Tru-POMDP: Task Planning Under Uncertainty via Tree of Hypotheses and Open-Ended POMDPs

Wenjing Tang[1,2§], **Xinyu He**[2,3], **Yongxi Huang**[1,2§], **Yunxiao Xiao**[2,4], **Cewu Lu**[1,2], **Panpan Cai**[1,2†]
[1]Shanghai Jiao Tong University [2]Shanghai Innovation Institute
[3]East China Normal University
[4]Beijing University of Posts and Telecommunications

Abstract

Task planning under uncertainty is essential for home-service robots operating in the real world. Tasks involve ambiguous human instructions, hidden or unknown object locations, and open-vocabulary object types, leading to significant open-ended uncertainty and a boundlessly large planning space. To address these challenges, we propose Tru-POMDP, a planner that combines structured belief generation using Large Language Models (LLMs) with principled POMDP planning. Tru-POMDP introduces a hierarchical *Tree of Hypotheses* (TOH), which systematically queries an LLM to construct high-quality particle beliefs over possible world states and human goals. We further formulate an open-ended POMDP model that enables rigorous Bayesian belief tracking and efficient belief-space planning over these LLM-generated hypotheses. Experiments on complex object rearrangement tasks across diverse kitchen environments show that Tru-POMDP significantly outperforms state-of-the-art LLM-based and LLM-tree-search hybrid planners, achieving higher success rates with significantly better plans, stronger robustness to ambiguity and occlusion, and greater planning efficiency.[1]

1 Introduction

Home-service robots are increasingly expected to perform complex tasks in unstructured household settings, such as tidying up, preparing for guests, or assisting with daily routines. Among these, object rearrangement tasks—e.g., "prepare the kitchen for a party"—require robots to interpret ambiguous, open-ended instructions and manipulate open-vocabulary objects, many of which may be hidden in drawers, cabinets, or containers. The robot must infer user intent, identify relevant items, and plan long-horizon action sequences to satisfy under-specified goals in partially observable environments.

While these challenges can be positioned within the framework of planning under uncertainty—where a robot must hedge against imperfect perception, unobservable environment states, and stochastic action outcomes—the difficulties in open household environments extend far beyond standard formulations. First, human instructions are inherently ambiguous and often omit crucial details (e.g., what specific items are needed for a party). Second, many objects are hidden from the robot's initial view, and the set of relevant objects is unbounded and diverse (e.g., cups, snacks, decorations). As a result, the robot must reason over *open-ended uncertainty*, where fundamental components of the planning problem—such as the state space (e.g., what objects exist), the action space (e.g., what objects to operate on) and task goals (e.g., what the human want)—cannot be predefined. This setting leads to an effectively boundless belief space, making long-horizon planning extremely challenging.

[§]Wenjing Tang and Yongxi Huang are the visiting students at Shanghai Innovation Institute.
[†]Corresponding author: `cai_panpan@sjtu.edu.cn`
[1]The code and demonstration video are available at: `https://tru-pomdp.github.io`

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

To address open-world problems, recent work has started to leverage large language models (LLMs) to bring commonsense reasoning into robot planning. One line of work treats LLMs as planners, leveraging their implicit knowledge and ability to reflect over past failures [1, 2, 3, 4, 5, 6]. However, such approaches often rely on implicit, unverified hypotheses about the world, and can fail catastrophically when the underlying hypotheses are incorrect. Another line combines LLMs with symbolic or probabilistic planners, for example, by prompting LLMs to generate POMDP models for informing LLM planning or suggesting default actions for search [7, 8, 9, 10, 11, 12]. While more structured, these methods still commit to a single high-probability hypothesis or operate in a closed domain, which limits robustness in the open-world.

To overcome these limitations, we propose Tru-POMDP, a task planner built upon an *open-ended POMDP* formulation, that tightly integrates commonsense reasoning by LLMs with *explicit* belief tracking and *principled* POMDP planning. Tru-POMDP operates in three stages: (1) it constructs a tree of hypotheses over possible world states and task goals using hierarchical LLM queries, to form a commonsense belief; (2) it fuses the LLM-generated belief with Bayesian filtering to construct a hybrid belief that is both diverse and consistent during updates; and (3) it solves for an optimal policy using online belief tree search with a dynamically constructed action space and guided by an LLM-generated rollout policy. This design leverages the generalization power of LLMs while maintaining rigorous, verifiable reasoning over open-ended uncertainty.

We evaluate Tru-POMDP on object rearrangement tasks across five diverse kitchen environments from RoboCasa [13], involving open-ended instructions, ambiguous user intentions, and a large number of hidden spaces. Results show that Tru-POMDP effectively tackles challenges brought by open-ended uncertainties. Through the integration of LLM-based belief generation with POMDP planning, Tru-POMDP significantly outperforms both LLM agents and LLM-augmented tree search planners, producing higher success rates and plan qualities with lower LLM token usage. Tru-POMDP also benefits from structured belief modeling with LLMs and hybrid belief tracking that combines LLM belief with Bayesian updates.

In summary, our main contributions include:

- The first framework to address "open-ended POMDPs", integrating LLM-based reasoning with *principled* POMDP planning for household tasks.
- A novel hybrid belief modeling approach that integrates LLM-generated hypotheses with *principled* Bayesian filtering.
- A POMDP model for open-world object rearrangement tasks and a practical belief-tree search planner for solving such tasks efficiently under large-scale uncertainty.

2 Background and Related Work

2.1 Online POMDP Planning

Partially Observable Markov Decision Processes (POMDPs) provide a principled framework for decision-making under uncertainty, where the true state of the world is only partially and noisily observed. A POMDP is formally defined as a 7-tuple (S, A, Z, T, O, R, b_0), where S is the set of states, A is the set of actions, Z is the set of observations, $T(s'|s, a)$ is the transition model, $O(z|s', a)$ is the observation model, $R(s, a)$ is the reward function, and b_0 is the initial belief over states. Online POMDP planning targets for single-query planning or even real-time planning, by computing policies via *belief tree search*. The agent maintains a belief—a probability distribution over possible world states—using Bayesian filtering. At each decision step, it constructs a belief tree by simulating future actions and observations [14], and applies Bellman's principle [15] in the tree to compute the best policy given the current belief. Only the immediate action is executed, and the process is repeated at each time step, allowing the planner to adapt to new observations in an online fashion (e.g., replan at 1 Hz). A POMDP model requires a full specification of all possible states, actions, and observations, which limits its use to "closed-domain" problems.

2.2 Robot Planning with Large Language Models

Recent advances in large language models (LLMs) have enabled robots to handle tasks expressed in open-vocabulary language and to incorporate commonsense knowledge into the planning process.

Broadly, two directions have emerged. The first uses LLMs *as planners*, either generating open-loop action sequences via prompt engineering or fine-tuning [4, 3, 16, 17, 5, 6, 18], or producing closed-loop actions through iterative feedback and reflection [2, 1, 19, 20, 21, 22, 23, 24]. While flexible, these approaches rely on implicit reasoning and often lack robustness when their internal hypotheses diverge from reality. The second direction combines LLMs with *explicit planning algorithms*. For example, some work uses LLMs to generate PDDL domains and problems, which are then solved by symbolic planners [25, 26, 27, 28, 29]. Others combine LLMs with Monte Carlo Tree Search (MCTS), using the LLM to hypothesize initial states or initial solutions [7, 8, 30, 31]. However, these methods generally assume full observability and deterministic goals, ignoring the inherent uncertainty in both human intention and world states. As a result, their solutions are often suboptimal in real-world deployments.

2.3 Planning under Uncertainty with Large Language Models

A few recent efforts have explored combining LLM reasoning with the structure of POMDPs. Some approaches use LLMs to generate elements of a POMDP model—such as beliefs or observation likelihoods—which are then fed back into the LLM to improve its planning ability under uncertainty [9, 10, 11, 12]. However, these methods still rely heavily on the LLM's implicit reasoning and lack the formal guarantees provided by explicit belief tracking and tree-search planning. Other work combines LLMs with tree search. One possibility is to generate a single most likely hypothesis about goals and world states, and then applies MCTS for planning [32, 33]. This strategy fails when the true goal diverges from the most likely one. LLM-MCTS [7] proposed to plan with LLM-generated state distributions. However, it uses a "flat" belief that only considers a single aspect of uncertainty on the initial placement of objects. It operates in a closed domain, assuming the object set is known beforehand, and incurs significant computational cost due to repeated LLM queries within the tree search [34]. In contrast, Tru-POMDP offers a scalable and flexible approach to address open-ended uncertainty. It uses the LLM to construct a *tree of plausible hypotheses* representing a complex, multi-aspect, and open-ended belief over entirely unknown objects and ambiguous human goals. We combine this with Bayesian filtering to maintain a consistently updatable belief. Finally, we apply belief tree search over this hybrid belief, using a dynamically constructed action space and a pre-compiled, LLM-generated rollout policy to efficiently guide optimal search.

3 Problem Formulation

3.1 Task Specification

We study the object rearrangement task performed by a dual-armed mobile robot in a household environment. Given a free-form, ambiguous natural language instruction I, the robot must arrange open-vocabulary objects existing in the scene into a target configuration that fulfills the human's underlying intention, while minimizing total execution time. The robot executes four types of actions: *MOVE* navigates to a specific area (e.g., in front of furniture or appliances); *PICK* picks an object from an open surface or an open container; *PLACE* places the held object onto a surface or into a container; and *OPEN* opens a container or drawer if it is closed. *MOVE* has a time cost proportional to the navigation distance, while the other actions incur a fixed cost. The robot can hold at most one object in its right hand; the left hand is reserved for the *OPEN* operation.

