10 Notations

10.1 Notation Glossary

- B_t: Belief state at timestep t, composed of $(B_t^{\mathcal{G}}, b_t^{\mathcal{S}})$
- $a \in \mathcal{A}_t$: Candidate symbolic action (e.g., "go to shelf") at time t
- $\hat{o}$: Predicted observation under action a, sampled from $P(o \mid a, \mathcal{B}_t)$
- $u(B_t, \hat{o})$: Utility of the predicted observation $\hat{o}$ when incorporated into the belief state
- IG_a: Information gain induced by taking action a and observing $\hat{o}$
- Align_a: Alignment reward computed by comparing $\hat{o}$ to actual environment observation
- λ: Weighting factor to trade off information utility and alignment score
- $H(\cdot)$: Shannon entropy of a probability distribution
- $\mathcal{G}$: Global belief consisting of high-level language hypotheses
- $\mathcal{S}_s$: Subject-level belief distribution over locations for target s
- $\text{sim}(m, h)$: Similarity function between parsed object mention m and hypothesis h
- $\text{symb}(a)$: Symbolic keyword extracted from action a (e.g., "cabinet" from "go to cabinet 2")
- π_{pred}: Observation prediction model
- π_{BU}^{g}, π_{BU}^{s}: Belief update model for global(g) and object(s)-level beliefs

10.2 A Toy ALFWorld MDP Example

For concreteness, we provide a minimal tabular instance. Consider a house with rooms $R_1, \ldots, R_5$, where an apple is initially in R_3 and the agent starts in R_1. The task is to "place the apple in R_5." We encode the state as

$$s_t = (r_t^{\text{agent}}, r_t^{\text{apple}}, h_t, z_t),$$

where $r_t^{\text{agent}} \in \{R_1, \ldots, R_5\}$ is the agent's room, $r_t^{\text{apple}} \in \{R_1, \ldots, R_5\}$ is the apple's room, $h_t \in \{0, 1\}$ indicates whether the agent is holding the apple, and $z_t \in \{0, 1\}$ indicates whether the apple has been placed in R_5. The action set is

$$\mathcal{A} = \{\textsc{GoTo}(R_i),\ \textsc{PickApple},\ \textsc{PlaceAppleInR5}\}.$$

For example, from $s_0 = (R_1, R_3, 0, 0)$, executing $\textsc{GoTo}(R_3)$ updates the agent room but leaves the other components unchanged; executing PICKAPPLE then yields $(R_3, R_3, 1, 0)$; after moving to R_5 and executing PLACEAPPLEINR5, we reach a terminal state with $z_t = 1$. The LLM policy does not observe s_t directly, but instead receives textual descriptions of these states and transitions, and chooses actions accordingly.

11 Motivation: Experimental Detail

Objective. This experiment evaluates whether SFT agents reuse train-time search behaviors at test time and whether such behavior leads to successful task completion. We analyze both the degree of behavioral copying and its effectiveness across environments that differ from training.

Setup. We define two evaluation sets:

- **Seen Set (Same Room)**: Rooms and object locations identical to training.
- **Unseen Set (Similar Room)**: Rooms with similar layouts but different object placements.

Each trajectory begins with a natural language room description followed by a sequence of actions. From each trajectory, we extract a **search sequence**, defined as a concatenation of all consecutive chains of `go to` actions. We then compute:

- **Room similarity**: Cosine similarity between object-count vectors, parsed from textual room inventory from test-time room descriptions the test-time room description.
- **Trajectory similarity**: Fraction of the training search sequence covered by the test sequence after stripping instance indices from object name.

To focus on cases where rooms are similar but not identical, we retain only test trajectories whose best room match falls within a high-similarity band $\delta_{\text{room}}^{\text{low}} \leq s_{\text{room}} \leq \delta_{\text{room}}^{\text{high}}$, where $\delta_{\text{room}}^{high} = 0.935$ and $\delta_{\text{room}}^{high} = 0.965$

Case Classification. We categorize each retained test trajectory based on (1) its trajectory similarity and (2) success outcome:

- **Train-like → Success**: Trajectory similarity $\geq \delta_{traj}$, task succeeded.
- **Train-like → Failure**: Trajectory similarity $\geq \delta_{traj}$, task failed.
- **Deviated → Success**: Trajectory similarity $< \delta_{traj}$, task succeeded.
- **Deviated → Failure**: Trajectory similarity $< \delta_{traj}$, task failed.

We set $\delta_{traj} = 0.33$ by default.

12 Related Works

Inference-time baselines Compared to existing inference-time agents, AWS explicitly maintains and updates a task-specific belief posterior and chooses actions by their expected information gain in this belief space, rather than by self-consistency or proxy scores. **Reflexion** [37] improves behavior across episodes by strong natural language reflections, but leaves first-episode, fixed-budget performance largely unchanged; AWS directly reduces cold-start errors within the same episode via posterior-guided exploratory

steps. **RAP** [13] and **LAC** [8] score candidates using actor log-probabilities or a learned critic without an explicit posterior, so they reinforce internally confident behaviors even when they are misaligned with the partially observable environment. **LATS** [53] assumes reversible simulators and accurate model-based MCTS rollouts, where assumptions violated in realistic interactive environments and badly mismatched when LLM rollouts hallucinate dynamics, whereas AWS performs one-shot forward execution with no rollback and updates belief only from action-conditioned real observations. **RAFA** [27] and **QuBE** [21] introduce richer reasoning signals (subgoal progress, QA-based belief states), but still ground decisions in heuristic progress/textual proxies without modeling calibrated uncertainty or information gain; AWS instead couples global language hypotheses with numeric belief scores to quantify uncertainty, reliability and IG jointly.

Self-search baselines Comparing to self-search baselines such as **Intuitor** [50]**/RENT** [30]**/SSRL** [10], which optimize internal confidence or search depth, AWS ties "confidence" to evidence-backed belief updates driven by observed outcomes, making its action scores calibrated by world feedback rather than by an internal entropy heuristic.

Train-time baselines Unlike AWS, which keeps the backbone LLM frozen and optimizes behavior purely at inference time via belief-guided exploratory steps, existing train-time methods reshape the policy itself using additional trajectories or supervision. **ETO** [40] contrasts failure and success trajectories with DPO-style objectives to update the agent, relying on large offline data rather than within-episode belief updates. **WKM** [32] distills expert demonstrations into structured task- and state-level knowledge that conditions action selection at test time (we reproduce WKM under our environment and LLaMA 3.1 setting), whereas AWS forms task-specific beliefs on the fly from observed outcomes. **MPO** [45] learns abstract meta-plans that are iteratively refined by feedback but does not maintain an explicit uncertainty-aware posterior, while AWS directly scores concrete actions by their expected information gain in belief space. **IPR** [44] and **STeCa** [43] perform step-level refinement and calibration using Monte Carlo rollouts and extra supervision to reshape intermediate decisions, whereas AWS dispenses with rollout-based critics and trajectory rewriting, instead correcting behavior online through information-gain–driven updates of a calibrated belief posterior.

LLM-based embodied baselines Compared to LLM-based embodied agents, AWS focuses on explicitly modeling uncertainty under partial observability and choosing actions by their expected information gain in a task-specific belief space. **ZSP** [15] treats frozen LLMs as zero-shot high-level planners that generate step-by-step textual plans and then semantically map each step to admissible actions, yielding executable sequences without belief tracking or closed-loop information gathering. **LLM-Planner** [39] and related ALFRED-style systems use LLMs to propose symbolic subgoal sequences and trigger re-planning when the low-level controller becomes stuck, but they revise plans based on success/failure signals rather than a calibrated posterior over hidden state. **SayCanPay** [14] scores actions by combining a language-model prior, an affordance model, and a learned payoff heuristic within a beam search, optimizing feasibility and cost but not explicitly reasoning about information gain or belief refinement. **WorMI** [47] adapts to new domains by retrieving and composing pre-trained world models and injecting their representations into an LLM policy via compound attention, whereas AWS uses a lightweight, task-specific belief posterior that is updated only from real observations and directly drives IG-based exploratory steps without relying on separate parametric world-model rollouts.

