Information Seeking for Robust Decision Making under Partial Observability

Anonymous ACL submission

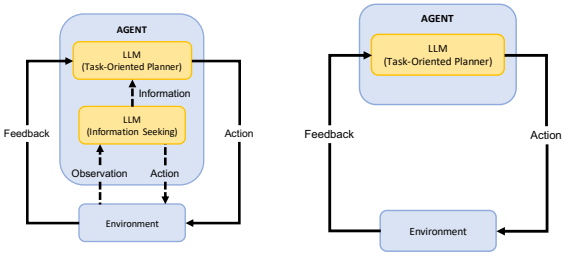
Abstract

Acting under uncertainty requires not only handling incomplete observations but also recognizing when one's internal model of the environment is wrong. Existing LLM planning agents largely address the former while assuming their internal dynamics match reality, leaving them brittle when this assumption breaks. We introduce InfoSeeker, a decision-making framework that interleaves task-oriented planning with active information seeking: before generating or revising a plan, the LLM proposes actions that validate its understanding, detect environmental changes, or test hypotheses. On a new benchmark of partially observable environments with uncertain dynamics, InfoSeeker attains a 74% absolute improvement over prior methods, and continues to outperform baselines that are explicitly informed of the perturbation – indicating that the gain stems from planned information seeking rather than prompt engineering or model scale. InfoSeeker also generalizes across LLMs and outperforms baselines on established robotic manipulation and web navigation benchmarks.

1 Introduction

Real-world decision-making tasks are often partially observable, where observations and dynamics may be noisy or uncertain. For example, in software engineering, a function may produce unexpected results from misuse or faulty implementation; in robotics, erroneous action control can result from wear or inaccurate controller tuning. Robust behavior in these settings requires diagnosing such failures from limited information.

Humans exhibit strong problem-solving capabilities in dynamic and uncertain environments, enabled by two core abilities. Task-oriented planning selects action sequences to achieve a goal, while information seeking (Gopnik, 2012; Kidd and Hayden, 2015; Case and Given, 2016) proactively gathers information to align internal beliefs, inferred from internal dynamics, with the external world. Information seeking is especially crucial under partial observability, where the true environment state is hidden and decisions rely on imperfect beliefs. For example, to reach the target in Fig. 2, we initially plan under the assumption that the robot arm follows the commanded (x, y) positions accurately. If the outcome deviates from expectations, we collect new evidence (e.g., trying various locations and measuring the resulting positions) to update our beliefs about environmental dynamics. Together, task-oriented planning and information seeking enable humans to uncover hidden causes and make robust decisions under uncertainty.



(a) Our InfoSeeker (b) Existing planners

Figure 1: **Decision-making frameworks. (a)** InfoSeeker validates its internal dynamics through information seeking before planning. **(b)** Existing approaches rely solely on execution feedback and fixed assumptions, leaving internal dynamics unchecked.

Large Language Models (LLMs) have emerged as versatile zero-shot agents for autonomous decision-making (Hu et al., 2023; Wang et al., 2023a; Hong et al., 2024). In interactive environments, they generate action sequences and revise strategies in response to feedback (Yao et al., 2023b; Wang et al., 2023b). This closed-loop planning paradigm has shown promise across diverse domains such as robotics (Sun et al., 2023; Wang et al., 2024) and scientific discovery (Jansen et al., 2024). Recent studies have examined LLMs under partial observability, emphasizing either the

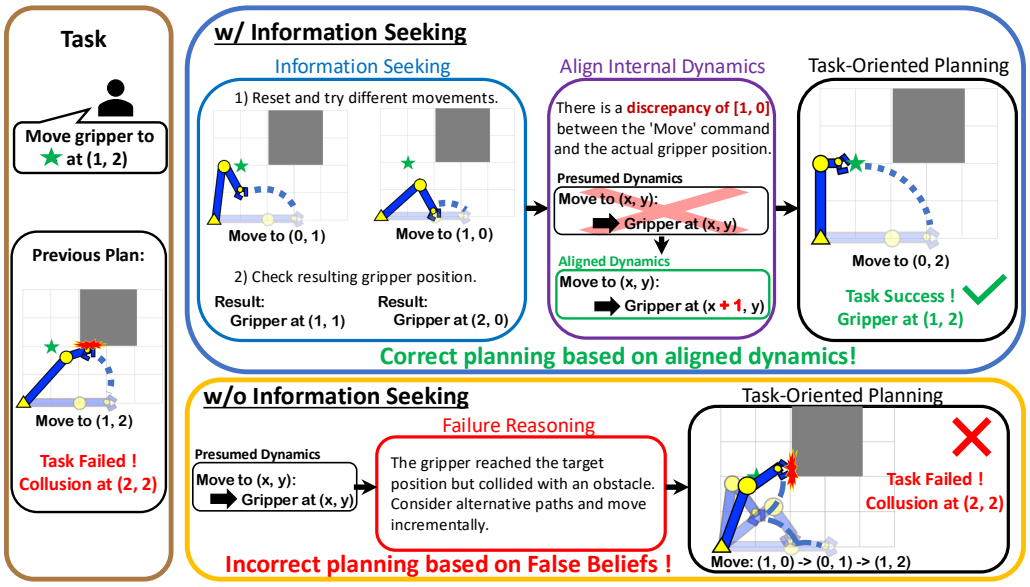


Figure 2: **Overcoming uncertainty through information seeking.** When tasked to move a robot gripper to a target location using miscalibrated controllers that add a constant (1, 0) offset to every command, existing planners (bottom) fail by over-relying on presumed dynamics without verifying them. In contrast, InfoSeeker (top) actively seeks information, detects discrepancies between commanded and executed motions, and updates its internal dynamics to generate a correct plan.

recovery of hidden knowledge (Ke et al., 2024; Krishnamurthy et al., 2024; Piriyakulkij et al., 2024; Pan et al., 2025) or identification of missing information in user instructions (Huang et al., 2024; Sun et al., 2024). However, these approaches overlook a critical challenge: mismatches between the agent's internal dynamics and the actual environment. Without informative observations to realign them, the agent develops inaccurate beliefs of latent states, leading to systematically flawed plans.

We propose InfoSeeker, a framework that integrates information seeking directly into the decision-making loop. Our key insight is that robust decision making requires explicitly planned information seeking actions to reconcile the agent's internal dynamics with the external environment. As illustrated in Fig. 2 (blue block) and Fig. 1a, InfoSeeker prompts the LLM to actively gather information before proposing or revising a plan. Specifically, the LLM conducts targeted diagnostic trials that validate its understanding and detect shifts in environmental dynamics. In contrast, prior interactive planning approaches (Fig. 2 yellow block, Fig. 1b) rely solely on reactive adaptation without explicit information gathering. By seeking evidence first, InfoSeeker uncovers the root causes of failure and adjusts its plans accordingly.