The environment is represented as a structured scene graph $\mathcal{G} = (V, E)$, where V denotes asset nodes and E encodes spatial relationships. Nodes are organized hierarchically into four layers: a root *ROOM* node representing the overall scene; *AREA* nodes representing surfaces or containers that can hold objects; *OBJECT* nodes representing physical items; and a *ROBOT* node indicating the robot's current location. Edges encode containment and adjacency (e.g., an object is at an area, or the robot is at an area). Each *AREA* node has a boolean attribute indicating whether it is open or closed. The robot can observe objects in open areas without noise but cannot observe contents in closed areas. While we assume noise-free observations, the method naturally extends to stochastic cases without affecting belief tracking or planning algorithm design.

Due to the inherent ambiguity of human instructions, we model the task goal as a set of plausible placement goals $\mathcal{P} = \{p_1, \ldots, p_n\}$. Each placement goal p_i specifies a list of target objects $\{o_1, \ldots, o_m\}$ and their respective desired destination areas $\{t_{o_1}, \ldots, t_{o_m}\}$. These specifications are

concise and do not constrain the locations of unrelated objects. Concrete goal representations for example tasks are provided in the appendix A.4.

3.2 The POMDP Model

States. A state $s \in S$ consists of a scene graph $\mathcal{G}_s$ encoding the current placement of both visible and hidden objects, and a hypothetical placement goal p_s of target objects, representing the intended final configuration consistent with the instruction of the task.

Observations. An observation $z \in Z$ is a partial scene graph $\mathcal{G}_z$ that includes only visible objects in open areas. The observation function is defined as:

$$O(s', a, z) = \Pr(\mathcal{G}_z \mid s' = (\mathcal{G}_{s'}, p_{s'})) = \begin{cases} 1, & \text{if } \mathcal{G}_z = \texttt{Visible}(\mathcal{G}_{s'}) \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where `Visible` removes all hidden objects from $\mathcal{G}_{s'}$.

Parameterized Actions and State Transition. The action space A is defined over parameterized operations on the scene graph. *OPEN(area)* opens a closed area, setting its status to open. *PICK(area, object)* picks an object from an area, reassigning its parent to the *ROBOT* node. *PLACE(area)* places the held object into the target area, making that area its new parent. These actions implicitly execute a *MOVE* when the robot is not in front of the specified area or object. The robot may also choose to take no action via *NULL*. Actions are subject to feasibility constraints: for instance, only closed areas can be opened. Infeasible actions are mapped to *NULL*.

Reward Modeling. Manipulation actions incur a fixed cost of 5, and navigation costs vary from 0 to 27 depending on the distance traveled. Infeasible actions are penalized with a cost of 100. Completing a subgoal yields a reward of 200, with an additional 200 granted upon full task completion.

Belief Modeling. The belief b is represented as a weighted particle set $\{(s_i, w_i) \mid s_i \in S, w_i \in (0, 1]\}$, where each s_i is a possible world state and w_i its associated probability. The initial belief is generated from the natural language instruction I and the initial observation $\mathcal{G}_z$ using LLMs, and updated using Bayesian filtering. Details of belief tracking are provided in Section 4.1.

4 The Tru-POMDP Planner

Tru-POMDP consists of three modules (Fig. 1): Tree of Hypotheses, Hybrid Belief Update, and Online POMDP Planning. Given a natural language instruction I and current observation $\mathcal{G}_z$, the Tree of Hypotheses module generates a belief over candidate world states and plausible placement goals $\mathcal{P}$. The Hybrid Belief Update module filters inconsistent hypotheses and augments the belief as needed. The Online POMDP module computes the optimal next action under uncertainty via belief tree search.

4.1 Tree of Hypotheses

This module queries an LLM to generate a diverse belief over three sources of uncertainty: ambiguity in human instructions, open-class objects, and occluded object locations. The LLM takes as input the human instructions I and the observed scene graph $\mathcal{G}_z$, transformed into a textual description T_z, and constructs a tree of hypotheses (TOH) consisting of three levels:

Level 1: Hypotheses of target objects. The LLM predicts sets of objects that are potentially relevant to the task described by I. Since we do not assume a known object set, the LLM has to hypothesize objects with commonsense priors, and returns a set of C_1 candidate combinations of target objects:

$$\mathcal{O} = \{(O_i, w_i) \mid i = 1, \ldots, C_1\}, \tag{2}$$

where each O_i is a set of objects that can be used to accomplish the task, and w_i is the LLM's confidence score for that hypothesis. These hypotheses reflect alternative interpretations of vague or underspecified instructions. The LLM also incorporates information from failed past executions (if available) to disambiguate user intent and generalize to open-class object names.

Level 2: Hypotheses of placement goals. For each combination of target objects, the LLM infers a destination area for each object in the set. This yields a plausible goal placement:

$$\mathcal{P}_{\text{goal}}(O_i) = \{(o_j, t_j) \mid j = 1, \ldots, |O|\}, \tag{3}$$

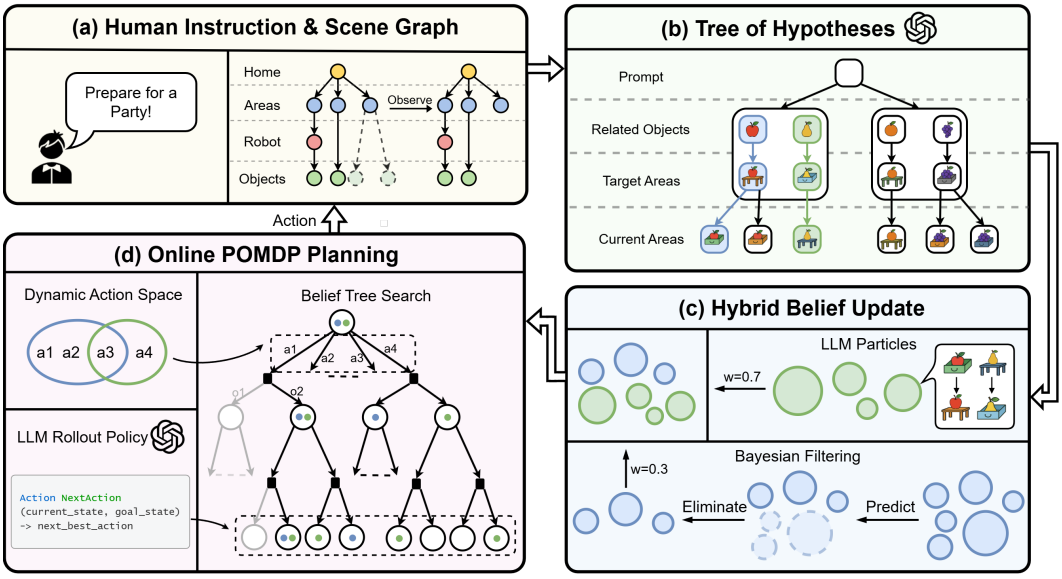


Figure 1: The architecture for Tru-POMDP. *(a) Task Input*: Human instruction and the observed scene graph. *(b) Tree of Hypotheses*: An LLM infers target objects, target areas, and initial locations, producing weighted particles. *(c) Hybrid Belief Update*: Bayesian filtering updates the belief using particle prediction and elimination, and augments the filtered belief with LLM particles. *(d) Online POMDP Planning*: Belief tree search computes the optimal action with the help of dynamic action branching and an LLM-written rollout policy.

which assigns each target object o_j in the combination O_i to its target *AREA* node t_j.

Level 3: Hypotheses of current placements. To estimate the current locations of target objects that are not visible in $\mathcal{G}_z$, an LLM is queried for each invisible target object o to predict a distribution over its possible locations, i.e., hidden areas:

$$\mathcal{P}_{\text{init}}(o) = \{(l_o, w_k) \mid k = 1, \ldots, C_2\}. \tag{4}$$

where a set of C_2 locations (l_o's) are generated, together with a confidence score w_k for each hypothesis. If o is observed in $\mathcal{G}_z$, its location is taken directly.

Inference for each invisible target object is queried independently, enabling parallelization.

Particle belief construction. Each path from the root to a leaf of the hypothesis tree defines a sampled particle $s = (\mathcal{G}_s, p_s)$, where $\mathcal{G}_s$ encodes a hypothesized complete scene graph (a union of observed scene graph $\mathcal{G}_z$ and inferred initial placements of invisible target objects), and p_s corresponds to the generated placement goal. Aggregating all such particles yields the LLM-inferred particle belief:

$$b_{\text{LLM}} = \{(s_n, w_n) \mid n = 1, \ldots, C_3\}. \tag{5}$$

where the particle weight w_n is proportional to the multiplication of all confidence scores, i.e., w_i's and w_k's along the top-down path. These particle weights are normalized to ensure the belief forms a valid probability distribution.

