Smart Help: Strategic Opponent Modeling for Proactive and Adaptive Robot Assistance in Households

Supplementary Material

Task	Goals (example)
MakeBreakfast	(Get, Potato), (KeepOff, Mircowave), (KeepOpen, Mircowave), (In, Potato, Microwave), (KeepClose, Microwave), (KeepOn, Microwave), (KeepOff, Microwave)
ArrangeRoom	(Get, Potato), (KeepOpen, Fridge), (In, Potato, Fridge), (KeepClose Fridge),
MakeCoffee	(Get, Mug), (In, Mug, CoffeeMachine), (KeepOn, CoffeeMachine), (KeepOff, CoffeeMachine)

Table 1. The table showcases examples of the parsing process, from overarching tasks to specific goals. This process may vary across different scenes and may also be probabilistic within the same scene. For instance, in certain scenarios, the mug may already be situated in the coffee machine. In such cases, the agent only needs to activate and then deactivate the machine to complete the "MakeCoffee" task.

Overview

This supplementary material includes:
- Appendix A has the detailed setting of the environment.
- Appendix B has the model implementation details.

A. Environment Setting

Scenes We use 30 scenes from AI2THOR kitchen scenes, which are shown in Fig. 1. In these scenes, 20 are used for training while the remaining 10 are reserved for testing. The object composition in each scene is diverse. We only pay attention to objects that were common across all scenes when parsing the task to goals.

Tasks We have classified our tasks into three categories: MakeBreakfast, ArrangeRoom, and MakeCoffee. Each task is further divided into a series of goals. The Tab. 1 shows some examples about the parsing process. Different room will leads to different parsing. For example, in some rooms the mug has been already put in the coffee machine, thus the main agent does not need to pick up the mug and put it in the coffee machine.

Goals Goals denote the target states that an agent must reach before successfully completing a task. We have identified seven potential goals for the agent to choose from. These are: (Wait), (Get, e_i), (On, e_i, pr_i), (In, e_i, pr_i), (KeepOpen, e_i), (KeepClose, e_i), (KeepOn, e_i), and (KeepOff, e_i). In certain scenarios, the sequence of goals is predetermined. This means that the agent must accomplish the goals in the given order, rather than arbitrarily. For instance, the microwave must be closed before it can be toggled on. Failing to follow this sequence results in an invalid action, as a microwave cannot be activated while open. Moreover, once a specific state has been reached, certain properties of that state need to be maintained until the next goal indicates a change. For example, if the goal sequence comprises (Open, Fridge) followed by (Put, Bread, Fridge), the agent must keep the fridge open after the first goal is achieved, until successfully accomplishing the ensuing goal of placing the bread in the fridge. These complexities present challenges both in the parsing process, which converts tasks into goals, and in the execution of these goals, adding layers of difficulty to the agent's tasks.

Intentional action We have defined seven intentional actions that the agent can perform: (Wait), (PickUp, e_i), (Put, pr_i), (ToggleOn, e_i), (ToggleOff, e_i), (Open, e_i), and (Close, e_i). Upon selecting an intentional action, a low-level planner devises a sequence of executable actions to be sent to the AI2THOR simulator. It's important to note that actions may fail due to various reasons. For instance, the action (Put, pr_i) will result in failure if the agent is not currently holding any objects. Similarly, an action to open an item that cannot be opened is destined to fail. Furthermore, the validity of an action can be context-specific. For example, while a microwave can typically be opened, it cannot be opened if it is currently active; it must first be toggled off. This context-dependent validity of actions presents a significant challenge for the model. It necessitates the model to learn the intricate dependency relationships between the context and the action, based on the reward and observation data provided by the environment.

Capability Some examples about how the capability will affect agent's interaction with different objects are shown in Fig. 2. We represent the capability by α, β, γ, δ, ϵ, and ζ.
- $\alpha_i \in [0, 1]$ denotes the maximum height that the agent can reach;
- $\beta_i \in [0, 1]$ represents the maximum weight that the agent can lift;
- $\gamma_i, \delta_i, \epsilon_i, \zeta_i \in [0, 1]$ represent the agent's ability to complete open, close, toggle on, and toggle off tasks, respectively.

In this challenge, we have seven types of agents to select from. These include agents with full capabilities and agents with one randomly assigned limitation. An agent with full

Figure 1. In our study, a total of 30 distinct scenes from AI2-THOR were utilized to train our model. The planforms of them are provided here. These scenes were divided into two datasets: 20 for training while the remaining 10 were reserved for testing. Different scenes have different size, but the figures presented here are scaled to the same size. The object composition in each scene is diverse. Thus the agent must learn a general policy for different scenes. However, for our tasks, the focus was solely on those objects that were common across all scenes. This approach was adopted to ensure that the task was consistently applicable across all the scenes used in the study.

capabilities will have $(\alpha, \beta, \gamma, \delta, \epsilon, \zeta) = (1, 1, 1, 1, 1, 1)$. If the agent has a limitation in α, the value will fall within the range of $(0.2, 0.8)$. The same concept applies to other capabilities, where β will be within $(0.1, 0.7)$, and $\gamma, \delta, \epsilon, \zeta$ will range from $(0, 0.49)$.