13 Method: Definitions

Global Belief ($\mathcal{G}$). The global belief captures abstract hypotheses about how the environment is structured or how a user organizes objects in the household. Formally, $\mathcal{G} = g_1, g_2, g_3$ is a set of natural language statements generated by an LLM at the beginning of the episode. These statements reflect user-specific organizational tendencies (e.g., “kitchen is well-organized” or “coffee-related items are grouped near the counter”) and serve as a latent prior that conditions downstream inference and action selection.

Subject Belief ($\mathcal{S}$). Given a search target s, the subject belief $\mathcal{S}_s$ is represented as a normalized categorical distribution over all symbolic locations (e.g., cabinet”, drawer”) and their instances (e.g., “cabinet3”) in the environment. Each score $p(l \mid s)$ reflects the agent’s current estimate of the likelihood that object s resides at location l. Subject priors are initialized from room beliefs and are progressively refined through both simulated and actual observations as the agent explores.

14 Method: Implementation

14.1 Belief Update Pipeline

At each timestep t, the agent updates its belief state hierarchically, following the top-down structure of $(\mathcal{G}, \mathcal{S})$. These updates can occur in two modes: (1) *simulated updates* for evaluating hypothetical actions using predicted observations, and (2) *actual updates* after executing an action and receiving a true environment observation.

Step 1: Global Belief Update. Given an observation o (real or predicted), the agent invokes a language model to revise its global belief $\mathcal{G}$. This involves generating a new set of high-level hypotheses that reinterpret how the environment may be structured given o. This update enables adaptive

abstraction, allowing the agent to reinterpret user-specific tendencies over time.

Step 2: Subject Belief Update. The subject-level belief $\mathcal{S}_s$ is refined based on the updated room scores. For a given subject s, the location prior $p(l \mid s)$ is adjusted such that locations associated with highly scored rooms receive a proportional boost. If $l \in r$, then:

$$p(l \mid s) \propto p(l \mid s) \cdot p(r)$$

This propagation ensures that high-level beliefs about the environment ultimately influence fine-grained search behavior.

This belief update process provides a flexible mechanism for belief propagation across abstraction levels, and serves as the foundation for estimating information gain and alignment in the action selection process.

14.2 Common Modules with Prompt Templates

Global Hypothesis Initialization To bootstrap the agent's environment understanding, we use the following prompt to initialize global-level hypotheses:

> **Prompt:**
> You are helping an agent search for the object `"mug"` in a household environment.
> Generate 3 plausible hypotheses about how the user organizes their space or uses the room.
> Each hypothesis should describe user habits or room-level behaviors (e.g., "kitchen is well-organized", "cabinets contain cups").

This prompt is passed to the LLM once per episode. The result is used to instantiate a `GlobalBelief` object, which stores a list of hypotheses guiding subsequent inference.

Low-level Subject Belief

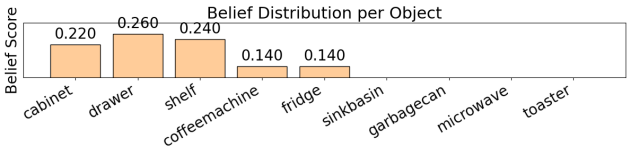

Figure 6. **Low-level Belief Distribution per Object.**

Figure 6 visualizes a snapshot of the low-level belief $b^{\mathcal{S}}$ over symbolic receptacles for a single search step. Each bar corresponds to a CHECK action (e.g., `cabinet`, `drawer`, `shelf`), and its height represents the probability $b^{\mathcal{S}}(a)$ that the target object will be found after executing that action (cf. Eq. 1). In this example, locations such as `drawer` and `shelf` receive higher belief mass than `fridge` or `garbagecan`, indicating that the global hypotheses and past observations have concentrated probability on containers that are more semantically plausible for the target. The distribution is normalized over all candidate receptacles and serves as the basis for IG-based action selection in AWS.

Hypothesis Update from Observation The global hypotheses are revised based on new environment observations:

> **Prompt: Global Hypothesis Update**
> You previously assumed the following about the user:
> `1. kitchen is well-organized`
> `2. cabinets contain cups`
> `3. mugs are often in the sink`
>
> Now the agent observed: `"the sink is empty"`
> Based on this new information, revise or expand the global hypotheses.

The updated hypotheses are used to refine prior scores or generate new priors.

Observation Simulation An observation of a candidate action is simulated based on the following prompt based on language-based belief:

> **Prompt: LLM-Based Observation Sampling**
>
> [Room Belief Summary]
> [Current Action] The agent goes to:
> [Prediction Task] List different sets of objects that the agent might observe at the location '{action}', based on the current belief. Each line should contain 3 likely objects, separated by commas.
> Example: `1. mug, plate, spoon 2. fork, mug, cup` ...

A number of simulated observations are used to quantify an information gain and an alignment score, which measures the similarity between predicted and real observations.

14.3 Variants in Belief Projection

We experiment with two inference-time belief adjustment variants, both operating on top of the shared global belief.

LLM-Based Projection Given a global hypothesis (e.g., "cabinets contain cups") and the known list of room-level symbols (e.g., `["cabinet", "drawer", "sink"]`), we use the following prompt:

> **Prompt: LLM-Based Projection**
> Given the hypothesis: `"cabinets contain cups"`
> Among the locations `["cabinet", "drawer", "sink"]`, decide which to boost or suppress.
> Return only a JSON object with this format:
> `{"boost": [...], "suppress": [...]}`

The returned result adjusts room scores: boosting adds +1.0, suppressing applies a 0.5 multiplier. The scores are then normalized. This process is repeated for each global hypothesis. Optionally, we log the decisions for interoperability.

Similarity-Based Projection. Upon receiving an environment observation $o \in \mathcal{O}_t$ after taking action $a \in \mathcal{A}_t$, we refine the global symbolic belief $B_t^{\mathcal{G}}$ and the action-level belief $b_t^{\mathcal{S}}$ based on the semantic similarity between observed object mentions and the agent's maintained symbolic hypotheses.

Let $\mathcal{M}_t = m_1, \ldots, m_k$ denote the set of symbolic mentions extracted from observation o, and let $\mathcal{H} = h_1, \ldots, h_n$ represent the current set of symbolic hypotheses maintained in the global belief. For each hypothesis $h \in \mathcal{H}$, we compute a similarity-based update score:

$$s_r = \sum_{m \in \mathcal{M}_t} \sum_{h \in \mathcal{H}_r} \text{sim}(m, h) + \delta_r(a)$$

where $\text{sim}(m, h) \in [0, 1]$ denotes a similarity function (e.g., cosine similarity), and $\delta_h(a)$ is an action-based heuristic adjustment defined as:

$$\delta_r(a) = \begin{cases} 0.2 & \text{if symb}(a) \in r \\ 0 & \text{otherwise} \end{cases}$$

These scores are used to adjust the confidence of each hypothesis in the belief state:

$$\text{score}_t(r) \leftarrow \text{score}_{t-1}(r) + s_r$$

The resulting scores are then normalized to yield a refined posterior belief over symbolic hypotheses:

$$\text{score}_t(r) \leftarrow \frac{\text{score}_t(r)}{\sum_{r' \in \mathcal{R}} \text{score}_t(r')}$$

This mechanism enables the agent to semantically align its symbolic belief with new observations through soft updates, without requiring any room-level decomposition or predefined symbolic rules.