4.2 Hybrid Belief Update

The Hybrid Belief Update module maintains a consistent and efficient particle belief over partially observed world states and placement goals. It fulfills two objectives: (1) ensuring smooth and reliable belief updates through Bayesian filtering, and (2) reducing the frequency of expensive LLM queries by only invoking them when necessary. The algorithm monitors the total weight of the particle belief. If the belief retains sufficient weight—indicating strong agreement with the current observation—it is updated using a particle filter. If the weight falls below a threshold (e.g., $1 - \epsilon = 0.3$), new particles are generated by invoking the Tree of Hypotheses module (Section 4.1) to replenish the belief.

Bayesian Filtering. We employ a particle filter to perform principled belief updates. Each step consists of two phases:

Prediction step. For each particle $s_i = (\mathcal{G}_{s_i}, p_{s_i})$, we simulate the effect of the last action to update the scene graph $\mathcal{G}_{s_i}$ according to the deterministic transition model.

Elimination step. Given the new observation $\mathcal{G}_z$, particles inconsistent with it are removed. Because the observation model is deterministic (Equation 1), this reduces to filtering out particles whose predicted visible scene $\texttt{Visible}(\mathcal{G}_{s_i})$ does not match $\mathcal{G}_z$. The result is a new filtered particle set b_{BF} with total weight $w_{\text{BF}} \leq 1$.

LLM Particle Supplementation. When $w_{\text{BF}} < 1 - \epsilon$, the belief is augmented using a new set of particles generated by the Tree of Hypotheses. This new set, b_{LLM}, is generated by conditioning on the full history of past actions and observations. The augmented belief is defined as:

$$b_{t'} = b_{\text{BF}} + (1 - w_{\text{BF}}) \cdot b_{\text{LLM}}, \tag{6}$$

where the weights of particles in b_{LLM} are scaled by $(1 - w_{\text{BF}})$ before merging. This ensures that the resulting belief remains a valid probability distribution.

4.3 Online POMDP Planning

This module plans the optimal policy given the current belief over uncertain world states and goals. Our planner is built upon DESPOT [14], a sampling-based online POMDP algorithm that provides asymptotic optimality guarantees. We extend DESPOT with two key innovations: (1) a dynamic action space constructed from belief particles, and (2) an LLM-generated rollout policy that injects commonsense domain knowledge while maintaining computational efficiency. Tru-POMDP operates in an online planning setting: at each time step, it executes the optimal action for the current belief and replans after receiving new observations.

Dynamic Action Space. As discussed in Section 3.2, the robot's action space includes *PICK* actions over objects that may be hidden and unknown. Since the full set of possible objects is unbounded, naively enumerating all possible parameterized actions would result in an intractably large—or infinite—action space. To address this, we dynamically construct a compact action space based on the current belief. Concretely, we extract the union of all relevant entities from the particles in the belief: hypothesized target objects, open areas, and closed containers. We then construct the grounded action set consisting of: (1) *OPEN* for each closed area; (2) *PICK* for each target object that is currently visible and graspable; and (3) *PLACE* for placing the held object (if any) into each open area. This dynamic action set varies with the belief but remains compact and tractable, while retaining sufficient expressiveness for solving the task.

Belief Tree Search. We adapt the DESPOT algorithm to recursively explore the space of future action-observation sequences from the current belief. At each time step, a sparse belief tree is constructed: nodes represent future belief states (particle beliefs), and branches represent possible actions and the resulting observations, simulated using the POMDP transition and observation models. Each belief node also maintains a value estimate, representing the expected cumulative reward that can be achieved under optimal behavior conditioned on that belief. These values are updated iteratively using Bellman's principle [14]. The planner leverages value estimates as tree search heuristics: it selects actions that maximize the value estimate, and expands observation branches based on excess uncertainty. This ensures targeted exploration of promising belief trajectories, ensuring efficient convergence of values and allowing planning to terminate at any time and output high-quality decisions.

LLM-Generated Rollout Policy. Following the DESPOT framework, leaf nodes in our belief tree require rolling-out a default policy to provide a heuristic lower bound on their value. However, unlike prior work that simply uses a random rollout policy [7], we query the LLM to synthesize a rollout policy in code. The rollout policy is a mapping from the current state $s = (G_s, p_s)$ to an action a, where G_s is the scene graph representing the hypothesized world state, and p_s is the placement goal. This mapping is implemented as a C++ function generated by an LLM. The generator LLM receives a general language description of the object rearrangement domain but is given no information about any specific task instance. The resulting policy generalizes across tasks in the same domain and is reused throughout the search. During rollouts, this policy is simulated for each rolled-out node to estimate cumulative rewards from the leaves. See Appendix A.2 for the generator prompt and the code of the rollout policy.

5 Experiments

We conduct comprehensive experiments to evaluate the effectiveness of Tru-POMDP in open-ended object rearrangement tasks under uncertainty. Our study aims to answer the following questions:

1. Is belief-space planning essential for household tasks involving partially observable environments and ambiguous free-form instructions?
2. Can Tru-POMDP effectively solve household tasks with high and open-ended uncertainty?
3. What are the benefits of integrating belief-space planning with LLM-based reasoning?
4. How does combining Bayesian filtering with LLM belief generation improve performance?
5. Does the proposed Tree of Hypotheses (TOH) structure generate high-quality beliefs?

5.1 Experimental Setup

We evaluate Tru-POMDP in five kitchen environments from RoboCasa [13], which highlights a large number of hidden areas: *One Wall*, *One Wall with Island*, *L-Shaped*, *L-Shaped with Island*, and *Galley*. Each scene contains up to 40 *AREA* nodes with semantically-rich names, of which up to 29 are initially closed. Tasks are procedurally generated using an LLM-assisted pipeline, emphasizing vague task instructions with substantial implicit information and ambiguous object references. For each task, we populate up to 20 objects (including distractors) in the scene using commonsense priors, and define a goal set $\mathcal{P}$ containing various plausible target object combinations and goal placements grounded in the scene. This setup ensures each task is both solvable and exhibits ambiguity in human intention. Example scene with instruction and goal sets are listed in Appendix A.3 and A.4.

We categorize tasks into three difficulty levels based on the number of target objects required in the goal: *easy* (requiring 2 target objects), *medium* (3), and *hard* (4–8). Each additional target object in the goal leads to exponential growth of uncertainty. For each level, we generate 100 tasks, resulting in a total of 300 tasks. Step limits are set to 25, 30, and 35 respectively. The planning time is capped at 600 seconds. A task fails if either the step or time limit is exceeded.

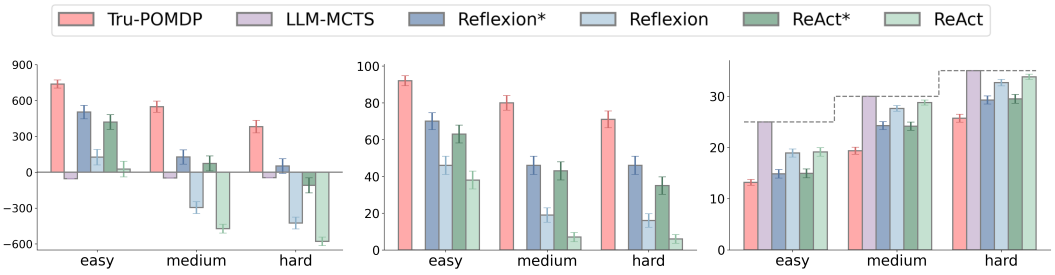
We evaluate performance using the following metrics. (1) *Cumulative reward:* the total reward collected in an episode, computed using the POMDP reward function described in Section 3.2. In experiment, we remove the reward for subgoals, and adapt the reward for full task completion to 1,000. In addition, we adapt the cost of infeasible actions to 25. (2) *Success rate:* the percentage of episodes in which the task is successfully completed within the step and time limits. (3) *Step count:* the number of execution steps per episode, where failed episodes are assigned the maximum allowed steps. (4) *Planning time:* the total wall-clock time spent in online planning across all steps of an episode. (5) *LLM token usage:* the total number of input and output tokens consumed by the planner, averaged per episode.

Experiments are run on a local machine equipped with a 12th Gen Intel® Core™ i7-12700KF CPU (20 threads), without GPU acceleration. All methods consistently use GPT-4.1 as the LLM.

5.2 Comparison Results

We compare Tru-POMDP against a set of strong baselines, including both pure LLM-based planners and tree search planners integrated with LLM reasoning.

- *ReAct* [1]: A closed-loop LLM-based planner that selects actions based on current observations and immediate feedback from the environment.
- *Reflexion* [2]: An extension of *ReAct* that introduces a reflection module. When repeated failures are encountered, it analyzes the execution history and generates a revised plan to guide subsequent decisions. For fair comparison in an online planning setup, we disable environment resets and trigger reflection after three consecutive failed actions.
- *ReAct** and *Reflexion**: Prompt-augmented variants of the above, in which the LLM receives additional structured descriptions of the task domain, including object types, action semantics, and goal structures. Detailed prompt templates are provided in Appendix A.6.
- *LLM-MCTS* [7]: LLM augmented closed-domain POMDP planning. Given a known object set, LLM generates object-location probability vectors as belief. It then performs Monte



(a) Cumulative Reward ↑ (b) Success Rate (%) ↑ (c) Total Step Number ↓

Figure 2: Performance comparison of Tru-POMDP and baselines. Each bar represents the average value with standard error (SE). In (c), the dashed line indicates the maximum allowed step number.

Carlo Tree Search (MCTS), calling the LLM repeatedly to guide the action selection in simulation procedure of MCTS.

As shown in Figure 2, Tru-POMDP significantly outperforms all baselines, demonstrating its capability for planning under uncertainty.

