, we assign the same cost to each CUR, GOAL, SUB, or DO action. Figure 3a demonstrates the effects of changing this cost on the success rate of the agent. Setting the cost equal to 0.5 makes it too costly to take any action, inducing a policy that always calls DONE in the first step and thus fails on all tasks. Overall, the success rate of the agent rises as we reduce the action cost. The increase in success rate is most visible in UNSEENENV and least visible in UNSEENSTR. Figure 3b provides more insights. As the action cost decreases, we observe a growth in the number of SUB and DO actions taken by the interaction policy. Meanwhile, the numbers of CUR and GOAL actions are mostly static. Since requesting subgoals is more helpful in unseen environments than in seen environments, the increase in the number of SUB action leads the more visible boost in success rate on UNSEENENV tasks.

**Performing tasks with deeper goal stacks.** In Table 2, we test the functionality of our framework with a stack size 3, allowing the agent to request subgoals of subgoals. As expected, success rate on UNSEENENV is boosted significantly (+11.9% compared to using a stack of size 2). Success rate on UNSEENOBJ is largely unchanged; the agent makes more SUB requests (averagely 4.5 requests per episode compared to 1.0 request made when the stack size is 2), but doing so does not further enhance performance. The agent makes less CUR requests, possibly in order to offset the cost of making more SUB requests. Due to this behavior, success rate on UNSEENSTR declines with larger stack sizes, as information about the current state is more valuable for these tasks than subgoals. These results show that the critic model overestimates the $V$ values in states where SUB actions are taken, leading to the agent learning to request subgoals more than needed.

## 7 RELATED WORK AND CONCLUSION

To decide when to ask humans for help, previous work measures uncertainty of output distributions (Nguyen et al., 2019; Zhu et al., 2008) or directly train an asking policy via interaction (Chi et al., 2020; Nguyen & Daumé III, 2019; Kolb et al., 2019). In these approaches, the agent only calls for generic help assistance and does not convey specific information-seeking intent. Teaching agents to generate questions by mimicking human-generated language utterances has been explored in constructing dialog systems (Roman et al., 2020; Rao & Daumé III, 2018; Das et al., 2017; Li et al., 2016) or generating natural-language explanations of model decisions (Camburu et al., 2018; Chen et al., 2021; Narang et al., 2020; Kim et al., 2018). As discussed previously, naively mirroring human external behavior cannot enable agents to understand the limits of their own knowledge. Our work is closely related to He et al. (2013), where an agents learn to select a subset of most useful input features. However, the work assumes the features arrive in a fixed order, and the agent accepts or rejects a feature offered to it. In contrast, our agent directly requests what type of information it needs. To teach agents to form and communicate information-seeking intents, our work enhances the single-agent POMDP model in two ways: (a) it allows an agent to interact with an assist to leverage their knowledge and (b) it allows the agent construct more complex goal hierarchy by employing a stack data structure, whereas most previous formulations of hierarchical reinforcement learning (Le et al., 2018; Kulkarni et al., 2016; Sutton et al., 1999) is only restricted to two levels of goals. Enhancing the sample efficiency of the learning policy by exploiting the hierarchical policy structure is an exciting future direction. Furthermore, techniques for generating faithful explanations (Kumar & Talukdar, 2020) can be applied to enhance the specificity of the generated questions.

## REFERENCES

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3674–3683, 2018.

Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-snli: Natural language inference with natural language explanations. In *Proceedings of Advances in Neural Information Processing Systems*, 2018.

Oana-Maria Camburu, Brendan Shillingford, Pasquale Minervini, Thomas Lukasiewicz, and Phil Blunsom. Make up your mind! adversarial generation of inconsistent natural language explanations. *arXiv preprint arXiv:1910.03065*, 2019.

Hanxiong Chen, Xu Chen, Shaoyun Shi, and Yongfeng Zhang. Generate natural language explanations for recommendation. *arXiv preprint arXiv:2101.03392*, 2021.

Ta-Chung Chi, Minmin Shen, Mihail Eric, Seokhwan Kim, and Dilek Hakkani-tur. Just ask: An interactive learning framework for vision and language navigation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 2459–2466, 2020.

Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José MF Moura, Devi Parikh, and Dhruv Batra. Visual dialog. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 326–335, 2017.

Jenny Rose Finkel, Christopher D Manning, and Andrew Y Ng. Solving the problem of cascading errors: Approximate bayesian inference for linguistic annotation pipelines. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pp. 618–626, 2006.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

He He, Hal Daumé III, and Jason Eisner. Dynamic feature selection for dependency parsing. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1455–1464, 2013.

Yicong Hong, Qi Wu, Yuankai Qi, Cristian Rodriguez-Opazo, and Stephen Gould. A recurrent vision-and-language bert for navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Jinkyu Kim, Anna Rohrbach, Trevor Darrell, John Canny, and Zeynep Akata. Textual explanations for self-driving vehicles. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 563–578, 2018.

Benjamin Kolb, Leon Lang, Henning Bartsch, Arwin Gansekoele, Raymond Koopmanschap, Leonardo Romor, David Speck, Mathijs Mul, and Elia Bruni. Learning to request guidance in emergent communication. *arXiv preprint arXiv:1912.05525*, 2019.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29:3675–3683, 2016.

Sawan Kumar and Partha Talukdar. NILE : Natural language inference with faithful natural language explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8730–8742, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.771. URL `https://aclanthology.org/2020.acl-main.771`.

Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé. Hierarchical imitation and reinforcement learning. In *International conference on machine learning*, pp. 2917–2926. PMLR, 2018.

Jiwei Li, Alexander H Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. Learning through dialogue interactions by asking questions. *arXiv preprint arXiv:1612.04936*, 2016.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Sharan Narang, Colin Raffel, Katherine Lee, Adam Roberts, Noah Fiedel, and Karishma Malkan. Wt5?! training text-to-text models to explain their predictions. *arXiv preprint arXiv:2004.14546*, 2020.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287, 1999.

Khanh Nguyen and Hal Daumé III. Help, anna! visual navigation with natural multimodal assistance via retrospective curiosity-encouraging imitation learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, November 2019. URL `https://arxiv.org/abs/1909.01871`.

Khanh Nguyen, Debadeepta Dey, Chris Brockett, and Bill Dolan. Vision-based navigation with language-based assistance via imitation learning with indirect intervention. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. URL `https://arxiv.org/abs/1812.04155`.

Sudha Rao and Hal Daumé III. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2737–2746, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1255. URL `https://aclanthology.org/P18-1255`.

Homero Roman Roman, Yonatan Bisk, Jesse Thomason, Asli Celikyilmaz, and Jianfeng Gao. RMM: A Recursive Mental Model for Dialog Navigation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2020. URL `https://arxiv.org/abs/2005.00728`.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Jingbo Zhu, Huizhen Wang, Tianshun Yao, and Benjamin K Tsou. Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pp. 1137–1144, 2008.