14.4 Alignment Heuristic and Termination

Reward Computation. Alignment is computed as set precision and averaged over recent steps for convergence detection. We use a set-overlap score for similarity:

$$\text{Align}(a) = \frac{|\texttt{Predicted}(a) \cap \texttt{Actual}(a)|}{|\texttt{Actual}(a)|}$$

When used in forward planning, we approximate expected alignment via:

$$\mathbb{E}_{\hat{o} \sim \pi_{\text{pred}}(B, a)} \left[\text{sim}(\hat{o}, o)\right]$$

where o denotes a sampled candidate outcome. This allows us to estimate how well an action aligns with the latent world model under the current belief.

Given the expected alignment $\text{Align}(a)$ and the estimated information gain $\text{IG}(a)$ for a candidate action a, we define

$$r(a) = (1 - \lambda_{\text{IG}}) \, \text{Align}(a) + \lambda_{\text{IG}} \, \text{IG}(a),$$

where $\lambda_{\text{IG}} \in [0, 1]$ controls the trade-off between exploiting actions that are already well aligned with the latent world model and exploring actions with high information gain.

Termination. The loop ends when the target is found or when the average alignment over the past Δ steps exceeds the pre-defined threshold $\delta \in (0, 1]$.

14.5 Pseudocode and Hyperparameters

14.5.1. Pseudocode

Algorithm 1 provides AWS's inference-time pseudocode.

Algorithm 1: Pseudocode for Align-While-Search

Input: Target s, environment $\mathcal{E}$
Initial belief $B_0^{\mathcal{G}}$, $b_0^{\mathcal{S}}$, predictor π_{pred}, LLM π_{LLM}
Parameters: steps T, samples K
Output: Final belief state, success/failure status
for $t = 1$ **to** T **do**
  $A_t \leftarrow$ Top-k unexplored symbolic actions from $b_t^{\mathcal{S}}$
  foreach $a \in A_t$ **do**
    $B_{t+1}^{\mathcal{G}} \leftarrow \texttt{GlobalBeliefUpdate}(B_t^{\mathcal{G}}, \pi_{\text{BU}})$
    foreach $k \in K$ **do**
      $\hat{o}_t \leftarrow \pi_{\text{pred}}(B_{t+1}^{\mathcal{G}}, a)$
      $b_{t+1}^{\mathcal{S}} \leftarrow$ $\texttt{BeliefProjection}(B_{t+1}^{\mathcal{G}}, b_t^{\mathcal{S}}, \pi_{\text{BP}})$
    end
    Estimate $\mathbb{E}[\text{IG}](a)$ via $\{\hat{o}_t\}_a^K$, $\{b_t^{\mathcal{S}}\}^K$, $\{b_{t+1}^{\mathcal{S}}\}^K$
  end
  $x^* \leftarrow \text{argmax}_{a \in A_t} \mathbb{E}[\text{IG}](a)$
  $o_t \leftarrow \mathcal{E}.\texttt{step}(x^*)$
  $\hat{o}_t \leftarrow \pi_{\text{pred}}(B_{t+1}^{\mathcal{G}}, x^*)$
  $r_t \leftarrow r(\texttt{Align}(\hat{o}_t, o_t), \text{IG}(x^*))$
  if $s \in o_t$ *or* $r_t \geq \theta$ **then**
    return FOUND
  end
end
return NOT FOUND

14.5.2. Hyperparameters

Hyperparameter configuration and default settings are listed below. We report the final score as configured, unless otherwise inferred.

Symbolic Location Selection. The agent ranks top-k unexplored symbols from the belief prior (k=3), estimates information gain for each, and selects the best action.

Observation Sampling. For each action, we sample $K{=}3$ predicted observations using a frozen LLM (temperature 0.7), each containing 2–4 likely objects.

Reward Computation. We normalize both terms to $[0, 1]$ and set $\lambda_{\text{IG}} = 0.5$ for the main experiment, unless otherwise specified in the sensitivity analysis.

Termination. We set step budget $\Delta = 20$, and alignment threshold $\delta = 0.75$ for the main experiment.

15 Method: Full Algorithms

Algorithm 2: `Predictor.sample_observation()`

Input: Belief context $B^{\mathcal{G}}$, action a, samples N
Result: Predicted observations $\{\hat{o}^{(1)}, ..., \hat{o}^{(N)}\}$
for $i = 1$ **to** N **do**
  Construct prompt p_i with $B^{\mathcal{G}}$ and a
  $\hat{o}^{(i)} \leftarrow \pi_{\text{LLM}}(p_i)$ `// LLM response`
end
return $\{\hat{o}^{(1)}, ..., \hat{o}^{(N)}\}$

Algorithm 3: SubjectBelief.update_with_observation()

Input: Subject s, location x, observation o
Data: Belief $P(x|s)$ over $x \in \mathcal{X}$
Result: Updated $P'(x|s)$
$\mathcal{O}_x \leftarrow$ `parse_subjects_from`(o) `// Extract objects at` x
foreach $x' \in \mathcal{X}$ **do**
  if $x' = x$ **then**
    if $s \in \mathcal{O}_x$ **then**
      $P'(x'|s) \leftarrow P(x'|s) \cdot \alpha$ `// Boost`
    else
      $P'(x'|s) \leftarrow P(x'|s) \cdot \beta$ `// Decay`
    end
  else
    $P'(x'|s) \leftarrow P(x'|s)$
  end
end
Normalize $P'(x'|s)$ over x'

Algorithm 4: Inference-Time Belief Alignment with Multi-Level Update

Input: Target subject s, environment $\mathcal{E}$
Initial beliefs: Global $B_0^{\mathcal{G}}$, Subject $b_0^{\mathcal{S}}$
Predictor π_{pred}, LLM π_{LLM}
Parameters: max steps T, sample count N, top-k actions, threshold θ_{align}, weight λ
Output: Final belief state, alignment log, found/not-found status
Initialize tracker, set $t \leftarrow 0$
for $t = 1$ **to** T **do**
  $A^c \leftarrow$ Top-k unexplored symbols from B_t^s `// Candidate actions`
  foreach $a \in A^c$ **do**
    IGList $\leftarrow [\,]$
    for $i = 1$ **to** N **do**
      $\hat{o}_a^{(i)} \leftarrow \pi_{\text{pred}}(B_t^{\mathcal{G}}, a)$ `// Predict observation`
      $B^{\mathcal{G}'} \leftarrow B_t^{\mathcal{G}}$.`update`$(\hat{o}_a^{(i)}, \pi_{\text{LLM}})$
      $b^{\mathcal{S}'} \leftarrow$ $b_t^{\mathcal{S}}$.`adjust_from_global`$(B^{\mathcal{G}'})$
      IGList.`append`$(H(b_t^s) - H(b^{s'}))$
    end
    $\mathbb{E}[\text{IG}](a) \leftarrow$ mean(IGList)
  end
  $a^* \leftarrow \arg\max_{a \in A^c} \mathbb{E}[\text{IG}](a)$
  $x^* \leftarrow$ `SelectInstance`(a^*)
  $o_t \leftarrow \mathcal{E}$.`step`$(x^*)$
  Log: tracker.`log_action`(x^*, o_t)
  $B_{t+1}^g \leftarrow B_t^g$.`update`$(o_t, \pi_{\text{LLM}})$
  $b_{t+1}^s \leftarrow b_t^s$.`adjust_from_global`$(B_{t+1}^{\mathcal{G}})$
  b_{t+1}^s.`update_with_observation`(b_t^s, x^*, o_t)

  $\hat{o}_{\text{align}} \leftarrow \pi_{\text{pred}}(B_{t+1}^{\mathcal{G}}, x^*)$
  $r_{\text{align}} \leftarrow$ `AlignScore`$(\hat{o}_{\text{align}}, o_t)$
  if $r_{align} \geq \theta_{align}$ ***or*** $s \in o_t$ **then**
    return FOUND, tracker
  end
end
return NOT FOUND, tracker

16 Baseline Comparison

Figure 11 contrasts how RAFA and AWS use model-based rollouts. RAFA (top) starts from the current observation and expands a tree over candidate actions and their predicted observations, scoring only the leaf nodes with a scalar value estimate; the action leading to the highest-valued branch is then executed, without keeping track of how the rollout changes the agent's belief. In contrast, AWS (bottom) maintains an explicit belief over hypotheses and action preferences. For each candidate action, AWS predicts the next observation under the current belief, selects the action with the highest expected information gain, and executes it in the real environment. The resulting observation is compared against the predicted observation to obtain an alignment score, which is used to update the belief. Subsequent top-K search is then performed under this updated belief, so that rollouts are guided not only by value but also by how well the hypothesized world model aligns with actual feedback.