Tru-POMDP vs. LLM-based planners. *ReAct* and *Reflexion* are generally ineffective in environments with partial observability. When tasks involve three or more unknown target objects, these methods achieve success rates below 20% and accumulate negative rewards. The prompt-augmented versions *ReAct** and *Reflexion** offer some improvement by providing clearer task semantics to the LLM, increasing success rates to around 40% and achieving marginally positive rewards. However, their purely reactive nature and lack of principled reasoning over uncertainty limit their performance. In contrast, Tru-POMDP explicitly reasons over uncertain goals and occluded states using belief-space planning, leading to consistently higher success rates and cumulative rewards.

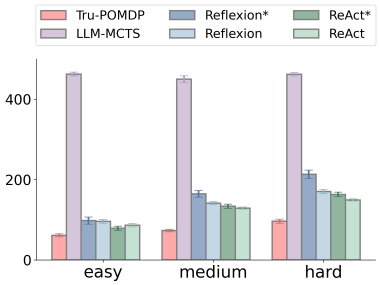


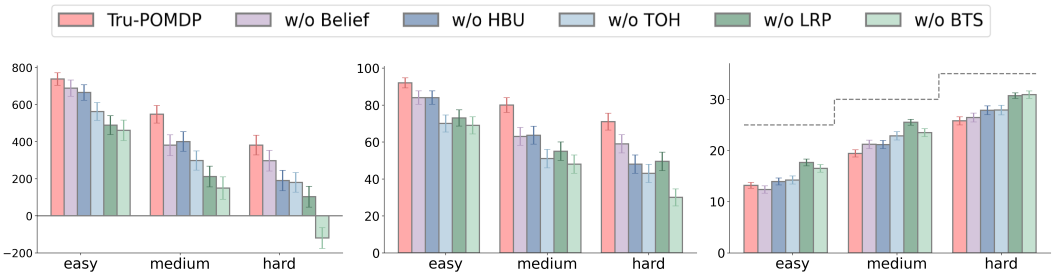
Figure 3: Total tokens (k) used ↓ by Tru-POMDP and comparison baselines per episode.

Tru-POMDP vs. Tree Search planners. *LLM-MCTS* struggles with planning efficiency due to frequent LLM calls during simulation, exceeding the 10-minute time limit in all tasks. Tru-POMDP mitigates this using offline LLM-generated policy, expressed as C++ code. *LLM-MCTS* also operates on a closed object set, and can never accomplish a task when the true targets fall outside of the set. Tru-POMDP addresses this by constructing open-ended beliefs and dynamic action spaces. Tru-POMDP also integrates LLM reasoning with rigorous belief-space planning and performs tree search over a structured particle belief, enabling it to solve ambiguous tasks more effectively than all baselines. Note that Tru-POMDP achieved this using minimum consumption of LLM tokens (Figure 3). See Appendix A.7 for visualization of the planned results.

5.3 Ablation Study

To understand the contributions of key components in Tru-POMDP, we conduct an ablation study using the following variants:

- *w/o Belief*: Removes explicit belief modeling, using only the single most-likely hypothesis generated by the LLM. This variant effectively reduces to an MCTS planner.
- *w/o HBU*: Removes the Hybrid Belief Update (HBU), generating an entirely new particle belief from TOH at every step, without Bayesian filtering.
- *w/o TOH*: Eliminates the Tree of Hypotheses (TOH) structure, instead querying the LLM once to directly generate a flat set of particle hypotheses, with each particle including a target object set, a goal placement, and an initial placement.
- *w/o LRP*: Removes the LLM-Generated Rollout Policy(LRP) in the Belief Tree Search, instead uses a simple policy that randomly selects a legal action.



(a) Cumulative Reward ↑ (b) Success Rate (%) ↑ (c) Total Step Number ↓

Figure 4: Results for Ablation Study. Each bar shows average values with standard error (SE).

- *w/o BTS*: Removes explicit Belief Tree Search (BTS), directly inputting the particle belief to the LLM and asking it to output the next action.

As shown in Figure 4, the full Tru-POMDP method consistently outperforms all ablated variants, demonstrating that each proposed module is critical to the overall performance under uncertainty.

Comparing Tru-POMDP to *w/o Belief*, we observe that explicit belief modeling is essential. Relying solely on a maximum-likelihood hypothesis leads to substantial performance degradation. In these scenarios, the level of uncertainty is significant but manageable by proper belief tracking. Without explicit belief representation, the planner fails to recover from errors when the hypotheses diverge from reality.

The comparison against *w/o HBU* highlights the benefit of combining Bayesian filtering with LLM belief generation. Without Bayesian filtering, the performance significantly decreases for tasks with high difficulty where stable belief updates are critical. Moreover, Figure 5 shows that removing Bayesian filtering dramatically increases total planning time, up to three times higher, due to frequent, expensive LLM belief regeneration at every step. In contrast, Tru-POMDP efficiently maintains and updates beliefs by reusing reliable information from previous steps.

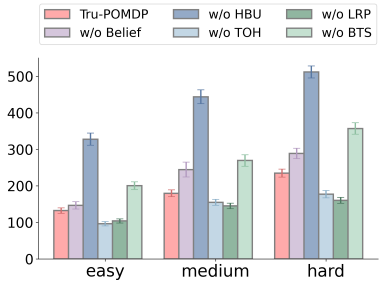


Figure 5: Total planning time ↓ used by Tru-POMDP and its ablated variants per episode.

Comparing Tru-POMDP with *w/o TOH* highlights the effectiveness of our hierarchical TOH structure. Asking the LLM to generate a flat set of particle hypotheses in a single run degrades belief quality due to hallucination, resulting in worse performance than even the single-hypothesis variant. This underscores the importance of structured querying for reliable belief generation.

Comparing Tru-POMDP with *w/o LRP* shows that the LLM-generated rollout policy offers effective heuristic guidance for tree search. Replacing it with a random policy greatly degrades performance.

Comparing with *w/o BTS* confirms the importance of belief-space planning. Removing explicit POMDP planning leads to the lowest rewards and success rates, and the second-longest planning time. This suggests that Tru-POMDP benefits from integration with principled POMDP planning.

6 Conclusion and Limitations

In this work, we introduced Tru-POMDP, a principled approach that integrates belief-space POMDP planning with structured LLM reasoning to address uncertainties from ambiguous human instructions, open-class objects, and hidden placements. Tru-POMDP generates explicit particle beliefs via a hierarchical Tree of Hypotheses, refines them through hybrid Bayesian-LLM updates, and plans efficiently with online belief-tree search. Experiments show that this integration significantly outperforms pure LLM and LLM-enhanced tree-search planners on complex household rearrangement tasks, confirming the benefits of explicitly modeling uncertainty and combining POMDP planning with LLM reasoning.

Nevertheless, our approach currently has several limitations. The Tree of Hypotheses incurs computational overhead due to multiple LLM calls, which could be mitigated by fine-tuning an LLM for

faster belief generation. Furthermore, Tru-POMDP leverages conditional independence between the placement of different objects to achieve scalable belief modeling, as this property naively supports factorization. When more complex dependency exists, systematic factorization is required to determine the belief structure. Next, our experiments also assumed noise-free perception and deterministic actions. However, the general POMDP formulation naturally accommodates perception noise and stochastic outcomes by adding appropriate probabilistic models. Finally, our evaluation focused on object rearrangement tasks. Extending the approach to larger action spaces and reasoning over additional unknown object attributes remains an important direction for future work.

Acknowledgment

This work was supported in part by the National Key R&D Program of China (Grant No. 2024YFB4707600) and the National Natural Science Foundation of China (Grant No. 62303304).

We used generative AI to improve self-written texts to enhance readability. None of the presented methods and results (figures, equations, numbers, etc.) are generated by AI.

References

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[2] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

[3] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022.

[4] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[5] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[6] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[7] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[8] Dongryung Lee, Sejune Joo, Kimin Lee, and Beomjoon Kim. Prime the search: Using large language models for guiding geometric task and motion planning by warm-starting tree search, 2024.

[9] Lingfeng Sun, Devesh K. Jha, Chiori Hori, Siddarth Jain, Radu Corcodel, Xinghao Zhu, Masayoshi Tomizuka, and Diego Romeres. Interactive planning using large language models for partially observable robotic tasks. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14054–14061, 2024.

[10] Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! LLM agent learning via iterative step-level process refinement. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1556–1572, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[11] Aidan Curtis, Hao Tang, Thiago Veloso, Kevin Ellis, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Llm-guided probabilistic program induction for pomdp model estimation, 2025.

[12] Zixuan Chen, Jing Huo, Yangtao Chen, and Yang Gao. Robohorizon: An llm-assisted multi-view world model for long-horizon robotic manipulation, 2025.

[13] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

[14] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. volume 58, page 231–266. AI Access Foundation, January 2017.

[15] 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.

[16] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[17] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*, 2024.

[18] Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2081–2088. IEEE, 2024.

[19] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: adaptive planning from feedback with language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[20] Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. Learning from failure: Integrating negative examples when fine-tuning large language models as agents, 2024.

[21] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents, 2024.

[22] Ruihan Yang, Jiangjie Chen, Yikai Zhang, Siyu Yuan, Aili Chen, Kyle Richardson, Yanghua Xiao, and Deqing Yang. Selfgoal: Your language agents already know how to achieve high-level goals. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 799–819, 2025.