To evaluate our method's robustness, we introduce a suite of text-based simulation benchmarks that test LLM agents under partial observability and noisy dynamics. While information gathering under uncertainty is a long-standing principle in the classical POMDP literature (Kaelbling et al., 1998), to our knowledge no prior benchmark directly evaluates the planning capabilities of *LLM-based* agents under noisy environmental dynamics. Prior benchmarks (Fan et al., 2022; Shridhar et al., 2020; Wang et al., 2022a) focus solely on observation uncertainty, where action outcomes are largely predictable. In contrast, our benchmarks incorporate uncertain dynamics, where actions may yield unexpected results due to unmodeled factors. For example, as shown in Fig. 2, the robot's final position can deviate from the commanded target due to miscalibrated controllers. This setting better reflects real-world scenarios, requiring agents to handle both incomplete observations and unpredictable dynamics that violate their assumptions. Although hand-crafted, our benchmark highlights the need for more rigorous evaluations of planning under uncertainty in observations and dynamics.

InfoSeeker achieves an absolute performance gain of 74% over prior methods on our challenging benchmark. Strikingly, this advantage holds even when baselines are explicitly informed about the environmental perturbation in their prompts, and persists on the latest Gemini 3.0 Flash backbone – establishing that the gain stems from planned infor-

mation seeking rather than prompt engineering or LLM scale. Active information seeking enhances information acquisition while preserving sample efficiency. As a result, InfoSeeker produces more effective task-oriented plans, achieves success more rapidly, and requires fewer inference tokens. Moreover, InfoSeeker generalizes across different LLMs and outperforms existing approaches on established benchmarks, including LLM3 (Wang et al., 2024) and TravelPlanner (Xie et al., 2024), demonstrating both versatility and robustness.

2 Preliminary

2.1 Partially Observable Markov Decision Process

A Partially Observable Markov Decision Process (POMDP) models decision-making under uncertainty and is defined by the tuple $(\mathbb{S}, \mathbb{A}, \mathbb{Z}, T, O, R, \gamma)$, where $\mathbb{S}$ is a set of states, $\mathbb{A}$ is a set of actions, and $\mathbb{Z}$ is a set of observations. At each interaction step t, the environment is in a hidden state $s_t \in \mathbb{S}$. The agent takes an action $a_t \in \mathbb{A}$, transitions to a new state $s_{t+1} \sim T(\cdot \mid s_t, a_t)$, and receives an observation $o_{t+1} \sim O(\cdot \mid s_{t+1}, a_t) \in \mathbb{Z}$. Because the state is not directly observable, the agent infers the latent state by maintaining a belief $b_t \in \Delta(\mathbb{S})$—a distribution over states—updated via Bayes' rule from an initial belief state b_0:

$$b_t(s_t) = \frac{O(o_t|s_t, a_{t-1})}{P(o_t|b_{t-1}, a_{t-1})} \times \sum_{s_{t-1}} T(s_t|s_{t-1}, a_{t-1})\, b_{t-1}(s_{t-1}) \tag{1}$$

with observation likelihood conditioned on previous belief b_{t-1} and action a_{t-1} as:

$$P(o_t|b_{t-1}, a_{t-1}) = \sum_{s_t, s_{t-1}} O(o_t|s_t, a_{t-1}) \times T(s_t|s_{t-1}, a_{t-1})\, b_{t-1}(s_{t-1}) \tag{2}$$

The sparse reward function $R : \mathbb{S} \times \mathbb{A} \to \{0, 1\}$ specifies the immediate reward, and the discount factor $\gamma \in [0, 1)$ determines the weighting of future rewards. The agent seeks to maximize expected return using the Bellman equation:

$$V^*(b_t) = \max_{a_t} \Bigg[\sum_{s_t} b_t(s_t)\, R(s_t, a_t) + \gamma \sum_{o_{t+1}} P(o_{t+1}|b_t, a_t) V^*(b_{t+1}) \Bigg] \tag{3}$$

where $P(o_{t+1} \mid b_t, a_t)$ is defined as in Eq. 2.

In our setting, the components $\mathbb{S}$, $\mathbb{Z}$, T, O, and γ are unknown or only partially known. The agent is only given the action space $\mathbb{A}$ and the reward function R, both specified in natural language. We consider a zero-shot generalization scenario, where the agent performs on previously unseen tasks—each corresponding to a different POMDP—without any task-specific training. To obtain the optimal belief state (Eq. 1) and make decisions that maximize return (Eq. 3), the agent must actively collect informative observations $\{o_1, o_2, \ldots, o_t\}$ and estimate the transition probability $T(s_{t+1}|s_t, a_t)$.

2.2 LLM Planners in POMDP

Large Language Models (LLMs), pretrained on broad corpora, encode knowledge that supports zero-shot planning in novel environments. We interpret an LLM planner as a POMDP agent: given a task description and a trajectory $\tau_t = (o_0, a_0, o_1, \ldots, a_{t-1}, o_t)$, the LLM can reason about the latent states by constructing a belief b_t, and anticipate future changes via internal dynamics $T(s_{t+1}|s_t, a_t)$. It then generates a finite-horizon, task-oriented plan $a_{t:t+h-1}$ that maximizes expected return over h steps.

Although not trained for belief updating or value iteration, the LLM's reasoning can approximate Bayes' rule (Eq.1) and satisfy the Bellman equation (Eq.3), enabling zero-shot planning from prior knowledge and the input trajectory. However, with uninformative observations or misaligned internal dynamics $T(s_{t+1}|s_t, a_t)$, the resulting belief and value estimates become unreliable.

3 Method

We introduce InfoSeeker, an LLM agent that optimally estimates belief states by actively performing information seeking behaviors to collect informative observations and correct errors in the internal dynamics presumed by the LLM. As shown in Fig. 3, InfoSeeker performs iterative decision making: it first analyzes past trajectories to identify uncertainty and takes actions to gather information; next, it examines the resulting information seeking trajectories and refines task-oriented plans. By incorporating information seeking into the decision loop, InfoSeeker effectively refines its belief states and enhances future performance under uncertainty in both dynamics and observations.