17 Baseline Reproduce Details

17.1 Inference-time baselines

ReAct Yao et al. [46] interleaves natural language reasoning ("Thought") with environment/tool interactions ("Act") in a single trajectory, so that an LLM can iteratively plan, execute actions (e.g., search or navigation), and update its beliefs using the resulting observations.

Reflexion Shinn et al. [37] proposes a lightweight framework that enables LLM agents to iteratively improve via verbal self-reflection, without parameter updates. By converting feedback signals into natural language reflections stored in episodic memory, agents refine future trajectories through in-context guidance.

RAP Hao et al. [13] treats LLM reasoning as planning with an internal world model: the LLM is repurposed both as a world model that predicts the next textual "state" given the current state and an action (e.g., a sub-step, operation, or move), and as an agent that proposes those actions. On top of this, RAP runs Monte Carlo Tree Search (MCTS) over state–action traces, using LM-derived heuristics (action likelihood, state confidence, self-critique, and simple task rewards) to evaluate and backpropagate returns along the tree.

LATS Zhou et al. [52] wraps a CoT- or ReAct-style language agent with MCTS over thought–action trajectories, where the LLM proposes multiple candidate thoughts or actions at each node and an LM-based value function, optionally augmented with self-reflection, evaluates partial rollouts. However, LATS fundamentally assumes access to a reversible environment simulator during search, which our text-based environments do not provide, and our attempt to replace the simulator with an LLM-based world model led to almost zero success; therefore, following LAC [8], we exclude LATS from our table.

RAFA Liu et al. [27] introduces a principled framework that integrates long-term reasoning and short-term acting for autonomous LLM agents. By modeling reasoning as Bayesian adaptive MDP planning and employing in-context learning for policy updates, RAFA achieves provable $\sqrt{T}$ regret and demonstrates strong empirical performance across multiple benchmarks.

QuBE Kim [21] augments ReAct-style LLM agents with a question-based belief module: when the agent appears stuck, an LLM asks and answers a few structured questions about the current state and history to form a textual belief, which is then used to regenerate reasoning and actions, reducing reasoning derailment in partially observable tasks. We exclude QuBE from our main table because it depends on environment-specific tool APIs and hand-crafted question templates that do not transfer to our setting.

LAC Dong et al. [8] proposes an actor-critic framework that equips LLMs with gradient-free policy improvement through Q-value estimation based on token-level logits and rollout-based reasoning. By jointly leveraging the actor's generation capabilities and the critic's long-horizon evaluation, LAC enhances goal-directed decision-making and achieves strong performance on complex planning and tool-use tasks. However, LAC requires an explicitly trained critic, and in our inference-time setting with proprietary LLMs we cannot train such a critic, making it unreliable; as a result, the success rates are near zero and we exclude LAC from our main comparison table.

ReflAct Kim et al. [20] replaces the standard ReAct-style "Thought→Action" loop with a goal-state reflection step that first summarizes the agent's belief about the current environment and its progress toward the task goal before selecting the next action. By conditioning action selection on this structured reflection, ReflAct encourages world-grounded reasoning, mitigates hallucinated or looping behaviors, and consistently improves success rates over both NoThinking and ReAct-style baselines in text-based embodied environments.

17.2 train-time baselines

ETO Song et al. [40] introduces a novel learning framework for LLM agents that leverages both successful and failed trajectories to enhance performance. By iteratively collecting failure trajectories during exploration and applying contrastive learning (e.g., DPO loss) between failure-success pairs, ETO refines agent policies beyond traditional imitation learning.

WKM Qiao et al. [32] enhances LLM-based agents with structured task-level and state-level knowledge extracted from expert trajectories. At inference time, the agent leverages global task knowledge for high-level planning and dynamic state knowledge for step-wise action grounding, reducing hallucinated or inefficient behavior. This approach improves generalization across unseen tasks and environments by integrating procedural context directly into the action selection process. As the original paper does not report scores for LLaMA 3.1, we reproduced their method under the same environment and evaluation setting.

MPO Xiong et al. [45] introduces a plug-and-play framework that enhances LLM-based agents by providing high-level, abstract meta plans to guide task execution. By leveraging feedback from agent performance, MPO refines these meta plans, leading to improved task completion efficiency and generalization across unseen scenarios.

IPR Xiong et al. [44] proposes an iterative step-level process refinement framework that augments trajectory-tuned LLM agents with step-level process supervision: from each step, a scorer policy Monte-Carlo–rolls out futures to estimate a step reward, agent and expert actions are compared to form contrastive step pairs, and the agent is updated by jointly applying outcome-level DPO, step-level DPO, and SFT so that intermediate decisions, not just final success, are directly optimized.

STeCa Wang et al. [43] proposes a step-level trajectory calibration framework for LLM agents in long-horizon tasks. Instead of relying only on episode-level rewards, STeCa uses Monte Carlo rollouts to estimate step-wise value drops along a trajectory, detects "deviation" steps where performance sharply deteriorates, and invokes an LLM-based reflection module to rewrite those problematic actions while stitching the remainder back to an expert suffix.

17.3 LLM-based embodied baselines

ZSP [15] treats large language models as zero-shot high-level planners for embodied tasks: given a natural language instruction and a few in-context examples, the LM generates a step-by-step textual plan, then each step is semantically mapped (via sentence embeddings) to admissible actions in environments like VirtualHome, optionally with iterative plan correction, achieving executable action sequences without any additional training or environment interaction.

LLM-Planner [39] uses a frozen LLM (GPT-3) as a high-level planner for ALFRED-style embodied tasks, generating a sequence of symbolic subgoals (e.g., navigation and manipulation targets) from natural language instructions via in-context learning. A separate low-level controller (HLSM) then executes each subgoal in the visual environment, and when the agent gets stuck, the system performs grounded re-planning by feeding back the completed subgoals and observed objects to the LLM to revise the remaining plan.

SayCanPay [14] extends SayCan-style LLM planning by scoring each candidate action with three factors: a language-model prior over actions ("Say"), an affordance model that checks whether the action is executable in the current state ("Can"), and a learned heuristic that estimates the downstream payoff or cost of continuing the plan from that action ("Pay"). Using these scores within a beam search over action sequences, SayCanPay finds plans that are both environment-feasible and cost-effective, improving success rate and plan efficiency over purely greedy LLM planners.

WorMI [47] is a framework where an LLM-based embodied agent adapts to new environments by composing multiple pre-trained world models only at test time. Each domain-specific world model is trained with dynamics prediction, affordance estimation, and behavior cloning, and WorMI (i) uses prototype-based retrieval over object-wise embeddings of the current scene to select a small subset of relevant world models, then (ii) injects their intermediate representations into the LLM policy via a world-wise compound attention module that first aggregates across world models and then aligns the result with the LLM's reasoning layers.