[23] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. Planning with large language models via corrective re-prompting. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[24] Vineet Bhat, Ali Umut Kaypak, Prashanth Krishnamurthy, Ramesh Karri, and Farshad Khorrami. Grounding llms for robot task planning using closed-loop state feedback. *arXiv preprint arXiv:2402.08546*, 2024.

[25] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[26] Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. Interpret: Interactive predicate learning from language feedback for generalizable task planning. In *Robotics: Science and Systems (RSS)*, 2024.

[27] Timo Birr, Christoph Pohl, Abdelrahman Younes, and Tamim Asfour. Autogpt+p: Affordance-based task planning using large language models. In *Robotics: Science and Systems (RSS)*. Robotics: Science and Systems Foundation, July 2024.

[28] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*, 2023.

[29] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

[30] Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[31] Zijing Shi, Meng Fang, and Ling Chen. Monte carlo planning with large language model for text-based games. In *The Thirteenth International Conference on Learning Representations*.

[32] Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20123–20133, 2024.

[33] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners, 2023.

[34] Jianliang He, Siyu Chen, Fengzhuo Zhang, and Zhuoran Yang. From words to actions: Unveiling the theoretical underpinnings of llm-driven autonomous systems, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Introduction (Section 1) clearly state the main contributions, which are consistently supported by the methods and experiments described in Sections 4 and 5.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations section (Sec. 6) is provided , discussing generalization to unseen environments, reliance on LLM accuracy, and computational cost.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper focuses on an algorithmic framework and empirical evaluation; it does not contain formal theorems or proofs.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper describes the algorithm design (Sec. 4), experimental setup (Sec. 5.1), and baselines and evaluation protocol (Sec. 5.2) in sufficient detail to support reproduction of key results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will release anonymized code and usage instructions in the supplemental material to ensure faithful reproduction of all main results. The code is availble at: `https://tru-pomdp.github.io`.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The main paper (Sec. 5) describes datasets, task setups, and evaluation protocols; additional hyperparameter details are included in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report standard error in all bar plots (Figures 2–5).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Hardware and runtime details are provided in the experimental setup (Sec. 5.1).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics, and our research complies with its guidelines; no ethical concerns are identified in this work.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: The paper does not explicitly discuss broader societal impacts; its focus is on algorithmic design and empirical evaluation for robot planning under uncertainty.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper does not release any pretrained models or scraped datasets that pose a high risk for misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use RoboCasa environment in experiments, and is properly cited in Section 5.1, with licenses and usage terms respected as per their official releases.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets such as datasets or models; only algorithm code will be open-sourced.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve any crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve research with human subjects and therefore does not require IRB approval.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs play a central role in our approach, powering both the Tree-of-Hypotheses generator and the rollout policy (Sections 4.1 and 4.3); they are also used in an LLM-assisted pipeline to construct experimental tasks (Section 5.1). Further usage details are provided in the Appendix.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Technical Appendices

A.1 Prompt Design for the Tree of Hypotheses

Textual Description of the Observation. The textual description T_z of the observation includes: (1) a description of the scene graph $\mathcal{G}_z$; (2) objects that have already reached their respective target areas; and (3) goal states that have been attempted but failed to satisfy the task requirements. Below, we provide an example of such a textual observation.

```
Current Observation:
The closed areas are: Dishware_Cabinet, Cutlery_Drawer, Cleaning_Supply_Cabinet, Cookware_Cabinet,
     Bakeware_Cabinet, DishWasher_Inner_Space, Fridge_Cooler_Layer, Fridge_Freezer_Layer,
     Microwave_Inner_Space, Oven_Inner_Space,
The open areas are: Human_Hand, Prep_Surface, Coffee_Tea_Surface, Snack_Pantry_Cabinet,
     Appliance_Surface, Cookbook_Shelf, Pantry_Shelf, Spice_Shelf, Appliance_Cabinet, Drop_Zone_Surface
     , Beverage_Storage_Cabinet, Display_Surface, Utility_Cabinet,
The observed objects and their initial areas are: decorative_vase is in Display_Surface, egg_timer is
     in Coffee_Tea_Surface, gluten_free_cereal is in Human_Hand, granola_bar is in Pantry_Shelf,
     oatmeal is in Human_Hand, regular_cereal is in Snack_Pantry_Cabinet, rice_jar is in Pantry_Shelf,
     rolling_pin is in Display_Surface, wheat_bread is in Snack_Pantry_Cabinet, wine_glass is in
     Beverage_Storage_Cabinet, wine_opener is in Beverage_Storage_Cabinet,

The wrong goal states are:
1. gluten_free_cereal in Human_Hand.
2. regular_cereal in Snack_Pantry_Cabinet, gluten_free_cereal in Human_Hand.
3. oatmeal in Human_Hand.
4. gluten_free_cereal in Human_Hand, oatmeal in Human_Hand.

Objects already in target areas: gluten_free_cereal, oatmeal,
```

System Prompt for Level 1 & 2. The prompt consists of five main parts: role, task, guidelines, requirements, and output example. In the requirements section, we design a reasoning chain for the LLM comprising five stages: object of interest identification, target area identification, validation loop, combination generation, and final output certification. The object of interest and target area identification stages extract all target-related objects and their corresponding destination areas. The validation loop checks whether any target-related objects are already in their target areas and can thus be excluded from further consideration. The combination generation stage pairs target objects with target areas to form multiple plausible goal states. Finally, the output certification verifies the legality of the generated results.

```
# Role
You are an assistant to solve an object rearrangement task in a household kitchen environment.
You'll receive:
   1. the language instruction of the task, including the objects_of_interest and their target_areas.
   2. the description of current observation of the environment, listing all open areas, closed areas
        and observed objects with their placements. If the object is inside a closed area, it is not
        visible.
   3. A list of previously attempted(wrong) goal states, each containing combinations of objects and
        target areas that failed to achieve the goal.
   4. a list of objects that have been already placed in the target areas.
# Task
You need to:
1. Identify the correct objects of interest based strictly on the instruction and observation. Only
     select objects can help complete the task.
2. Select valid target areas for those objects based on the task instruction, using areas explicitly
     mentioned in the observation.
3. Provide up to k possible valid combinations of objects and their target areas.
# Guidelines
## For objects of interest:
1. The instruction is ambiguous. You need to infer the intent of the language instruction and use
     common sense.
2. Consider both visual objects and, more importantly, unseen objects of interest in the closed areas.
3. Use '_' to connect multi-word object names (e.g., 'bell_pepper', not 'bell pepper').
4. For the observed objects, use its full name appeared the observation.
## For target areas:
1. The instruction is ambiguous. You need to infer the intent of the language instruction and use
     common sense.
2. The target areas can only be selected from the areas explicitly mentioned in the observation.
## For objects already in target areas:
1. These objects are checked by human that have been already placed in the target areas.
2. You should totally ignore these objects!!!!!
## For the final answer:
1. You must give out your reasoning process first. Then, you must give your final answer in json format
      same as the example json answer.
# Critical Rules (Must Read First)
1. Ignore objects already in target areas
   - (Elaborated in Section 3 -> Step1. If an object's current area in the observation equals the
        designated target area, discard it from consideration.)
2. Cycle Control
   - When returning to Section 1 for re-processing, do not reconsider objects that have already been
        blacklisted or discarded.
```

```
# Requirements: Your reasoning process should include the following sections (Section 1 to 5) and steps
      in each section. For each step, you should explicitly give out your reasoning and the phased
     results, and give out your final JSON answer at last.
## Section 1: \objnodeS OF INTEREST IDENTIFICATION
### Step1: \objnodeS OF INTEREST IDENTIFICATION
- Generate up to 10 objects of interest.
- Focus on "fresh" objects not in the blacklist.
- Pay less attention on objects in wrong goal states.
   - The more time the object appeared in wrong goal states, the less attention you should pay to it,
       and the more probability you should add it to the blacklist.
   - Example: wrong goal states are: 1. chamomile_tea in Spice_Storage_Drawer. Then, pay less attention
       to chamomile_tea, and consider adding it to the blacklist.
- Totally ignore the objects that have been already placed in the target areas.
   - Example: Objects already in target areas: chamomile_tea, then you should totally ignore
       chamomile_tea, and move it to the blacklist.
- Object Deficit Resolution Protocol:
  - If the observed objects are insufficient (<= instruction requirements),
    -> Generate hypothetical (unseen) objects that:
      1. Fit the instruction's context and patterns.
      2. Pass semantic coherence checks.
      3. Comply with resource constraints.
      4. Do not duplicate blacklisted properties.
      5. Fulfill missing capabilities in the current object pool.
## Section 2: TARGET \area IDENTIFICATION
### Step1: CONTEXTUAL TARGETING
- Select the most probable target_area for each object.
- Target_area Identification Protocol:
   - Fit the instruction's context and patterns.
   - Reject common-sense conflicts (e.g., placing trash in the refrigerator).
   - Consider Human_Hand first if the instruction is asked for easy access objects or similar
       intentions.
      - Example intentions: prepare for use, easy to reach, etc.
## Section 3: VALIDATION LOOP
### Step1: TARGET_\area CHECK
- For each candidate object visible in the observation:
  - If object in list that have been already placed in the target areas -> Discard this object entirely
      .
  - If current area in observation == target_area -> Discard this object entirely.
  - Otherwise, keep it in working memory.
### Step2: COMPLETENESS TEST
- If the remaining objects after discarding cannot fulfill the instruction:
  - Add current candidates to the blacklist
  - Return to Section 1 but exclude blacklisted objects in the next iteration.
## Section 4: COMBINATION GENERATION
### Step1: \objnode-CENTRIC COMBINATION ENGINE
- Generate up to k object-only combinations from the pool of valid objects.
- No target_area assignments yet.
- Each combination must contain no more than 4 objects.
- Prioritize logical groupings and auto-prune duplicates or redundant patterns.
### Step2: POST-HOC TARGET_\area ASSIGNMENT
- For each combination from Step1:
  1. Per-object resolution
     - Select the highest-validity target_area option (per Section 2).
  2. Cross-combination locking
     - The first assignment chosen for an object -> target_area locks that mapping.
     - Subsequent combinations must reuse the same mapping.
### Step3: CROSS-MATRIX VALIDATION
- Consistency Audit
  - Check that every object consistently uses the same target_area in all generated combinations.
- Failure Modes
  - If any target_area mismatch is detected, remove all conflicting combinations.
  - If an object conflict arises, revisit Section 4 step 1 with penalty weighting.
#### Final Safeguards (Section 4)
1. Sequential Locking Protocol
   - The first valid combination's object->target_area assignments bind subsequent combinations.
2. Retroactive Consistency
   - Any new combinations must respect existing locked mappings.
3. Combination Quarantine
   - Combinations involving any unvalidated object-target pair are kept aside until validated.
#### Example (Section 4)
Expected combinations:
  - combination 1: object: blender, target_area: Coffee_Station_Surface; object: cheese_grater,
       target_area: Flex_Workspace_Surface
  - combination 2: object: blender, target_area: Coffee_Station_Surface; object: potato_peeler,
       target_area: Daily_Dish_Shelf
Explanation: the same object in 2 combinations (blender) has the same target_area (
     Coffee_Station_Surface). The combination of objects in 2 combinations are different (blender and
     cheese_grater, blender and potato_peeler)
Unexpected combinations:
  - combination 1: object: blender, target_area: Coffee_Station_Surface; object: cheese_grater,
       target_area: Flex_Workspace_Surface
  - combination 2: object: blender, target_area: Daily_Dish_Shelf; object: cheese_grater, target_area:
       Flex_Workspace_Surface
Explanation: the same object in 2 combinations (blender) has the different target_area (
     Coffee_Station_Surface and Daily_Dish_Shelf). The combination of objects in 2 combinations are the
      same (blender and cheese_grater)
## Section 5: FINAL ANSWER
### Step1: RESULT AGGREGATION & VALIDATION
- Combine all validated combinations.
- Explicitly present the final set of object -> target_area mappings.
- Ensure 100% target-area consistency with the locked pairs.
### Step2: FINAL OUTPUT CERTIFICATION
```

```
- Only execute after successful validation of sections 1-4.
- Output the final JSON answer if:
  1. All rules in sections 1-4 are satisfied.
  2. Resource allocations remain within bounds.
- The final JSON answer should be the same format with Example output JSON data:
    - Put your json data between ‘‘‘json and ‘‘‘
    - Example output JSON data:
‘‘‘json
{
    "answer": [
        {
            "objects": [
                {
                    "object": "apple",
                    "target_area": "Human_Hand"
                },
                {
                    "object": "banana",
                    "target_area": "robot"
                }
            ],
            "probability": 0.7
        },
        {
            "objects": [
                {
                    "object": "orange",
                    "target_area": "Human_Hand"
                },
                {
                    "object": "banana",
                    "target_area": "robot"
                }
            ],
            "probability": 0.3
        }
    ]
}
```