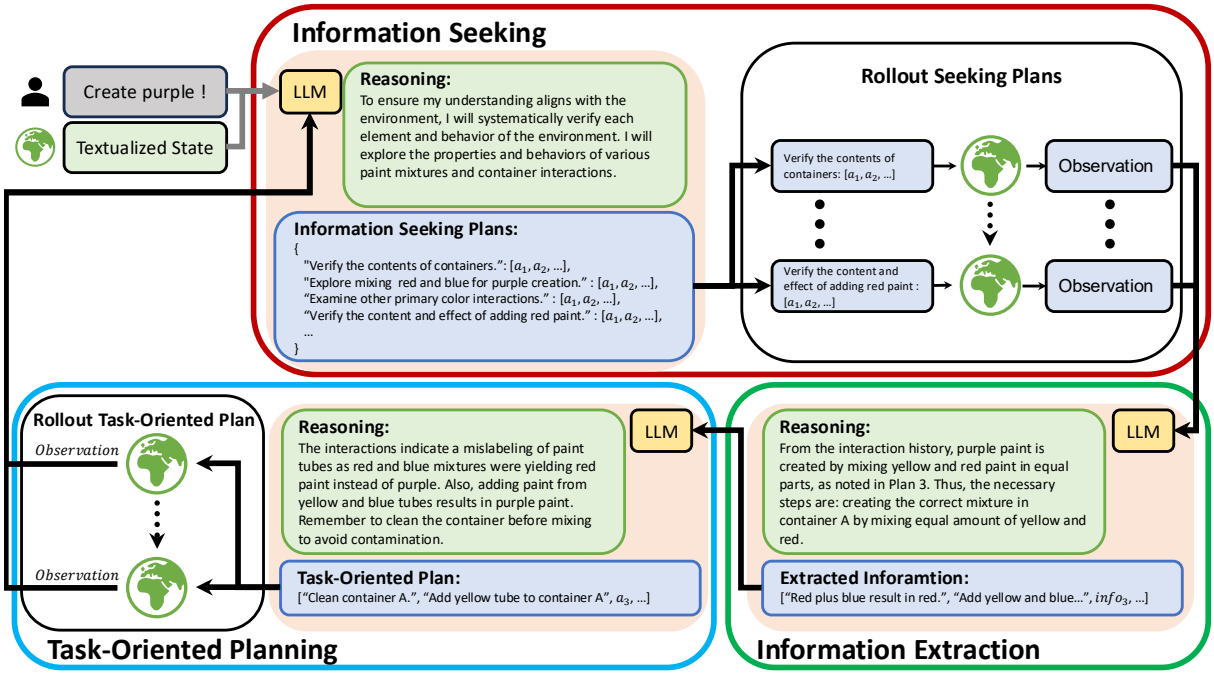


Figure 3: **System overview of InfoSeeker.** Our framework integrates *Information Seeking* (top-right) and *Task-Oriented Planning* (bottom-left) in a closed-loop process. The agent formulates and executes strategies to acquire missing knowledge, addressing gaps in its internal dynamics before generating more effective task plans. This iterative approach, supported by the reasoning capabilities of LLMs, enables the agent to reduce uncertainty and enhance planning effectiveness.

3.1 Information Seeking

Previous work (Huang et al., 2024; Sun et al., 2024) has focused primarily on uncovering missing information related to task instructions, often overlooking discrepancies between the agent's internal dynamics and the environment. In contrast, we propose a general prompting strategy that first guides the LLM to reason over current observations, prior plans, and interaction history, and then to identify targeted exploratory actions. When inconsistencies are detected, the agent can hypothesize potential errors, design focused experiments to test its assumptions, or detect shifts in environmental dynamics. This reasoning-driven process supports both the verification of the internal dynamics and the acquisition of informative observations, allowing for more accurate belief state update and more effective task-oriented planning.

After executing exploratory actions, the agent receives a new trajectory which it uses to update internal dynamics and refine belief states for subsequent planning. In practice, we introduce an information extraction module that prompts an LLM to analyze the information seeking trajectory, extract key insights, and generate concise summaries. These summaries are then incorporated into the context of subsequent task-oriented planning. We refer to Appendix K.1 for details of our prompt.

3.2 Task-Oriented Planning

Based on the extracted insights, the LLM is prompted to generate an initial plan or revise a previous one. Following Wang et al. (2024), our prompting strategy guides the LLM to reason over the context before producing a new sequence of task-oriented actions. Once these actions are executed, the updated trajectory is passed back to the LLM, initiating the next cycle of information seeking and task-oriented planning.

This iterative process continues until the task is completed or a predefined number of interaction steps is reached. By continually collecting informative observations and refining its internal dynamics, InfoSeeker adapts effectively to partially observable environments with uncertain or shifting dynamics. Full methodological details are provided in Algorithm 1 (Appendix C).

4 Benchmarking Robustness in POMDPs with Noisy Dynamics

We introduce a suite of text-based simulation benchmarks for evaluating the robustness of LLM

agents in partially observable environments with noisy environmental dynamics. Existing benchmarks (Fan et al., 2022; Shridhar et al., 2020; Wang et al., 2022a) consider only uncertainty in observations, where actions always yield predictable outcomes (e.g., "turn on lamp" action exactly turns on the lamp). In contrast, we consider incorporating uncertain dynamics: actions may fail due to unmodeled factors, such as control errors or environmental noise. This setup highlights the critical challenges of decision-making in real-world scenarios. To contrast agent performance under both certain and uncertain dynamics, we implement each task in two configurations: a *Basic* version, which resembles existing benchmarks, and a *Perturbed* version that additionally incorporates noisy dynamics.

4.1 Task Suite Overview

Our benchmark consists of five categories, each evaluating a different aspect of decision-making. Four categories have a *basic* and a *perturbed* variant, while *mix color* has one basic and two perturbed variants, yielding 11 variants in total. Every variant contains 50 task instances.

Robot arm control. A 2D motion planning task in which the agent controls a planar 2-joint robot arm to move its end-effector to a target location while avoiding static obstacles. In the *Basic* condition, actions are executed as intended. In the *Perturbed* condition, a constant offset $(\Delta x, \Delta y)$ is added to each commanded action, simulating real-world uncertainty from actuation bias or calibration errors. The agent must infer this discrepancy and adapt its planning to compensate for the offset.

Robot navigation. A long-horizon task in which a mobile robot must reach a ball, pick it up, and navigate to a designated goal location. The agent controls the robot using four directional actions: forward, backward, left, and right. In the *Basic* condition, each action produces the expected movement. In the *Perturbed* condition, uncertain dynamics are introduced by inverting the action mappings—for example, the "left" action causes the robot to move right. To complete the task, the agent must detect and adapt to these inconsistencies between its commands and the robot's actual behavior.

Stack Single Block and **Stack Multiple Blocks.** Inspired by classic BlocksWorld scenarios (Valmeekam et al., 2023), the agent must rearrange stacks of colored blocks to match a target configuration. It can pick up the top block from a stack and store it in a one-slot inventory, and may place blocks from the inventory onto a stack. The single block version has one target stack, while the multiple blocks version requires coordination across stacks and longer planning horizons. In the *basic* condition, the agent has complete knowledge of the initial inventory; in the *perturbed* condition, the inventory is unknown and requires inspection before planning. This task tests reasoning under uncertainty during precise, goal-directed actions.