In the training process, the main agent will be randomly set to a kind of capability. In the testing process, we will set the main agent capability to the lowest level. For example, when the main agent's type is β limitation, his capability is $(1, 0.1, 1, 1, 1, 1)$.

Type	Capability
Full capability	(1, 1, 1, 1, 1, 1)
α limitation	(α, 1, 1, 1, 1, 1), $\alpha \in (0.2, 0.8)$
β limitation	(1, β, 1, 1, 1, 1), $\beta \in (0.1, 0.7)$
γ limitation	(1, 1, γ, 1, 1, 1), $\gamma \in (0, 0.49)$
δ limitation	(1, 1, 1, δ, 1, 1), $\delta \in (0, 0.49)$
ϵ limitation	(1, 1, 1, 1, ϵ, 1), $\epsilon \in (0, 0.49)$
ζ limitation	(1, 1, 1, 1, 1, ζ), $\zeta \in (0, 0.49)$

Table 2. We sample agents from seven different types. The types also include the sampling of agents with full capabilities, serving as a benchmark to examine the model's ability to differentiate between agents with limitations and those without. The range of capabilities for an agent with limitations is established based on the statistical information derived from the scenes.

B. Implementation Details

B.1. Trajectory dataset

We assembled a main agent trajectory dataset for pretraining the opponent modeling module. The pretraining phase equips the model with a more effective starting point for the subsequent reinforcement learning (RL) training. During data collection, we employ a human-derived expert policy to guide the main agent's actions. Simultaneously, the helper will randomly choose an action, with its observations and the ground label constituting the dataset. This dataset comprises 60,024 trajectories, each containing observations at five discrete time points and the final goal of the main

Figure 2. This figure shows the impact of different capabilities, represented by α, β, γ, δ, ϵ, and ζ, on the agent's interaction with various objects. α denotes the agent's ability to lift objects of different weights. A lower α value implies a limited capacity to handle only lighter objects, such as a potato (weight 0.18) or a tomato (weight 0.20). However, as α increases, the agent can manage heavier items, such as bread (weight 0.7) or a cup (weight 1.0). These object weights are determined by the AI2THOR simulator [1], reflecting the careful considerations of its developers. β symbolizes the agent's ability to reach varying heights. The floor, with a height of 0, is the most accessible, while the other objects' height, like the cabinet's height, determined by its presence in the scene, poses a greater challenge. γ, δ, ϵ and ζ respectively represent the agent's ability to perform specific operations: opening, closing, toggling on, and toggling off. We assume a threshold of 0.5, which serves as the tipping point between success and failure in these actions under normal circumstances.

Goal	KeepClose	Get	In	On	Open	ToggleOff	ToggleOn	Wait	None
# Trajectory	4192	8295	3420	214	5066	2384	2337	12953	21154

Table 3. **Dataset Statistics.** Our dataset is characterized by an imbalance in the number of trajectories corresponding to each goal. This discrepancy is addressed during the training process through a rebalancing strategy. The frequency of "On" goals is significantly lower compared to the others. This is primarily due to the interchangeable use of "In" goals to represent "On" goals. This substitution is reasonable as the agent can achieve them via the same singular "Put" action. This substitution is used particularly during the parsing of the task "ArrangeRoom", which aids in maintaining uniform terminology across the parsing.

Property	Data type	Method	Output size
Object type	Int	Embedding	32
Parent receptacle	Int	Embedding	32
Height	Float	MLP	32
Weight	Float	MLP	32
Position	Float	MLP	32
Distance	Float	MLP	32
Other properties	Bool	MLP	32

Table 4. The properties of object and how to change them to embedding. Here other properties include 'isPickedUp', 'isOpen', 'isCooked', 'isToggled', and 'isVisible'.

agent. When at all of the five discrete time points the helper can not observe the main agent, the label of the main agent's goal will be set to None. The statistics of the dataset is shown in Tab. 3.

B.2. Opponent Modeling Module

We train an opponent modeling module utilizing the main agent trajectory dataset.

First, we need to change the observations to embedding with the object encoder. We use MLPs and embedding model to achieve this.