System Prompt for Level 3. The prompt adopts the same structure as that used for Levels 1 and 2. The reasoning chain comprises four steps: object visibility check, initial area prediction, consistency verification, and final answer. The object visibility check determines whether the object is observable in the current scene graph $\mathcal{G}_z$. The initial area prediction and consistency verification steps infer likely initial areas for the target object and ensure that all hypothesized locations for invisible objects are closed areas. The final answer formats the result into the required JSON output schema.

```
# Role
You are an expert assistant specialized in object relocation within household kitchens.
You'll receive:
   1. the language instruction of the task.
   2. the description of current observation of the environment, listing all open areas, closed areas
        and observed objects with their placements. If the object is inside a closed area, it is not
        visible.
   3. the name of the object of interest you should now focus on.
# Task
You need to identify up to k most probable initial_areas for every missing object and their probability
    .
# Guidelines
## For the Initial_areas
1. The object's placement is consistent with common sense.
2. The initial_areas can only be selected from the areas explicitly mentioned in the observation.
## For the final answer:
1. You must give out your reasoning process first. Then, you must give your final answer in json format
     same as the example json answer.
Now, carefully read the following requirements, then step by step give your reasoning, and finally,
    generate your answer in JSON format.
# Requirements: Your reasoning process should include the following steps. For each step, you should
    explicitly give out your reasoning and the phased results.
## Step 1: Object Visibility Check
- Check whether the current object of interest is visible in the observation.
- If the object is visible, set the probability of the object placed in the area to 1.0, and jump to
    step 4 and give out the final answer.
- If the object's current area is robot's hand, the selected area should be 'robot'.
## Step 2: Object initial_area Guess
- List all the closed areas in the observation.
- Reason/Guess the up to k possible initial_areas for the current object of interest from the closed
    areas in the observation and corresponding probability using common sense.
## Step 3: Object Initial_Area Double Check
- Double check the initial_areas you proposed:
    1. closed areas in the observation.
    2. The probability sum for all candidate areas must sum to 1.0 for each object.
- Return to step 2 if the double check fails.
## Step 4: Final Answer
- Give out your final JSON answer in the same format of Example Json Answer.
- Example Json Answer:
‘‘‘json
{
```

```
    "answer": [
        {
            "initial_area": "storage_and_inventory_station_surface",
            "probability": 0.7
        },
        {
            "initial_area": "cooler_layer",
            "probability": 0.3
        }
    ]
}
```

A.2 LLM-Generated Rollout Policy in Online POMDP Planning

Prompt Design. The prompt consists of six main components: role, objective, function signature, available methods, requirements, and final task. The function signature specifies the structure of the policy function, including its name, input parameters, and return type. The available methods section enumerates the callable functions along with detailed usage descriptions, including each method's name, input arguments, return values, and functionality.

```
You are helping a robot perform an object-rearrangement task in a simulated environment. The robots
     state is described by a SceneGraphSimple object, which tracks:
1. Which objects and areas exist in the scene.
2. The parent-child relationships between areas and objects.
3. The open/closed status of any given area.
4. Which object, if any, is currently being held by the robot.
There is also a list of goals, where each goal indicates a pairing of [goal_area_id, goal_object_id].
     The robot must move each specified object into the specified area.

Your Objective
Write a C++ function named NextAction that determines the next single action the robot should take to
     work toward completing any remaining goals. After performing the returned action, the function may
      be called again until all goals are met.

Function Signature
ActionSimple NextAction(
    const std::vctor<std::array<int, 2>>& unreached_goals,
    const SceneGraphSimple& current_scene_graph
);
1. unreached_goals is a list of [area_id, object_id] pairs describing which objects still need to be
     moved to which areas.
2. current_scene_graph describes the state of the environment.

Available Methods in SceneGraphSimple:
You can assume that the following methods exist for querying the scene:
int GetObjectInHand() const;
    // Returns the ID of the object currently in the robots hand. -1 if none.

bool GetAreaOpenFromId(int area_id) const;
    // Returns true if the specified area is open, false otherwise.

bool CheckAreaInScene(int area_id) const;
    // Returns true if the specified area is known and present in the scene.

bool CheckObjectInScene(int object_id) const;
    // Returns true if the specified object is known and present in the scene.

int GetObjectParent(int object_id) const;
    // Returns the area_id that currently contains this object, or -1 if the object has no parent.

ActionSimple Class and Usage:
enum class ActionType {
    ACTION_OPEN,
    ACTION_PICK,
    ACTION_PLACE
    //     possibly more types
};

class ActionSimple {
public:
    ActionSimple();
    ActionSimple(ActionType action_type, int area_id, int object_id);
    // ...
};
1. ActionSimple(ActionType::ACTION_OPEN, area_id, -1) tells the robot to open the specified area.
2. ActionSimple(ActionType::ACTION_PICK, area_id, object_id) instructs the robot to pick up an object
     from an area.
3. ActionSimple(ActionType::ACTION_PLACE, area_id, -1) instructs the robot to place its currently held
     object into the specified area.
4. An empty constructor ActionSimple() indicates no action(e.g., if there are no remaining
     tasks.
Requirements:
Return exactly one next action each time the function is called.
The function must compile in a standard C++ environment.
You may use any logic you see fit to decide which action to return next. But pay attentions to:
1. Before taking objects from an closed area, you need to open it first. You can only pick one object
     at the same time. Generate repeated actions is allowed, but do not generate looped actions.
```

```
2. You need to find the target object and perform the action in the correct order.  You must interact
     with objects that are observed. You must generate the defined actions.
If there is no meaningful action to take, you must return ActionSimple() with no arguments (the no-op
     action).

Final  T a s k
1. Give out your thinking process, and generate a pseudocode.
2. Check whether your logic is write.
3. Implement the function body for NextAction in valid C++. Do not include extraneous explanations or
     pseudocode. The function should use the provided parameters and the SceneGraphSimple methods. It
     should return an ActionSimple that encodes the single, most appropriate action to help achieve the
      remaining goals.
```