5 Empirical Results

This section details the empirical evaluation of the proposed InfoSeeker in partially observable environments. Our primary objectives are to: (1) assess its efficacy and generalization under uncertainty in observations (Sec. 5.2); (2) quantify its performance under uncertainty in both observations and dynamics (Sec. 5.3); (3) evaluate the efficiency of iterative planning (Sec. 5.5); and (4) distinguish the contributions of each method component through ablation studies (Sec. 5.6). We benchmark InfoSeeker against prior LLM-based planning methods using both our proposed tasks and two established benchmarks: LLM3 (Wang et al., 2024) and TravelPlanner (Xie et al., 2024).

5.1 Experimental Setup

Experiments were conducted across the 11 category variants in our benchmark. These categories are designed to evaluate iterative planning capabilities in partially observable environments (POMDPs). We use success rate—the percentage of tasks successfully completed within each variant under a limit of 100 interaction steps per task—as the evaluation metric, where an interaction step corresponds to a single action taken by the agent in the environment.

We compare InfoSeeker against LLM planning baselines (ReAct (Yao et al., 2023b), Reflexion (Shinn et al., 2023), AdaPlanner (Sun et al., 2023), LLM3 (Wang et al., 2024) with its *backtrack* and *from-scratch* variants) and reasoning-only methods (CoT-SC (Wang et al., 2022b) and ToT (Yao et al., 2023a)) applied on top of the *backtrack* version of LLM3.

To evaluate robustness and generalization, methods were tested across three LLMs: Gemini 2.0 Flash (AI, 2025), Gemini 2.5 Flash (Comanici et al., 2025) and GPT-4o (Hurst et al., 2024)[1]. Results

[1]All experiments use the same machine, the official APIs and default hyperparameters from OpenAI and Google.

in the body use Gemini 2.0 Flash and 2.5 Flash; GPT-4o results are reported in Appendix H and exhibit the same trends. We refer to Appendix K for further implementation details and prompts.

5.2 Performance under uncertainty in observations

We evaluated InfoSeeker under uncertainty in observations. We test on the *basic* setting of our proposed benchmark, as well as two established benchmarks: **LLM3** and **TravelPlanner**.

As shown in Table 1, InfoSeeker is competitive with strong baselines in the *basic* setup. All baselines perform well here, since they have been evaluated on larger benchmarks; in contrast, our method is general and achieves consistent gains across most tasks. In the *mix color* task, InfoSeeker reached 80%, an 4% absolute gain over the strongest baseline among the methods tested.

We additionally evaluate InfoSeeker on two established benchmarks. First, we test on a robotic manipulation benchmark introduced by LLM3, where agents control a robotic arm in a physical simulation. This task emphasizes challenges in spatial reasoning. To ensure a fair comparison, we follow the original evaluation protocol, allowing up to 20 planning attempts per task. As shown in Table 2, InfoSeeker consistently outperforms LLM3 baselines across all setups. Notably, in the most difficult scenario (*Large* basket size in Setting 2) InfoSeeker achieves a 20% absolute performance gain. Next, we test on the TravelPlanner benchmark, where web agents gather information based on user preferences to propose travel plans. As shown in Table 3, InfoSeeker surpasses all baselines in final pass rate. In particular, our method can identify ambiguities in user instructions. For example, it tries to clarify whether Washington refers to Washington state or Washington, D.C., while other methods fail to identify this ambiguity. These results demonstrate InfoSeeker's strong generalization capabilities in more realistic and challenging use cases.

5.3 Performance under uncertainty in observations and dynamics

InfoSeeker demonstrates strong performance under uncertain environmental dynamics that may be inconsistent with the agents' assumptions. We test on the *perturbed* settings of our benchmark, and as shown in Table 1, InfoSeeker consistently outperforms all baselines. On *robot arm control* with a

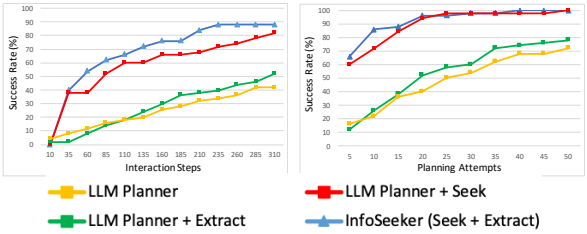


Figure 4: **Ablation study** on *perturbed stack single block*: success rate (%) vs. interaction steps (left) and planning attempts (right).

miscalibrated controller, InfoSeeker achieves an 80% success rate while the best baseline drops from 100% to 6%. The gains also transfer to two external benchmarks (LLM3, TravelPlanner) not designed to favor information-seeking strategies, indicating that our perturbed conditions are not artificially aligned with the method's inductive bias; see Appendix D for visualized trajectories.

5.4 Information seeking vs. explicit perturbation prompts

A natural question is whether existing planners could match InfoSeeker if simply *told* what is perturbed. We add a hand-written perturbation description (Listing 1) to each baseline prompt and evaluate across all 11 variants on the strongest production model (Gemini 2.5 Flash) and the latest Gemini 3.0 Flash. As shown in Table 4, the augmented baselines still fail to match InfoSeeker on most perturbed tasks; InfoSeeker – which receives no hints and discovers the dynamics purely through interaction – matches or outperforms them, including on Gemini 3.0 Flash. The gap is bridged by planned information seeking, not by prompt engineering or larger LLM backbones.

5.5 Model efficiency

Although information-seeking behaviors require extra environment interactions, they let InfoSeeker diagnose failures and generate better plans, yielding overall time- and token-efficiency gains. Table 5 shows that under a fixed one-minute wall-clock budget on *stack single block*, InfoSeeker surpasses all baselines and outperforms LLM3 by 36% in the *perturbed* setting while using a similar number of tokens. Additional results across step and planning budgets are in Appendix E.

5.6 Ablation Studies

We ablate InfoSeeker's components on the *perturbed stack single block* task. As shown in the

	robot arm control		mix color			robot navigation		stack multiple blocks		stack single block	
	basic	perturbed	basic	contaminated	wronglabel	basic	perturbed	basic	perturbed	basic	perturbed
Gemini 2.0 Flash											
ReAct	48	2	48	32	2	**100**	18	30	6	68	22
Reflexion	24	16	58	46	4	**100**	8	48	22	70	34
AdaPlanner	12	0	40	16	2	76	0	14	2	14	0
LLM3 (backtrack)	**100**	6	64	60	6	96	4	50	26	76	6
LLM3 (from scratch)	98	2	54	44	4	**100**	4	42	36	**86**	0
CoT-SC	10	14	54	50	4	96	4	50	40	**86**	4
ToT	30	34	60	54	4	96	8	**58**	**38**	**86**	2
InfoSeeker (ours)	**100**	**80**	**76**	**80**	**14**	**100**	**46**	42	34	82	**62**
Gemini 2.5 Flash											
ReAct	0	2	42	28	6	98	34	38	32	78	28
Reflexion	20	40	54	58	6	**100**	24	74	44	**100**	96
AdaPlanner	20	0	40	32	6	36	0	22	30	14	28
LLM3 (backtrack)	8	18	74	74	22	**100**	6	**100**	**100**	**100**	96
LLM3 (from scratch)	10	10	72	72	12	**100**	24	**100**	**100**	**100**	**100**
CoT-SC	4	4	72	72	5	**100**	8	**100**	**100**	**100**	**100**
ToT	4	10	76	64	10	**100**	4	**100**	**100**	**100**	**100**
InfoSeeker (ours)	**76**	**82**	**80**	**82**	**40**	**100**	**100**	98	98	**100**	**100**