Code repositories are as follows:

- ReAct [46]: https://github.com/ysymyth/ReAct
- Reflextion [37]: https://github.com/noahshinn/reflexion
- RAP [13]: N/A
- LATS [52]: https://github.com/lapisrocks/LanguageAgentTreeSearch
- RAFA [27]: https://github.com/agentification/RAFA_code
- LAC [8]: https://github.com/drdh/LAC
- ETO [40]: https://github.com/Yifan-Song793/ETO
- WKM [32]: https://github.com/zjunlp/WKM
- MPO [45]: https://github.com/WeiminXiong/MPO
- IPR [44]: https://github.com/WeiminXiong/IPR
- STeCa [43]: https://github.com/WangHanLinHenry/STeCa/tree/main
- ZSP [15]: https://github.com/huangwl18/language-planner
- SayCanPay [14]: https://github.com/RishiHazra/saycanpay

- WorMI [47]: https://github.com/mjyoo2/WorMI/tree/main

18 Experiment Configuration

18.1 Environment

We evaluate our method on two partially observable instruction-following benchmarks: ALFWorld. In ALFWorld, the agent is tasked with finding a target object (e.g., mug, toiletpaper, soap) within a simulated household, navigating and interacting with discrete actions (go, open, search) to gather natural language feedback (e.g., "You open cabinet 1. It is empty."). Observations do not explicitly name the target unless correctly found. We cap the maximum number of steps per episode at 30 for ALFWorld to enforce consistent evaluation with baselines [38]. In VirtualHome, the agent acts in a 3D household simulator where everyday activities (e.g., "Read book", "Listen to music") are encoded as symbolic programs, and must follow a natural-language task description to execute an appropriate sequence of high-level actions under partial observability. In BabyAI-Text, the agent operates in a grid-based environment with first-person partial observability, receiving high-level language instructions (e.g., "go to the yellow box") and navigating via primitive actions (turn, move, pick up). The environment provides symbolic feedback without explicit target names unless the agent reaches the correct location [6].

18.2 Agent Configuration

We evaluate both open-source and proprietary LLMs as agents. For open-source models, we use LLaMA 2 and LLaMA 3.1–8B variants, testing both task-specific fine-tuned models and unmodified (zero-shot) versions. We additionally evaluate large open-source models including LLaMA 3.1-70B and Qwen 2.5-72B via the Together API. Proprietary models such as GPT-4o-mini and GPT-4 are accessed via the OpenAI API and used as-is without any additional adaptation. All models are prompted using a unified instruction-following format across tasks. We set the maximum output token length to 1024. Temperature is set to 0.7 for observation sampling tasks and defaults otherwise.

18.3 AWS instantiation

AWS is instantiated over different search spaces depending on the environment. In **ALFWorld**, AWS is used as a *location-centric* search procedure: the belief state b_t is defined over hypotheses about *where* the target object may reside (e.g., rooms, receptacles, or local regions), and the planner selects exploratory actions that refine this spatial belief. In **BabyAI**, we adopt a similar location-centric formulation, but on a 2D grid with a lower-level action space. Here, b_t is defined over hypotheses about the grid/room location of goal-relevant objects and intermediate waypoints (including which rooms or doors are likely to matter for the current instruction), and AWS uses this belief to prioritize navigation, door-manipulation, and pickup actions that are expected to yield the largest information gain about the instruction-relevant part of the map. MCTS thus explores different candidate navigation traces under this belief, updating b_t whenever observations confirm or rule out hypothesized object locations or paths. In **VirtualHome**, object locations and executable symbolic programs are already provided, so we instead instantiate AWS as a *rule-centric* search over candidate program rules and action schemata rather than over physical locations. The belief b_t in VH-AWS thus represents the plausibility and priority of symbolic rules (i.e., which sequence of room transitions and object interactions, subject to preconditions, is most consistent with the goal and observations), and MCTS explores this rule space by simulating rollouts of competing rule hypotheses. Observed successes, failures, and prediction–observation mismatches act as information-gain signals that update the rule-level belief, preserving the same belief-guided exploratory inference principle while shifting the search space from a spatial world model (ALFWorld) to a symbolic rule/world-model space (VirtualHome).

VirtualHome multi-modal. For the multi-modal VirtualHome setting, we reuse the same instruction-following tasks, splits, and evaluation protocol as in the text-only setup. The only difference is the observation modality: at each step the agent receives both the natural-language goal and a rendered first-person view of the scene, and AWS is instantiated with a vision-language backbone instead of a text-only LLM. We keep all search hyperparameters identical to the text-based VirtualHome configuration so that differences in performance can be attributed purely to the change in observation modality.

18.4 Task Protocol

In ALFWorld, we follow the standard evaluation splits introduced in prior work, using 140 seen and 134 unseen tasks from the reference benchmark. Each task is executed once without retries. Success is measured by correct object localization or item selection within the step limit.

Task type	# train	# seen	# unseen
Pick & Place	790	35	24
Examine in Light	308	13	18
Clean & Place	650	27	31
Heat & Place	459	16	23
Cool & Place	533	25	21
Pick Two & Place	813	24	17
All	3,553	140	134

Table 6. Six ALFWorld task types with heldout seen and unseen evaluation sets.

In VirtualHome, we follow the instruction-following task protocol of Wang et al. [43] and Yoo et al. [47], evaluat-

ing on the released test set of scripted household activities (e.g., "Read book", "Listen to music", "Pet cat"). Each task provides a natural-language goal and a reference symbolic program; during evaluation, the agent observes only the goal text and must generate an executable program. An episode is counted as successful if the produced program executes without error and brings the environment to a goal-satisfying state within the step limit.

In BabyAI-Text, we follow the task configuration and instance count from Dong et al. [8], evaluating each subtask with 50 held-out instances. We focus on the `GoTo` subtask, which serves as the primary evaluation setting in LAC and directly aligns with the core challenge in AWS, which is efficient object search under partial observability. Each task begins with a goal instruction (e.g., "go to the yellow box") and proceeds under a limited action budget. An episode is considered successful if the agent reaches the correct object location within the step limit.

18.5 Computation Resources

All experiments were conducted using NVIDIA A100 (40GB) and RTX 3090 (24GB) GPUs on internal compute clusters. Experiments with open-source models were performed on 1–2 GPUs per run. Proprietary models (e.g., GPT-4) were accessed through the OpenAI API.

19 Additional Experimental Result

19.1 BabyAI Experiment

Across all backbones, AWS achieves a mean rank of **1.5**, tied for the best overall with LAC. Specifically, AWS ranks **1st or 2nd** on three out of four backbone models, demonstrating robust performance under varying LLM backbones. Note that the reported results are based on the LLM-projection variant of AWS, further highlighting the consistency and adaptability of our method in comparison to prior inference-time baselines.

Backbone	**Method**	**GoTo**
GPT-4	ReAct	28.0$^{(2)}$
	RAFA	36.0$^{(3)}$
	LAC	**64.0**$^{(1)}$
	AWS (Ours)	**64.0**$^{(1)}$
GPT-4o-mini	ReAct	36.0$^{(2)}$
	RAFA	22.0$^{(3)}$
	LAC	**72.0**$^{(1)}$
	AWS (Ours)	**72.0**$^{(1)}$
LLaMA-70B	ReAct	56.0$^{(2)}$
	RAFA	44.0$^{(4)}$
	LAC	48.0$^{(3)}$
	AWS (Ours)	**58.0**$^{(1)}$
Qwen2.5-72B	ReAct	52.0$^{(2)}$
	RAFA	20.0$^{(4)}$
	LAC	**76.0**$^{(1)}$
	AWS (Ours)	32.0$^{(3)}$

Table 7. **Performance on BabyAI-Text `GoTo` subtask. Bold** indicates the best score per backbone. Superscript$^{(n)}$ denotes the relative rank (n; 1 = highest, 4 = lowest) within each backbone. AWS ranks 1st or tied-1st on three out of four backbones, achieving an overall mean rank of 1.5, on par with the strongest baseline (LAC).