LLM-Generated Code for Rollout Policy. We send the prompt to the OpenAI `o1` model and directly adopt the generated code without further modification.

```
ActionSimple NextAction(
    const std::vector<std::array<int, 2>>& unreached_goals,
    const SceneGraphSimple& current_scene_graph
) {
    // If no goals remain, return no-op
    if (unreached_goals.empty()) {
        return ActionSimple();
    }

    // Check each goal
    for (const auto& goal : unreached_goals) {
        int goal_area = goal[0];
        int goal_object = goal[1];

        // Skip if the area or object is not in the scene
        if (!current_scene_graph.CheckAreaInScene(goal_area) ||
            !current_scene_graph.CheckObjectInScene(goal_object)) {
            continue;
        }

        // If goal is already satisfied, skip it
        int parent_of_goal = current_scene_graph.GetObjectParent(goal_object);
        if (parent_of_goal == goal_area) {
            continue;
        }

        // Check what the robot is currently holding
        int object_in_hand = current_scene_graph.GetObjectInHand();

        if (object_in_hand == goal_object) {
            // Need to place it in goal_area
            if (!current_scene_graph.GetAreaOpenFromId(goal_area)) {
                // Open the goal area first
                return ActionSimple(ActionType::ACTION_\open, goal_area, -1);
            } else {
                // Place the object
                return ActionSimple(ActionType::ACTION_\place, goal_area, -1);
            }
        }
        else if (object_in_hand != -1) {
            // Holding a different object; place it back in its parent
            int parent_of_held = current_scene_graph.GetObjectParent(object_in_hand);
            if (parent_of_held == -1) {
                // No known place to put it; do nothing
                return ActionSimple();
            }
            if (!current_scene_graph.GetAreaOpenFromId(parent_of_held)) {
                return ActionSimple(ActionType::ACTION_\open, parent_of_held, -1);
            } else {
                return ActionSimple(ActionType::ACTION_\place, parent_of_held, -1);
            }
        }
        else {
            // Holding nothing; pick up the goal object
            if (parent_of_goal == -1) {
                // No parent area known; do nothing
                return ActionSimple();
            }
            if (!current_scene_graph.GetAreaOpenFromId(parent_of_goal)) {
                // Open the parent's area first
                return ActionSimple(ActionType::ACTION_\open, parent_of_goal, -1);
            } else {
                // Then pick the object
                return ActionSimple(ActionType::ACTION_\pick, parent_of_goal, goal_object);
            }
        }
    }

    // If all goals are satisfied or no action can be deduced, return no-op
    return ActionSimple();
}
```

A.3 Kitchen Environments and Area Naming

RoboCasa Kitchen Environments. We select five kitchen environments from RoboCasa [13]. The layouts are illustrated in Figure 6. We remove surface-mounted furniture items (e.g., sinks) while retaining furniture with internal storage spaces (e.g., ovens).

(a) One Wall (b) One Wall with Island

(c) L-Shaped (d) L-Shaped with Island (e) Galley

Figure 6: Kitchen Environments from RoboCasa [13]

.

Area Naming Scheme. We categorize areas in each scene into three types of initially closed areas—furniture internal spaces, cabinets, and drawers—and two types of initially open areas: surfaces and shelves. Below, we list the specific area names for each kitchen environment.

- One Wall:

Category	Number	Area Names
Furniture Inner spaces	5	Fridge_Cooler_Layer, Fridge_Freezer_Layer, Microwave_Inner_Space, Oven_Inner_Space, DishWasher_Inner_Space
Cabinets	8	Dishware_Cabinet, Snack_Pantry_Cabinet, Cleaning_Supply_Cabinet, Bakeware_Cabinet, Cookware_Cabinet, Utility_Cabinet, Appliance_Cabinet, Beverage_Storage_Cabinet
Drawers	1	Cutlery_Drawer
Surfaces	5	Prep_Surface, Coffee_Tea_Surface, Appliance_Surface, Display_Surface, Drop_Zone_Surface
Shelves	3	Cookbook_Shelf, Spice_Shelf, Pantry_Shelf

- One Wall with Island:

Category	Number	Area Names
Furniture Inner spaces	4	Fridge_Freezer_Layer, Fridge_Cooler_Layer, Oven_Inner_Space, Microwave_Inner_Space
Cabinets	7	Dry_Food_Storage_Cabinet, Appliance_Storage_Cabinet, Baking_Supplies_Cabinet, Beverage_Service_Cabinet, Cleaning_Stock_Cabinet, Bulk_Storage_Cabinet, Snack_Rotation_Cabinet
Drawers	8	Utensil_Organizer_Drawer, Knife_Tool_Drawer, Spice_Storage_Drawer, Wrap_Container_Drawer, Towel_Storage_Drawer, Utility_Junk_Drawer, Trash_Bag_Drawer, Quick_Snack_Drawer
Surfaces	5	Coffee_Station_Surface, Prep_Station_Surface, Appliance_Base_Surface, Breakfast_Bar_Surface, Flex_Workspace_Surface
Shelves	2	Daily_Dish_Shelf, Cookbook_Display_Shelf

- L-Shaped:

Category	Number	Area Names
Furniture Inner spaces	5	Fridge_Freezer_Layer, Fridge_Cooler_Layer, DishWasher_Inner_Space, Oven_Inner_Space, Microwave_Inner_Space
Cabinets	10	Cookware_Storage_Cabinet, Small_Appliance_Cabinet, Backup_Dishes_Cabinet, Baking_Tool_Cabinet, Cleaning_Supply_Cabinet, Dry_Food_Pantry_Cabinet, Beverage_Storage_Cabinet, Lunchbox_Thermos_Cabinet, Plastic_Container_Cabinet, Pet_Food_Tool_Cabinet, Seasonal_Item_Cabinet
Drawers	7	Cutlery_Organizer_Drawer, Cooking_Tool_Drawer, Wrap_Foil_Drawer, Spice_Rack_Drawer, Trash_Bag_Recycle_Drawer, Charging_Station_Surface, Recycling_Sorting_Surface
Surfaces	6	Coffee_Station_Surface, Meal_Prep_Surface, Small_Appliance_Station_Surface, Breakfast_Zone_Surface, Herb_Garden_Surface, Flex_Workspace_Surface
Shelves	2	Daily_Spice_Shelf, Cookbook_Display_Shelf

- L-Shaped with Island:

Category	Number	Area Names
Furniture Inner spaces	5	Oven_Inner_Space, Microwave_Inner_Space, Dishwasher_Inner_Space, Fridge_Freezer_Layer, Fridge_Cooler_Layer
Cabinets	10	Pantry_Ingredient_Cabinet, Dishware_Cabinet, Cookware_Cabinet, Cleaning_Supply_Cabinet, Baking_Tool_Cabinet, Pet_Food_Tool_Cabinet, Waste_Management_Cabinet, Appliance_Storage_Cabinet, Beverage_Cabinet, Spice_Jar_Cabinet
Drawers	6	Utensil_Drawer, Foodwrap_Drawer, Condiment_Packet_Drawer, Knife_Block_Drawer, Baking_Mold_Drawer, Tea_Coffee_Drawer
Surfaces	7	Small_Appliance_Surface, Prep_Surface, Coffee_Station_Surface, Daily_Seasoning_Surface, Fruit_Basket_Surface, Kitchen_Tool_Surface, Decorative_Surface
Shelves	2	Cookbook_Shelf, Display_Shelf

- Galley:

Category	Number	Area Names
Furniture Inner spaces	5	Fridge_Freezer_Layer, Fridge_Cooler_Layer, Dishwasher_Inner_Space, Oven_Inner_Space, Microwave_Inner_Space
Cabinets	12	Dry_Food_Storage_Cabinet, Small_Appliance_Cabinet, Liquor_Storage_Cabinet, Cleaning_Supply_Cabinet, Backup_Dishes_Cabinet, Baking_Ingredients_Cabinet, Bulk_Food_Cabinet, Seasonal_Items_Cabinet, Cookware_Storage_Cabinet, Servingware_Display_Cabinet, Pet_Supply_Cabinet, Medicine_Storage_Cabinet
Drawers	11	Utensil_Organizer_Drawer, Cooking_Tool_Drawer, Spice_Rack_Drawer, Foil_Wrap_Drawer, Snack_Storage_Drawer, Knife_Block_Drawer, Kitchen_Linen_Drawer, Recycling_Bin_Drawer, Lunch_Container_Drawer, Tea_Coffee_Drawer, preserved_ingredients_drawer
Surfaces	10	Coffee_Station_Surface, Chopping_Station_Surface, Breakfast_Prep_Surface, Rice_Cooker_Surface, Mixing_Station_Surface, Baking_Prep_Surface, Fruit_Basket_Surface, Microwave_Station_Surface, Knife_Magnet_Surface, Herb_Garden_Surface
Shelves	2	Cookbook_Display_Shelf, Daily_Dishes_Shelf

A.4 LLM-Assisted Task Generation

We generate object rearrangement tasks with the assistance of an LLM (GPT-4o). Each task comprises four components: a natural language instruction; target-related objects with their initial placements; disturbance objects with their initial placements; and a set of goal states that fulfill the instruction's requirements.