Table 1: **Quantitative results on the proposed benchmark.** Success rate (%) within 100 interaction steps per task on Gemini 2.0 Flash and 2.5 Flash. InfoSeeker consistently outperforms baselines, particularly under perturbed conditions. GPT-4o results are reported in Appendix H.

left panel of Figure 4, the vanilla LLM planner achieves 42% success after 310 interaction steps. Extracting information from previous task-oriented trajectories (*Extract*) yields only marginal gain, while prompting for explicit information seeking (*Seek*) boosts success to 82%. InfoSeeker, combining both, reaches 72% in just 135 steps—nearly halving the steps *Seek* needs. The right panel shows the same trend; doubling LLM3's trace budget does not close the gap (Appendix J).

The above holds the LLM backbone fixed. To rule out that the information-extraction (IE) module drives the gains, we swap each module's backbone independently between weak (Gemini 2.0 Flash) and strong (Gemini 3.0 Flash) on the *mix color* task. Table 6 shows that strong IS with weak IE matches the all-strong configuration on the hardest *wronglabel* perturbation (48%), whereas strong IE with weak IS does not (24%): information seeking, not extraction quality, drives InfoSeeker's gains.

	Setting 1			Setting 2		
	Easy	Medium	Hard	Small	Medium	Large
LLM3 (backtrack)	**100**	80	50	50	80	50
LLM3 (from scratch)	**100**	**100**	**80**	**80**	90	60
InfoSeeker (ours)	**100**	**100**	70	70	**100**	**80**

Table 2: **LLM3 benchmark (Wang et al., 2024).** Setting 1 varies object placement difficulty; Setting 2 fixes objects at the hardest level and varies basket size.

	Final Pass Rate (↑)
ReAct (using original prompt)	5.56
LLM3 (backtrack)	5.00
LLM3 (from scratch)	5.56
InfoSeeker (ours)	**6.11**

Table 3: **TravelPlanner benchmark (Xie et al., 2024).** Evaluated with a fixed budget of 100 interaction steps.

6 Related Work

6.1 LLM Planning Agents

Foundational works like Inner Monologue (Huang et al., 2022) and ReAct (Yao et al., 2023b) demonstrated that LLMs could process execution feedback and scene descriptions to revise plans. Subsequent methods, including ProgPrompt (Singh et al., 2023) and AdaPlanner (Sun et al., 2023), generate plans in programming languages and have been applied to embodied agents (Wang et al., 2023a,b; Zhu et al., 2023) and robotic systems (Wang et al., 2024; Ding et al., 2023; Chen et al., 2024; Joublin et al., 2024). Beyond planning, several studies investigate how LLMs gather and reason about information (Ke et al., 2024; Krishnamurthy et al., 2024; Pan et al., 2025; Piriyakulkij et al., 2024), identify missing instructions (Huang et al., 2024; Sun et al., 2024), or query humans (Li et al., 2023; Chen et al., 2023; Ren et al., 2023); Ren et al. (2023) in particular model the agent's epistemic uncertainty to trigger human queries, whereas InfoSeeker plans information-seeking *actions* that

	robot arm control		mix color			robot navigation		stack multiple blocks		stack single block	
	basic	perturbed	basic	contaminated	wronglabel	basic	perturbed	basic	perturbed	basic	perturbed
Gemini 2.5 Flash											
Reflexion	48	48	58	60	16	**100**	42	76	50	98	78
LLM3 (backtrack)	**100**	94	76	74	28	**100**	38	**100**	**100**	**100**	**100**
LLM3 (from scratch)	**100**	94	74	76	40	**100**	32	**100**	**100**	**100**	**100**
ToT	**100**	**98**	68	72	**44**	**100**	30	**100**	98	**100**	**100**
InfoSeeker *(no perturbation prompt)*	76	82	**80**	**82**	40	**100**	**100**	98	98	**100**	**100**
Gemini 3.0 Flash											
Reflexion	94	88	88	84	64	**100**	**100**	**100**	98	98	**100**
LLM3 (backtrack)	**100**	**100**	88	88	64	98	68	**100**	**100**	**100**	**100**
LLM3 (from scratch)	**100**	**100**	**92**	**92**	**74**	**100**	96	**100**	**100**	**100**	**100**
ToT	**100**	98	80	76	46	**100**	**100**	**100**	**100**	**100**	**100**
InfoSeeker *(no perturbation prompt)*	**100**	**100**	90	84	48	**100**	**100**	**100**	**100**	**100**	**100**

Table 4: **Perturbation prompts fail to close the gap with InfoSeeker.** Baselines augmented with a description of the perturbation, evaluated on Gemini 2.5 Flash and Gemini 3.0 Flash. InfoSeeker receives no such hints.

	Time eff.		Token eff.			
	Basic	Pert.	Basic		Pert.	
Gemini 2.0 Flash	SR	SR	SR	TC	SR	TC
ReAct	68	30	68	139	22	100
Reflexion	58	30	70	81	34	103
AdaPlanner	10	0	14	323	0	349
LLM3 (backtrack)	98	46	76	3	6	13
LLM3 (from scratch)	98	62	**86**	**2**	0	**12**
CoT-SC	94	0	**86**	118	4	48
ToT	86	8	**86**	12	2	46
InfoSeeker (ours)	**100**	**98**	82	9	**62**	13

Table 5: **Time and token efficiency of InfoSeeker** on the *stack single block* task (SR%, TC in k tokens). Under a one-minute wall-clock budget, InfoSeeker outperforms LLM3 by 36% while keeping low TC.

	basic	contam.	wronglab.
Weak IS + Weak IE	76	80	14
Weak IS + Strong IE	**92**	**90**	24
Strong IS + Weak IE	90	88	**48**
Strong IS + Strong IE	90	84	**48**

Table 6: **Information-seeking (IS) and information-extraction (IE) backbone swap on *mix color*.** Weak = Gemini 2.0 Flash, Strong = Gemini 3.0 Flash.

update internal dynamics from environment interaction. Memory-augmented agents (Shinn et al., 2023; Zhao et al., 2024; Sarch et al., 2023; Song et al., 2024; Sarch et al., 2024) replay past trajectories but assume static dynamics. Active information gathering under uncertainty is a long-standing principle in classical POMDP planning (Kaelbling et al., 1998; Sondik, 1978); our contribution is to operationalize it for zero-shot LLM planners that otherwise rely on reactive replanning.