Model	SR(%) (↑)	PS (↓)
ZSP [15]	8.19 ± 0.33	29.4
LLM-FT	42.8 ± 0.76	20.8
LLM-Planner [39]	21.5 ± 0.42	27.7
SayCanPay [14]	45.7 ± 0.59	19.0
WorMI [47]	$\underline{66.1} \pm 0.80$	**15.2**
AWS (Ours)	**67.5** ± 0.73	$\underline{17.1}$

Table 8. **Performance comparison with LLM-based embodied baselines on VirtualHome**. **Bold**: best performance, Underline: second-best performance

19.2 VirtualHome Experiment

Table 8 compares AWS against recent LLM-based embodied agents on VirtualHome, where AWS is instantiated as a rule-centric search (Appendix 18.3). In the text-based setting, AWS attains a success rate of 67.5%, outperforming all prior methods such as WorMI (66.1 ± 0.80), SayCanPay (45.7 ± 0.59), LLM-FT (42.8 ± 0.76), and LLM-Planner (21.5 ± 0.42). At the same time, AWS achieves the second-lowest planning-step (PS) score among all compared agents, indicating that it solves tasks with fewer planning iterations. Overall, AWS delivers both higher success and more efficient planning than existing LLM-based embodied agents on VirtualHome.

19.3 Hyperparameter Sensitivity

Table 9 summarizes the sensitivity of AWS to its main search hyperparameters on the ALFWorld text benchmark when using the LLaMA 3.2 70B Instruct backbone (without supervised fine-tuning). We first vary the branching factor K while fixing the number of rollouts to 20 and the information-gain weight to $\lambda_{\text{IG}} = 0.5$. Increasing K from 1 to 3 improves the success rate (from 77.6% to 81.3%), while further increasing to $K = 5$ reduces performance (73.2%) and slightly increases the average number of steps. This suggests that our default configuration ($K = 3$) lies near a reasonable efficiency–performance trade-off, and that overly large branching factors can even hurt performance.

We then sweep the information-gain weight $\lambda_{\text{IG}} \in \{0, 0.25, 0.5, 0.75, 1.0\}$ while fixing $K = 3$ and the rollout budget. Across this range, the success rate varies within about 4.5 points (79.8%–84.3%) and the average number of steps changes by at most 1.3 steps, indicating that AWS is relatively insensitive to the exact choice of λ_{IG}. While the best success rate is obtained at $\lambda_{\text{IG}} = 0$ on this backbone,

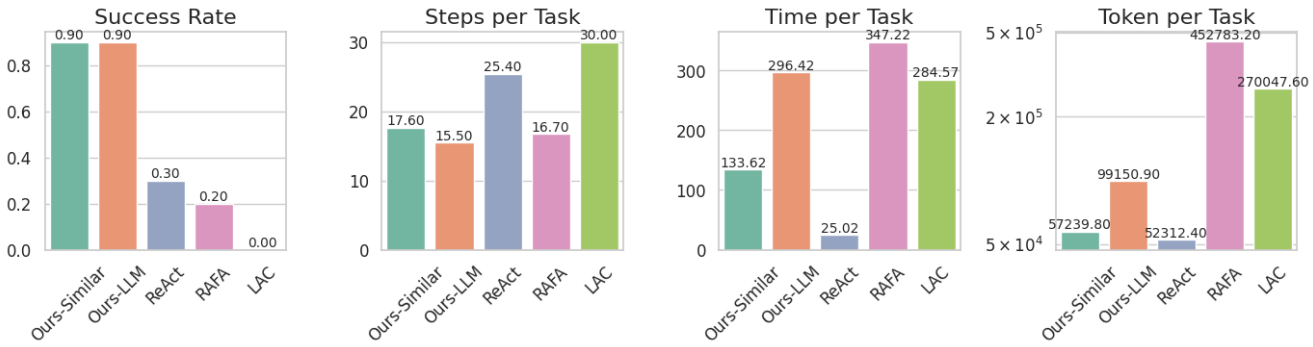

Figure 7. **Time/Step/Token Cost Comparison on ALFWorld subtasks**

moderate values between 0.25 and 0.75 achieve comparable performance with similar step counts. Overall, these results indicate that AWS does not rely on finely tuned hyperparameters: a single default setting provides robust performance across a broad range of K and λ_{IG}.

Setting	K	λ_{IG}	SR (%)	Step
Top-K sweep ($\lambda_{\text{IG}} = 0.5$, rollouts = 20)				
K1	1	0.50	77.6	15.6
K3	3	0.50	81.3	16.2
K5	5	0.50	73.2	16.9
λ_{IG} sweep ($K = 3$, rollouts = 20)				
L0	3	0.00	84.3	15.2
L025	3	0.25	79.8	15.7
L05	3	0.50	81.3	16.3
L075	3	0.75	80.3	16.5
L1	3	1.00	81.3	16.0

Table 9. **Sensitivity of AWS to the branching factor K and the information-gain weight λ_{IG} on ALFWorld.** We vary one hyperparameter at a time while fixing the other to its default value ($K = 3$, $\lambda_{\text{IG}} = 0.5$, rollouts $= 20$). SR denotes success rate and Step denotes number of steps (lower is better).

19.4 Latency Scaling

Estimating expected IG can require $A \times S$ simulated observations per step, raising latency concerns as branching grows. In our implementation, this overhead is explicitly capped by *Top-A pruning* (branching) and a *small fixed S*(sampling) (Top-A=3, S=1 by default):

$$\#\text{LLM calls/step} \approx 1\ (\text{agent}) + \text{Top}-A \times S\ (\text{surrogate})$$

To directly address *latency scaling* beyond aggregate token/-time, we add (i) a per-step compute breakdown (Table 10) and (ii) a controlled scaling ablation varying Top-A and S (Figure 8). Per-step latency grows roughly linearly with $B = \text{Top}-A \times S$, while success does not systematically improve with larger B. Importantly, the IG-rollout overhead is independent of the raw number of interactable objects since we cap evaluations to Top-A candidates (fixed A) and small S.

Method	SR↑	Tok/step↓	time(s)/step↓	Notes
Reflexion	51.3	35×10^4	197.84	iter. refine
AWS (ours)	76.0	4.9×10^4	102.93	Top-A=3, S=1

Table 10. **Per-step compute breakdown.**

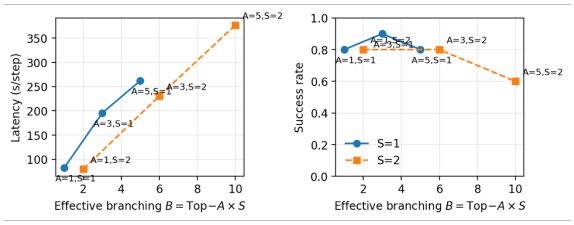

Figure 8. **Scaling ablation (Top-A, S).** We vary Top-$A \in \{1, 3, 5\}$ and sampling count $S \in \{1, 2\}$ and plot per-step latency (s/step) and success rate against the effective branching factor B.