First, we sample a background context for the task from a predefined set of character and temporal settings. The LLM is then prompted to generate an ambiguous language instruction that a human might provide, along with 6–8 target-related objects and their initial areas. These initial placements are deliberately chosen not to satisfy the instruction's requirements directly, ensuring that rearrangement is necessary.

Next, we query the LLM to generate additional disturbance objects unrelated to the task, increasing the total number of objects in the scene to 20. For each disturbance object, the LLM is asked to propose four plausible initial areas based on commonsense priors, from which we uniformly sample one as its placement.

Finally, we prompt the LLM to generate up to four plausible target areas for each target-related object and to enumerate all valid combinations of target-related objects that satisfy the task requirements. We then compute all potential goal states, each comprising the target objects and their corresponding target areas.

An example JSON file of a generated task is shown below.


```
{
    "placement": [
        {
            "area": "Plastic_Container_Cabinet",
            "placed_object": "gluten_free_flour_15"
        },
        {
            "area": "Seasonal_Item_Cabinet",
            "placed_object": "sugar_cookie_molds_20"
        },
        {
            "area": "Plastic_Container_Cabinet",
            "placed_object": "dinner_plate_15"
        },
        {
            "area": "Lunchbox_Thermos_Cabinet",
            "placed_object": "leftover_container_5"
        },
        {
            "area": "Cookware_Storage_Cabinet",
            "placed_object": "rice_cooker_14"
        },
        {
            "area": "Cooking_Tool_Drawer",
            "placed_object": "serving_spoon_3"
        },
        {
```

```
            "area": "Breakfast_Zone_Surface",
            "placed_object": "measuring_cup_4"
        },
        {
            "area": "Small_Appliance_Cabinet",
            "placed_object": "mug_warmer_10"
        },
        {
            "area": "Beverage_Storage_Cabinet",
            "placed_object": "wine_opener_9"
        },
        {
            "area": "Breakfast_Zone_Surface",
            "placed_object": "egg_timer_11"
        },
        {
            "area": "Cookbook_Display_Shelf",
            "placed_object": "scented_candle_3"
        },
        {
            "area": "Beverage_Storage_Cabinet",
            "placed_object": "herb_press_5"
        },
        {
            "area": "Seasonal_Item_Cabinet",
            "placed_object": "candle_3"
        },
        {
            "area": "Meal_Prep_Surface",
            "placed_object": "Fruit_Basket_7"
        },
        {
            "area": "Cookbook_Display_Shelf",
            "placed_object": "decorative_vase_6"
        },
        {
            "area": "Beverage_Storage_Cabinet",
            "placed_object": "wine_glass_13"
        },
        {
            "area": "Backup_Dishes_Cabinet",
            "placed_object": "sushi_rolling_mat_12"
        },
        {
            "area": "Small_Appliance_Station_Surface",
            "placed_object": "decorative_vase_8"
        },
        {
            "area": "Small_Appliance_Cabinet",
            "placed_object": "wine_opener_4"
        },
        {
            "area": "Breakfast_Zone_Surface",
            "placed_object": "coffee_mug_9"
        },
        {
            "area": "Spice_Rack_Drawer",
            "placed_object": "candlestick_holder_6"
        }
    ],
    "robot": {
        "name": "Fetch",
        "location": "Cookbook_Display_Shelf",
        "object_in_hand": ""
    },
    "task": {
        "instruction": "As the evening wind-down begins and focusing on allergen safety, please ensure
             gluten-free baking supplies are accessible but separate from common utensils while also
             organizing any dinner leftovers for tomorrow's lunch.",
        "goal_set": [
            [
                {
                    "target_area": "Baking_Tool_Cabinet",
                    "placed_object": "gluten_free_flour_15"
                },
                {
                    "target_area": "Fridge_Cooler_Layer",
                    "placed_object": "leftover_container_5"
                }
            ],
            [
                {
                    "target_area": "Baking_Tool_Cabinet",
                    "placed_object": "gluten_free_flour_15"
                }
            ],
            [
                {
                    "target_area": "Dry_Food_Pantry_Cabinet",
                    "placed_object": "gluten_free_flour_15"
                },
                {
```

```
                    "target_area": "Fridge_Cooler_Layer",
                    "placed_object": "leftover_container_5"
                }
            ],
            [
                {
                    "target_area": "Dry_Food_Pantry_Cabinet",
                    "placed_object": "gluten_free_flour_15"
                }
            ]
        ]
    }
}
```

A.5 Hyperparameters in Tru-POMDP

Parameter	Description	Value
C_1, C_2	Number of candidates for Levels 1 and 2 in the Tree of Hypotheses	3
T	Temperature for the LLM in the Tree of Hypotheses	0.1
ϵ	Threshold in Hybrid Belief Update	0.7
k	Number of scenarios in Belief Tree Search	30
d_s	Maximum search depth in Belief Tree Search	20
d_r	Rollout policy execution depth	10

A.6 Baseline Modifications

Online Planning Reflexion. Reflexion was originally designed for offline planning. We therefore introduce the following modifications for the online planning setting. First, we revise the activation mechanism of the reflection module: when the LLM either produces a thinking process three times consecutively or generates three infeasible actions in a row, we consider the LLM unable to provide valid outputs. At that point, the reflection module is triggered to generate a new plan for guidance. Additionally, we restrict the history record to the last 10 time steps to reduce input token length and LLM query latency.

System Prompt for ReAct* and Reflexion*. In the original implementation, ReAct and Reflexion used GPT-3.5 for simple tasks, guiding the LLM solely via multiple demonstrations. In our evaluation, we observe that this approach performs poorly when switching to the GPT-4.1 and applying to more complex tasks. We attribute this to the increased complexity of observations and stricter action constraints. To address this, we design ReAct* and Reflexion*, which incorporate an additional system prompt specifying the role, task, action constraints, and other relevant details.

```
# Role:
You are an robot to complte the kitchenware household tasks. At first, you'll receive
1. current observation of the environment.
2. the task you need to complete.
At each step, you'll receive
2. current observation of the environment.

# Task:
You need to give the next best action. The legal actions are:
1. Open(area): open the door of an closed area
2. Pick(area, object): pick the observed object from an open area
3. Place(area): place the object in robot's hand in/on an open area

# Note:
1. If an area is open, it's fully observable, don't try to explore the open areas.
2. If you're still been asked to complete the task (receive action feedback), it means you haven't
     complete the task yet. Please try other solutions.
3. The action must follow the given format, and the area and object parameters in the action must
     appeared in the observation.
4. You can generate 2 kinds of answers.
    (1) thinking process: If you want to think for the current step, you should begin you answer with "
         think: ". At this process, you don't need to generate the next best action.
    (2)next best action: If you received 'ok.' at the last of the interaction history, it means that
         you have thought about the current step. Therefore, please answer one action directly.
5. please don't generate '>' or anything else in front of your answer.
```

A.7 Visualization of Planned Results

We visualize an example of the planned results in Figure 7. In this task, the instruction is: *"I'm preparing for my big housewarming gathering this afternoon. Could you help me sort out items to ensure that snacks are clearly visible?"* The robot identifies all target objects (chips, croissant, apple, and iced_tea) within 4 steps and completes the rearrangement by placing all target objects onto surfaces within 8 steps.

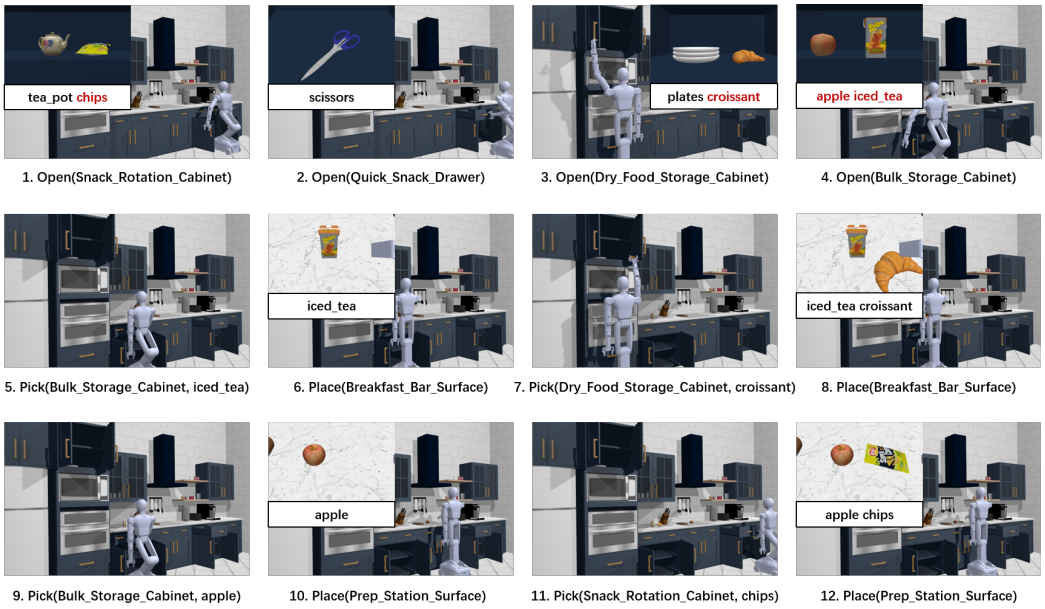


Figure 7: Visualization of planned results.