6.2 Evaluation in Interactive Environments

A variety of benchmarks have been developed to evaluate the planning and reasoning capabilities of LLMs. Valmeekam et al. (2023) focus on structured reasoning tasks, while text-based environments (Shridhar et al., 2020; Côté et al., 2019; Chevalier-Boisvert et al., 2023) assess decision-making under incomplete observation. Long-horizon and open-ended reasoning are evaluated in complex simulations such as game environments (Fan et al., 2022; Küttler et al., 2020), scientific discovery (Wang et al., 2022a; Jansen et al., 2024), and web navigation (Yao et al., 2022; Deng et al., 2023; Xie et al., 2024). Although many of these benchmarks involve partial observability, they typically focus solely on uncertainty in observations. In contrast, our benchmark captures more realistic conditions: observations are incomplete, and environmental dynamics are noisy.

7 Conclusion

We propose InfoSeeker, an LLM decision-making framework that interleaves explicit information seeking with task-oriented planning to adapt internal dynamics under noisy environments, together with a text-based benchmark for partial observability and uncertain dynamics. Experiments show that InfoSeeker outperforms baselines and generalizes to established benchmarks, showing that information seeking improves planning under uncertainty.

Future Work. Since current LLMs are not explicitly trained to reason over probabilistic transitions, incorporating Bayesian frameworks for stochastic modeling in InfoSeeker is a promising direction. LLMs also have limited capacity to reason over the continuous dynamics common in real-world planning, so integrating LLM-based planning with low-level continuous control is an important next step for embodied agents in real-world deployments.

Limitations

Our benchmark is small-scale and hand-crafted; to our knowledge no existing benchmark evaluates robustness under uncertain environmental dynamics, and building a larger, more rigorous one is an important direction for future work. Active information gathering under uncertainty is a long-standing principle in classical POMDP and decision-theoretic literature (Kaelbling et al., 1998; Sondik, 1978); our contribution operationalizes it for zero-shot LLM planners rather than introducing the principle.

References

Google AI. 2025. Gemini 2.0 flash. Available at https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash.

Donald O Case and Lisa M Given. 2016. Looking for information: A survey of research on information seeking, needs, and behavior.

Xiaoyu Chen, Shenao Zhang, Pushi Zhang, Li Zhao, and Jianyu Chen. 2023. Asking before acting: Gather information in embodied decision making with language models. *arXiv preprint arXiv:2305.15695*.

Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2024. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In *2024 IEEE International conference on robotics and automation (ICRA)*, pages 6695–6702. IEEE.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. 2023. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36:73383–73394.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, and 1 others. 2019. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. 2023. Task and motion planning with large language models for object rearrangement. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2086–2092. IEEE.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362.

Alison Gopnik. 2012. Scientific thinking in young children: Theoretical advances, empirical research, and policy implications. *Science*, 337(6102):1623–1627.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. 2023. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning. *arXiv preprint arXiv:2311.17842*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, and 1 others. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Wese: Weak exploration to strong exploitation for llm agents. *arXiv preprint arXiv:2404.07456*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. 2024. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. *Advances in Neural Information Processing Systems*, 37:10088–10116.

Frank Joublin, Antonello Ceravola, Pavel Smirnov, Felix Ocker, Joerg Deigmoeller, Anna Belardinelli, Chao Wang, Stephan Hasler, Daniel Tanneberg, and

Michael Gienger. 2024. Copal: corrective planning of robot actions with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8664–8670. IEEE.

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

Nan Rosemary Ke, Danny P Sawyer, Hubert Soyer, Martin Engelcke, David P Reichert, Drew A Hudson, John Reid, Alexander Lerchner, Danilo Jimenez Rezende, Timothy P Lillicrap, and 1 others. 2024. Can foundation models actively gather information in interactive environments to test hypotheses? *arXiv preprint arXiv:2412.06438*.

Celeste Kidd and Benjamin Y Hayden. 2015. The psychology and neuroscience of curiosity. *Neuron*, 88(3):449–460.

Akshay Krishnamurthy, Keegan Harris, Dylan J Foster, Cyril Zhang, and Aleksandrs Slivkins. 2024. Can large language models explore in-context? *arXiv preprint arXiv:2403.15371*.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. 2020. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684.

Boyi Li, Philipp Wu, Pieter Abbeel, and Jitendra Malik. 2023. Interactive task planning with language models. *arXiv preprint arXiv:2310.10645*.

Lan Pan, Hanbo Xie, and Robert C Wilson. 2025. Large language models think too fast to explore effectively. *arXiv preprint arXiv:2501.18009*.

Top Piriyakulkij, Cassidy Langenfeld, Tuan Anh Le, and Kevin Ellis. 2024. Doing experiments and revising rules with natural language and probabilistic reasoning. *Advances in Neural Information Processing Systems*, 37:53102–53137.

Allen Z Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, and 1 others. 2023. Robots that ask for help: Uncertainty alignment for large language model planners. *arXiv preprint arXiv:2307.01928*.

Gabriel Sarch, Lawrence Jang, Michael J Tarr, William W Cohen, Kenneth Marino, and Katerina Fragkiadaki. 2024. Vlm agents generate their own memories: Distilling experience into embodied programs. *arXiv preprint arXiv:2406.14596*.

Gabriel Sarch, Yue Wu, Michael J Tarr, and Katerina Fragkiadaki. 2023. Open-ended instructable embodied agents with memory-augmented large language models. *arXiv preprint arXiv:2310.15127*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

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

Edward J Sondik. 1978. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245.

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

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36:38975–38987.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022a. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*.

Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. 2024. Llm^ 3: Large language model-based task and motion planning with motion failure reasoning. In

2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 12086–12092. IEEE.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022b. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, and 1 others. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

A Declaration of LLM Usage

We used large language models (LLMs) to assist with writing refinement (e.g., grammar, spelling, word choice) and to support the design and execution of experiments. Our paper proposes a novel LLM-based framework for autonomous decision planning, making LLM a central component of our methodology; this framework is detailed in Section 3 and Algorithm 1. The specific LLMs used are listed in Section 5.1.

B Reproducibility Statement

The complete implementation of our proposed InfoSeeker along with the benchmarks is included in the supplementary material. While the implementations of baseline methods and ablation experiments are not included, we provide the corresponding prompts used with LLMs in Appendix K.