20 Additional Performance Analysis

Belief Alignment Curve. To evaluate how effectively the agent refines its belief during exploration, we measured an alignment reward that reflects how well the predicted observation $\hat{o}_a$ matches the actual environment observation o_a after actually executing action a. Specifically, we define $\text{Align}(a) = \text{similarity}(\hat{o}_a, o_a)$, where $\hat{o}_a = \pi_{\text{pred}}(B, a)$ and the similarity$(\cdot, \cdot)$ function is described in Appendix 14. Then we plotted the alignment trajectory, i.e., the belief mass assigned to the ground-truth object location $b_t(y^*)$, over interaction steps. As shown in Figure 9(left), our method initially maintains moderate alignment, but exhibits a clear rising trend in the later steps, demonstrating that belief-guided exploration accumulates useful information over time. While

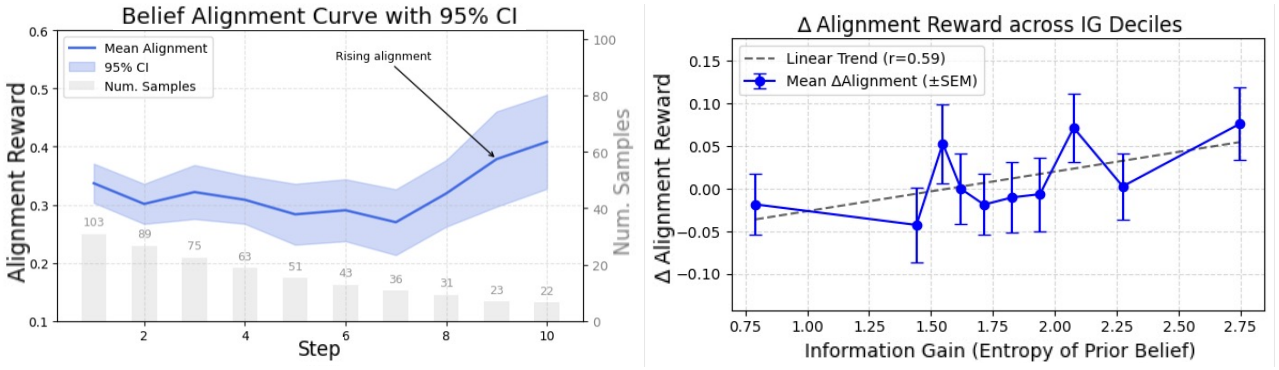

Figure 9. **Belief Alignment Analysis.** (**Left**) Belief alignment scores over time by tracking the agent's belief in the correct object location ($b_t(y^*)$) across interaction steps. Shaded regions indicate 95% confidence intervals. (**Right**) We group predicted information gain scores into deciles and measure the corresponding change in belief alignment (ΔR^{align}).

early steps (1–6) show limited gains due to the agent's need to disambiguate possible hypotheses, steps 7–10 demonstrate consistent alignment improvement, even under high variance from reduced sample counts. The upward shift in the alignment curve, coupled with tight confidence intervals in the early stages, indicates that our agent learns to prioritize informative actions. Overall, this trajectory supports our hypothesis that inference-time belief refinement enables robust adaptation, particularly in underexplored or ambiguous environments.

Information Gain and Alignment. To evaluate whether LLM-predicted IG scores are reliable indicators of belief refinement, we analyzed the correlation between predicted IG rankings and actual changes in belief alignment, defined as $\Delta R^{\text{align}} = R(b_{t+1}) - R(b_t)$, where $R(b)$ denotes the alignment reward (i.e., belief mass on the true location). We grouped predicted IG scores into deciles and computed the average ΔR^{align} per bin. As shown in Figure 9(right), higher IG bins are associated with greater alignment gains, forming a broadly monotonic trend. While per-step predictions are noisy, the global ordinal structure remains consistent, particularly in successful episodes where exploration is well-executed. These results validate the effectiveness of ordinal-scale LLM reasoning as a lightweight yet reliable proxy for belief-shifting action selection.

Max information gain achieves the best in downstream task. To evaluate the effectiveness of information gain as a guide for exploration, we analyzed how the choice of action-based on IG ranking-impacts downstream task success. Rather than relying on the absolute value of IG, we tested whether selecting the top-ranked IG candidate among available actions leads to better outcomes. As shown in Figure 5(left), selecting the top-IG action yields the highest success rate (87% on seen and 85.3% on unseen environments). Performance drops when choosing the second-best IG candidate (77.85% / 76.31%) and degrades further with the lowest-IG choice (58.33% / 63.38%). These results indicate that IG serves as an effective relative ranking signal for epistemic utility, enabling the agent to prioritize informative actions that ultimately improve task performance.

Belief prior aligns with actual object presence. To evaluate how well the agent's belief prior aligns with actual object presence, we analyzed the belief scores assigned to visited locations and compared them against whether the target object was found (*hit*) or not (*miss*). For each exploration step, we extracted the belief probability assigned to the visited receptacle and label the outcome based on object presence. As shown in Figure 5(right), hit locations receive substantially higher belief scores than miss locations. Specifically, the average belief score for hit cases was 0.452 (std = 0.133, $n = 117$), while for miss cases it was 0.314 (std = 0.143, $n = 555$). This difference is highly significant, with a t-test yielding $t = 10.06$ and $p = 3.98 \times 10^{-19}$. These results confirm that belief priors provide meaningful guidance for search, effectively distinguishing promising targets from distractors. Rather than serving as a passive estimate, the belief prior plays an active role in structuring efficient exploration, even under partial observability.

21 Computation Comparison

21.1 Computational Analysis

Let k be the number of symbolic candidate actions considered per step (e.g., top-k from the belief prior), and n the number of predicted observations sampled per action (e.g., from an LLM-based observation predictor). At each decision step, the following computations are performed:

- **Observation Simulation**: For each of the k candidate actions, we sample n possible observations using the observation predictor (typically an LLM). This requires $k \times n$ forward passes.

- **Belief Update**: Each predicted observation is used to simulate an update to the agent's hierarchical belief $(\mathcal{G}, \mathcal{S})$. These updates involve lightweight operations such as string matching or LLM-based projection, repeated $k \times n$ times.
- **Utility Estimation**: For each candidate action, we compute the expected information gain and predicted alignment score over the n samples. This involves entropy calculations and output similarity comparisons.

The overall per-step computational complexity is:

$$\mathcal{O}(kn \cdot C_{\text{LLM}} + kn \cdot C_{\text{update}} + k \cdot C_{\text{score}})$$

where C_{LLM} is the cost of a single LLM forward pass (typically for short prompts), C_{update} is the cost of belief update per sample, and C_{score} includes entropy and alignment computation.

21.2 Time/Step/Token Cost Comparison

To assess the computational cost of our method, we evaluate each baseline on 10 randomly sampled unseen tasks from the ALFWorld benchmark. We report the average values across four key metrics: success rate, number of steps per task, wall-clock time per task, and token consumption per task. As shown in Figure 7, both variants of our method (Ours-Similar and Ours-LLM) achieve significantly higher success rates (0.90) compared to baselines such as ReAct (0.30), RAFA (0.20), and LAC (0.00), while maintaining competitive or lower computational costs. In particular, Ours-Similar demonstrates strong efficiency in both inference time (133.62s) and token usage (57,239 tokens), making it suitable for practical deployment without sacrificing performance. Although Ours-LLM incurs a higher cost (99,150 tokens), it still outperforms all prior baselines in success rate. These results highlight the superior cost-performance tradeoff of our approach under real-world constraints.

22 Qualitative Analysis

We analyze the evolution of the hypotheses over steps and provide a case study on the task of `put mug in garbagecan`.

22.1 Case Study

To better understand how our belief-guided search enables effective exploration, we conduct a case study comparing two variants, similarity-based and LLM-based belief updates, on the same task. As shown in Figure 10, both variants successfully discover the target object, but they exhibit distinct belief dynamics and exploration strategies. The similarity-based agent gradually increases its cabinet belief through localized refinement (Figure 10a), maintaining high entropy early on before committing to high-probability regions (Figure 10c; *blue*). In contrast, the LLM-based agent exhibits a sharper belief shift (Figure 10b) and faster entropy reduction by leveraging global cues extracted from past object co-occurrence or spatial semantics (Figure 10c; *yellow*). These belief dynamics translate to different search paths (Figure 10d): the similarity-based agent narrows down its focus through nearby locations, while the LLM-based agent performs semantically informed jumps to higher-yield receptacles early in the episode. In Table 11, we further compare the step-by-step hypotheses evolution of both agents, highlighting how their reasoning structures emerge over time.