After the change, we will get a feature will size 224 for every object. Then we use a transformer to handle all the features. The transformer has 8 heads, 0.2 dropout, hidden size of 128, and 4 layers. With the transformer, we get the object feature.

Property	Data type	Method	Output size
Held object type	Int	Embedding	32
Action type	Int	Embedding	32
Action success	Bool	MLP	32
Position	Float	MLP	32
Rotation	Float	MLP	32

Table 5. The observations of objects and how to change them to embedding.

For dealing with the observation of agents, we use the agent encoder 5. After the model processing, we concatenate the feature of action type and action success and put them into a MLP to get the action feature. Then we use other three features and the action feature to get the agent feature with MLP.

The object feature and the agent feature are concatenate to form the state feature. Next, we concatenate state features of five time points and put them into a MLP. The time feature output by the MLP has size of 128. Then, the time feature is fed into two transformers, and then we will get two features, which represent goal and capability respectively. The transformer has two heads, 0.1 dropout, hidden size of 256, and 4 layers. The parameters of these two transformers are not shared. Then we concatenate the time feature and the goal feature and put them to a MLP. The we get the feature of o_1, whose size is 128. Then we concatenate the time feature, the goal feature and the o_1 feature and get o_1 feature with a MLP. The o_1 feature represents the object that the goal involves. The o_2 feature represents the parent receptacle that the goal involves.

In the training process, we use four MLP as classifiers to get the probability of goals, the relevant objects, and the predicted capabiities. The we use cross-entropy loss and the Adam optimizer, with a learning rate of 1×10^{-6} and a batch size of 32.

We achieved a commendable prediction accuracy of 83.6%. This high level of precision indicates that our model possesses a remarkable capability to infer the goals of human agents based on a constrained set of observational data.

B.3. Helping Model

Our model accepts symbolic observations as input, which encompass the helper's partial observation, as well as the symbolic agent states of both the helper and the main agent. The observation includes the symbolic states of all objects within the helper's field of view. The object encoder and the agent encoder have been introduced in the last part. With the assistance of our pre-trained opponent modeling module, we can effectively model the main agent using its trajectory. It is worth noting that the classifiers from the opponent modeling module are not utilized within the helper model. Instead, the latent representation from the opponent modeling module is concatenated with the observation. This composite embedding is then processed through a linear layer and a ReLU function before being input into a single-layer LSTM with 512 hidden units. To generate an action policy, we apply a linear layer to the output of the LSTM, with no additional nonlinearity required. This streamlined approach ensures the effective processing of symbolic observations and accurate policy generation.

B.4. LLM

In our setup, LLM, i.e., "gpt-3.5-turbo-instruct" is used as a planner to generate a policy for the helper agent. For each scene, we provide the basic information of this environment, as shown in Fig. 3. At each step, we give the partial observations of scenes and agents to the LLM, as shown in Fig. 4. Besides, as shown in Fig. 5, we stipulate how the LLM should output to select a valid action, and give an example to help it understand. The LLM processes the prompt and produces an action decision based on its understanding of the given information. If the LLM outputs an invalid action decision, the "Wait" action will be adopted as its next action. To assess the effectiveness of the LLM-generated actions, we execute this action in the AI2-THOR simulator to determine how well the LLM performs as a planner for the helper agent in the given environment.

B.5. MCTS-heuristic

- **MCTS.** We implement a MCTS algorithm which outputs an action based on a given goal, whether predicted, ground truth, or randomly selected. In each simulation, it samples an action under the guidance of the Upper Confidence Bound (UCB), which balances exploration and exploitation according to node values and visit counts. Then it performs a rollout to reach one possible end state, with the maximum rollout depth limited to 5 steps. If the goal is successfully achieved within this depth, the end state's value is evaluated by:

$$v = 50/d$$

  where v represents the value and d the number of steps taken to complete the goal. The algorithm returns the action represented by the most visited node after $n = 500$ simulations.
- **MCTS-heuristic.**To enhance search efficiency and task completion capabilities, we use a heuristic MCTS method, which incorporates hand-written rules that are specifically defined to break down a goal into a sequence of actions necessary for task completion. This model employs a probabilistic strategy during the sampling of the MCTS, with a probability parameter p_{sample} determining whether to sample a random action for rollout or, at $1 - p_{\text{sample}}$, select the next action from the rule-based sequence, excluding completed actions. When p_{sample} is set

There are two agents in the environment: you and your partner. Your goal is to help your partner. To do this, you need to observe the current environment to infer the task and goal of your partner. There are three possible tasks: MakeBreakfast, MakeCoffee, and ArrangeRoom. There are seven possible goals: Wait, PickUp something, Put something on somewhere, ToggleOn something, ToggleOff something, Open something, Close something. To complete the task, your partner will separate the task into several goals and complete them successively. However, your partner's capability may be lacking. He will fail in some actions. There are six kinds of capabilities: mass, height, open, close, toggle_on, toggle_off, and your partner will fail in none or several kinds of capabilities. Mass and height determine the maximum weight and height that the helper can pick up, and the last four capabilities will affect the success of the corresponding action. Then you will receive the objects' information which are in your sight, and the state of your partner.