C InfoSeeker Algorithm

Algorithm 1 InfoSeeker (InfoSeeker)

Input: a LLM f, task instruction l, prompt for information seeking p_s, information extraction p_e, and task-oriented planning p_t, maximum number of planning attempts N_{max}, maximum number of interaction steps K_{max}, environment e
Output: a sequence of task-oriented actions A_t
Initiate interaction history $H \leftarrow \{\}$ and the total number of interaction steps $K \leftarrow 0$
for planning attempt $n = 1$ to N_{max} **do**
  Generate actions for information seeking $A_s = f(p_s, l, H)$
  Execute actions and append new observations to history $H \leftarrow H \cup \{(a, e(a)) \mid a \in A_s\}$
  Update the total number of interaction steps $K \leftarrow K + |A_s|$
  Extract information $I = f(p_e, l, H)$
  Generate task-oriented plans $A_t = f(p_t, l, H, I)$
  Reset interaction history $H \leftarrow \{\}$
  Execute actions and append feedback to history $H \leftarrow H \cup \{(a, e(a)) \mid a \in A_t\}$
  Update the total number of interaction steps $K \leftarrow K + |A_t|$
  if success **or** $K \geq K_{max}$ **then**
    break
  end if
end for
Return A_t

D Planning Visualization

The adaptive planning process is illustrated in Figure 5, comparing the behaviors of InfoSeeker and LLM3 when planning *from scratch*. While LLM3 repeatedly applies an invalid reasoning pattern—such as attempting to move "forward then right," even though this consistently results in the

robot moving "backward then left" (Figure 5c, bottom row)—InfoSeeker systematically explores alternative actions and adjusts its strategy accordingly. For example, it first probes all four directions to map actions to their actual outcomes, then uses this knowledge to adapt effectively on subsequent attempts (see Figure 5c, top row). This process demonstrates InfoSeeker's ability to actively acquire informative observations and adapt its internal dynamics to align with the environment.

E Performance with Varying Interaction Steps and Planning Attempts

We evaluated InfoSeeker's performance in the *stack single block* task by measuring its success rate under varying interaction steps and planning attempts (Figure 6). The results demonstrate that InfoSeeker effectively leverages increased resources, particularly in perturbed environments. Figure 6(left) shows the impact of interaction steps; increasing interaction steps in the perturbed setting leads to a substantial performance gain—from 0% at 10 steps to 88% at 235 steps. Notably, this success rate matches the best-performing baseline (LLM3 *from scratch*) in the much easier *basic setting*. A similar trend appears when varying planning attempts (Figure 6, right). InfoSeeker achieves a 66% success rate with just 5 attempts in the perturbed task, significantly outperforming leading baselines LLM3 *from scratch* and *backtrack*, which reach just 4%. As planning attempts increase, InfoSeeker continues to improve, reaching 100% success after 40 attempts. These findings highlight InfoSeeker's strong adaptability through active information seeking and the adaptation of its internal dynamics, enabling robust performance under uncertainty in both observations and environmental dynamics.

F Vanilla LLM Planner with In-Context Learning

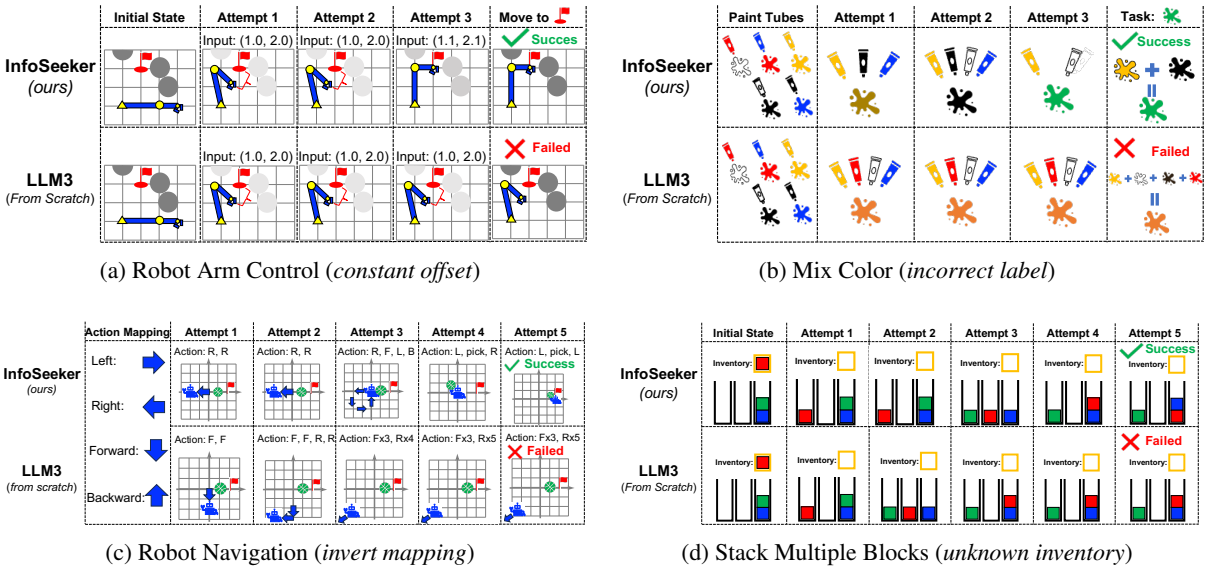
We carried out experiments on the *mix color* task, where a vanilla LLM planner was guided using in-context demonstrations of successful InfoSeeker execution traces in perturbed environments. As shown in Table 7, the in-context examples only bring marginal performance gain to the planner. They do not induce effective information seeking behaviors. In particular, we observed that the planner fails to follow the provided examples and cannot align its internal dynamics with the environment, as evidenced by its inability to detect incorrect paint labels. Our results demonstrate that abstracting information seeking behaviors from in-context examples is challenging.

	Basic	Contaminated	Wrong Label
Vanilla Planner	52	46	6
In-Context Learning	58	56	2
InfoSeeker (ours)	**76**	**80**	**14**

Table 7: **In-context learning fails to induce information-seeking behavior.** We evaluate the *mix color* task, where a planner is guided by successful InfoSeeker demonstrations in perturbed environments. In-context examples yield only marginal gains, indicating that information-seeking cannot be effectively learned through in-context learning.

G Failure Analysis

To understand the limitations of InfoSeeker, we present an analysis of failure cases in *block stacking* tasks. We categorize failure cases into four types: (1) **Information Seeking**, where the agent fails to acquire informative observations needed to detect the mismatches between its internal dynamics and the environment; (2) **Information Extraction**, where the agent extracts misleading or irrelevant information that misguides the planner; (3) **Instruction Understanding**, where the agent misinterprets user instructions (e.g., confusing "red on top of blue" with "blue on top of red"); and (4) **Long-horizon Planning**, where the agent detects a mismatch between internal and environmental dynamics during the information seeking phase, but fails to produce effective actions to complete the task. As shown in Figure 7, extracting incorrect information is the major failure type of InfoSeeker, especially in the *stack multiple blocks* task. How to enhance the quality of information extraction is an interesting direction for future work.