22.2 Step-Wise Hypothesis Evolution

To better understand how each variant adapts global hypotheses over time, we present a step-wise comparison of inferred world structures during the mug search task (see Table 11). At each step, the agent forms hypotheses about room structure and object placement, conditioned on recent observations.

Similarity-Based Variant. This agent builds structure incrementally. It first assumes the kitchen counter is a centralized utility zone, then refines this into functional subregions (e.g., breakfast prep, plant care) as more items are observed. Later, it detects inconsistencies (e.g., salt and pepper in different cabinets) and ultimately confirms a coherent storage pattern-such as a mug stored alongside cleaning items-highlighting a gradual, zone-based interpretation process.

LLM-Based Variant. The LLM-based agent starts with less grounded hypotheses, interpreting surfaces as generic and cluttered workspaces. However, as structural signals emerge (e.g., numbered forks, categorized placements), it transitions rapidly to a policy-driven world model based on cleaning routines or semantic labeling. By the final step, it exhibits consistent categorization and spatial order.

In summary, while both variants succeed, their reasoning paths differ: **Similarity-Based** infers structure gradually via bottom-up spatial cues, whereas **LLM-Based** generalizes early from sparse patterns, leading to faster semantic organization.

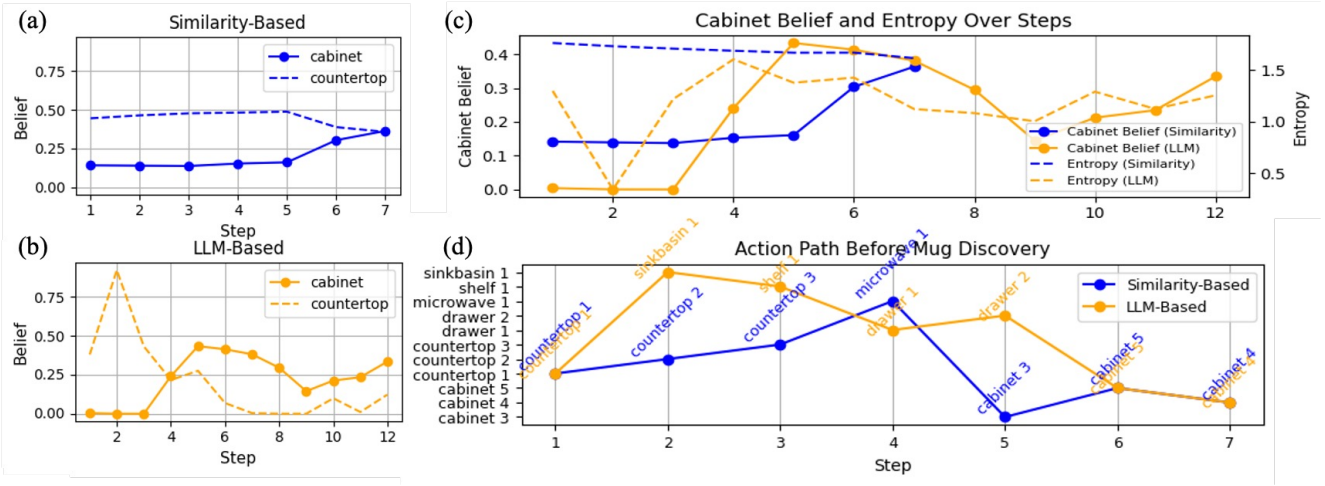

Figure 10. **Belief Update and Search Behavior Comparison.** (a) Similarity-based belief projection: step-wise belief dynamics (b) LLM-based projection: step-wise belief dynamics (c) For Similarity-based projection, belief changes gradually while for LLM-based projection, belief dynamics and cabinet-related entropy show more focused belief shaping. (d) The LLM agent explores semantically distant locations early, whereas the similarity-based agent refines search inlocalized clusters.

Step	Agent	Hypothesis Summary	Interpretation
1	Similarity	"Centralized kitchen counter"	Used for daily tools (sponge, spoon, knives)
	LLM	"Countertops used for daily activity"	Includes non-kitchen items (e.g., credit card, potato)
2	Similarity	"Multi-zone structure emerging"	Breakfast prep zone, Plant care zone
	LLM	"Countertop clutter"	No dedicated item location
3	Similarity	"Meal/snack prep inferred"	Tomato and potato → cooking zone emerging
	LLM	"Sinkbasin & countertop = temp workspace"	Multi-purpose usage (cluttered logic remains)
4	Similarity	"Storage strategy detected"	Soap + bottles categorized into cabinet
	LLM	"Zone cleaning in progress"	Begins forming structure (cleaning-based policy)
5	Similarity	"Gaps in storage logic"	Cabinets partially filled, inconsistent
	LLM	"Temporary vs. designated storage co-exist"	Sponge & bottle in cabinet → early categorization
6	Similarity	"Category mismatch in cabinet items"	Salt & pepper in different cabinets
	LLM	"Systematic labeling emerging"	Forks numbered → category grouping via labels
7	Similarity	"Storage pattern identified"	Mug observed in cabinet with soap → final confirmation
	LLM	"Consistent categorization & order"	Spatula 2 in drawer, cabinets tidied → structured conclusion

Table 11. Step-wise evolution of global hypotheses for each agent. The similarity-based variant exhibits gradual zone-based inference, while the LLM-based variant begins with cluttered understanding and quickly transitions to structured categorization.

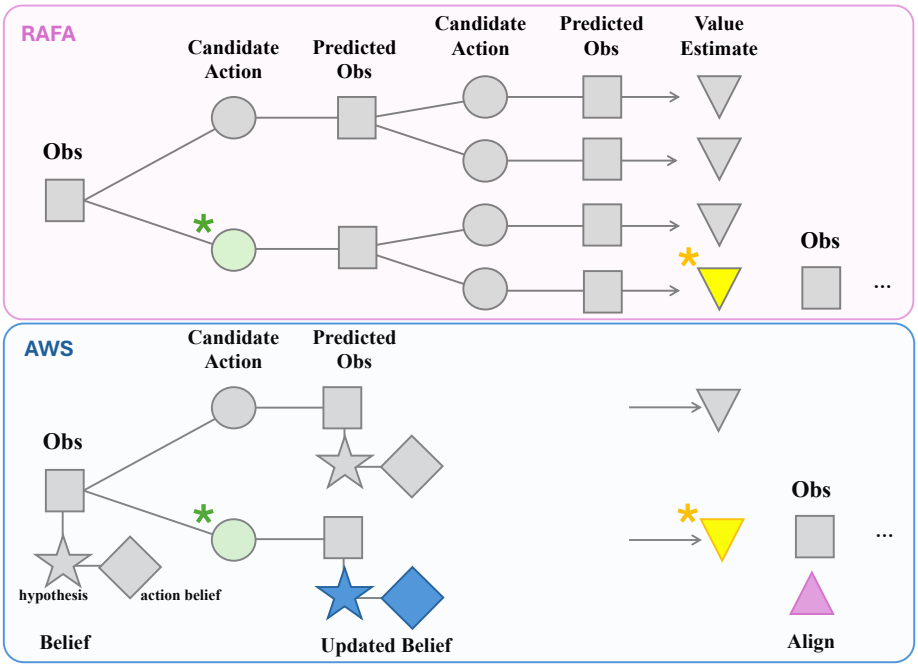

Figure 11. **Step-wise schematic comparison between RAFA and AWS.** RAFA performs rollout-based value estimation over candidate action sequences, while AWS maintains and updates a belief over hypotheses and chooses actions based on alignment between predicted and actual observations.