Figure 3. **The LLM prompt (the first part)**. We provide the basic setting information for the LLM, including what needs to be completed, what can be done, what types of the main agents are in the environment, etc.

The objects and their properties are as follows:
<0, Bread, has no parent receptacle, it can be picked up, it cannot be opened, it cannot be cooked, it cnnnot be toggled on or off, its height is 0.73 and its weight is 0.70, its position is x:-0.25, y:0.73, z:0.88, its distance from main agent is 0.41.>
……
<5, Window, has no parent receptacle, it cannot be picked up, it cannot be opened, it cannot be cooked, it cnnnot be toggled on or off, its height is 1.45 and its weight is 0.00, its position is x:0.32, y:1.45, z:3.09, its distance from main agent is 2.71. >

All the actions are as follows: 0, Wait. 1, PickUp. 2, Put. 3, ToggleOn. 4, ToggleOff. 5, Open. 6, Close.

Your partner has taken PickUp at the last step. But his last action failed. His previous actions are: [['PickUp', False], ……, ['PickUp', False]]. For a single action, it is [action, success]. His position is x:-0.25, y:0.9009991884231567, z:0.5. His rotation is x:-0.0, y:0.0, z:0.0. Now he is holding None. His position is x:-0.25, y:0.9009991884231567, z:0.5. His rotation is x:-0.0, y:0.0, z:0.0. Now he is holding Bread. You have taken PickUp at the last step. But your last action has failed.

Figure 4. **The LLM prompt (the second part)**. At each step, we provide the LLM with its partial observations. For scene observations, we enumerate all observable objects and their properties. For agent observations, we summarize the action trajectory of the main agent, the completion of his previous action, and whether this main agent holds an object. All the optional actions and the previous action information of the LLM will also be mentioned.

to 1, this model is equivalent to the *MCTS* model, and when it is set to 0, it strictly follows the rule-based action sequence. Initially, we set p_{sample} to 1 and 0.9 to encourage exploration for more efficient action sequences. The action sequences obtained through this exploratory search align with the hand-written rules, validating their optimality to a certain degree. During testing, we adjust $p_{\text{sample}} = 0.5$ to ensure a more efficient search and a higher success rate in task completion.

- **MCTS$_{\text{TG}}$ and MCTS$_{\text{RG}}$.** We develop *MCTS$_{TG}$* and *MCTS$_{RG}$* to replicate the baselines used in Watch-and-Help [2]. In the original work, a hierarchical planner was implemented, using MCTS for high-level planning and regression planning (RP) for low-level planning. Since our task environment does not require low-level planning, we use the *MCTS-heuristic* model as a counterpart of their planner in this specific environment. To provide comparison, we implemented two additional baselines: *MCTS$_{TG}$*, which has knowledge of the main agent's true goal, and *MCTS$_{RG}$*, which follows a random goal. As shown in the results table, the performance of *MCTS$_{RG}$* is similar to the *Random* agent, with minimal ability to provide assistance and complete tasks. This highlights the importance of a smart agent that can infer the main agent's goal. *MCTS$_{TG}$*, knowing the true goal of the main agent, achieves the best performance across all metrics. However, while it significantly surpasses other models in other metrics, completing the task better and faster, its *HN* is relatively low, indicating the limitation of its simple take-over helping strategy.

To help your partner, you need to infer which task and goal he is doing based on your observation of objects and agents, and infer his capability at the same time. Based on your inferred goal and capability, you can choose whether and how to help your partner. Only when you find your partner can not finish the task on his own, the help is appropriate. You must output your decision following the format of "action-object", e.g., "Put-Fridge" or "ToggleOn-Faucet". If you have not decided on an action, you can output "Wait-None". You can only output a single "action-object" pair at once. You must choose a concrete action and object for outputting. For example, ('action', 'object') is invalid, please output the action like ('PickUp', 'Cup') instead. Let's think step by step and output your action.

Figure 5. **The LLM prompt (the third part)**. We will specify the criteria for selecting a valid action through the LLM's output, along with providing illustrative examples to facilitate its comprehension.

References

[1] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017. 3

[2] Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B Tenenbaum, Sanja Fidler, and Antonio Torralba. Watch-and-help: A challenge for social perception and human-ai collaboration. *arXiv preprint arXiv:2010.09890*, 2020. 5