(a) Robot Arm Control (*constant offset*)

(b) Mix Color (*incorrect label*)

(c) Robot Navigation (*invert mapping*)

(d) Stack Multiple Blocks (*unknown inventory*)

Figure 5: **Visualization of InfoSeeker and LLM3 (*from scratch*) in perturbed environments. (a)** Robotic arm guidance to target $(1.0, 2.0)$ with a fixed action offset $(-0.1, -0.1)$. **(b)** Paint mixing for seagreen (yellow + black) using mislabeled tubes: the "red" tube contains white, "blue" contains red, "white" contains black, and "black" contains blue. **(c)** Navigate to ball at $(1, 0)$ and deliver it to goal location $(2, 0)$ under inverted action mappings ("left" moves right, "forward" moves backward). **(d)** Rearranging blocks to match a target configuration, starting with a hidden inventory (contains a red block). Across these perturbations, InfoSeeker adapts through information seeking and feedback, while LLM3 repeatedly fails due to persistent misinterpretation.

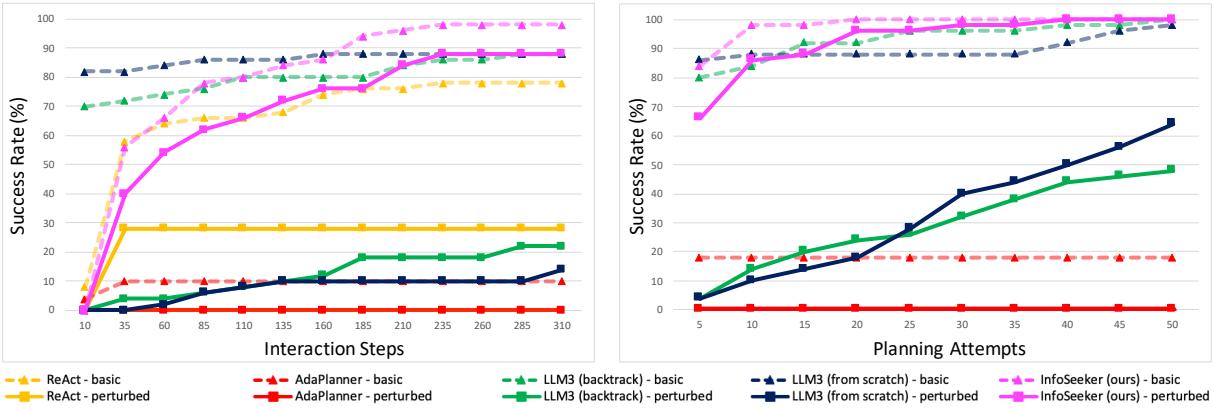


Figure 6: **Performance with varying interaction steps (*left*) and planning attempts (*right*).** The plots show the success rate (%) of InfoSeeker and four LLM baselines (Yao et al., 2023b; Sun et al., 2023; Wang et al., 2024) on the stack single block task, as interaction steps (*left*) and planning attempts (*right*) are varied. ReAct is included only in the interaction step analysis due to its lack of distinct planning attempts. The plots demonstrate that InfoSeeker effectively leverages increased resources, particularly in perturbed environments.

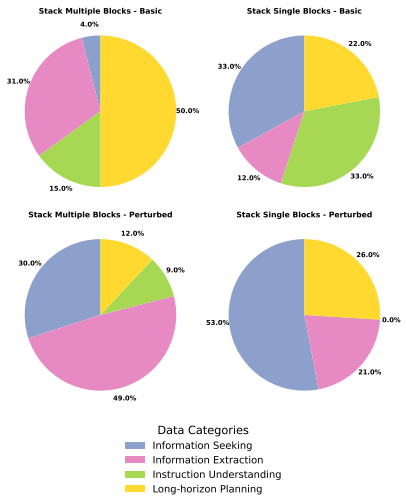


Figure 7: **Failure Analysis.** Extracting incorrect infor-

H GPT-4o Results on the Proposed Benchmark

Table 8 reports per-task success rates on the proposed benchmark using GPT-4o as the LLM backbone for all methods. The trends mirror those observed for Gemini 2.0 Flash and 2.5 Flash in Table 1: baselines remain competitive on *basic* variants but collapse under perturbation, while InfoSeeker maintains substantially higher success rates on the perturbed settings. This confirms that the

effectiveness of explicit information seeking generalizes across model families and is not specific to Gemini.

I Probe-Then-Plan Intermediate Baseline

A natural intermediate baseline between a vanilla plan-and-replan agent and InfoSeeker is *probe-then-plan*, which performs a single round of information-seeking actions at the start of each task and then commits to plan-and-replan without further probing. Table 9 compares the vanilla LLM planner, probe-then-plan, and InfoSeeker on the *robot arm control* and *stack single block* tasks using Gemini 2.0 Flash. Probe-then-plan offers only modest gains over the vanilla planner – and is sometimes worse on the basic setting – while InfoSeeker dominates both, especially in the perturbed conditions where the environmental dynamics may shift mid-trajectory. This confirms that information seeking is most effective when interleaved with task-oriented planning across every decision cycle, rather than performed as a one-shot probe at task start.

	robot arm control		stack single block	
	basic	perturbed	basic	perturbed
Vanilla LLM planner	76	6	78	18
Probe-Then-Plan	82	8	70	48
InfoSeeker (ours)	**100**	**80**	**82**	**62**

Table 9: **Probe-then-plan is not enough.** A baseline that performs information seeking only once at the start of a task (*Probe-Then-Plan*) underperforms InfoSeeker, especially in perturbed settings. Information seeking must be interleaved with task-oriented planning across the decision cycle, not executed as a single up-front probe. Gemini 2.0 Flash.

J LLM3 Reasoning-Trace Sensitivity

By default, LLM3 conditions on the five most recent planning traces. One might worry that the gap between LLM3 and InfoSeeker in Table 1 is an artifact of this hyperparameter choice. We re-run both LLM3 variants with ten traces across all benchmark variants using Gemini 2.0 Flash and report results in Table 10. Doubling the trace budget does not close the gap – it yields at most marginal change on the perturbed settings – indicating that the limitation of LLM3 in these tasks stems from the lack of explicit information-seeking actions rather than from insufficient context.

K Prompts and Implementation Details

ReAct and AdaPlanner were evaluated using few-shot prompting, in line with their original publications, while both LLM3 variants and InfoSeeker used zero-shot prompting. We used API-default hyperparameters (e.g., temperature, top_p) for all experiments to ensure comparability among methods, though performance may improve with dedicated hyperparameter tuning. Specifically, the prompts are presented in Appendices K.1 (InfoSeeker), K.2 (ReAct), K.3 (AdaPlanner), K.4 (LLM3), K.5 (Reflexion), K.6 (CoT-SC), K.7 (ToT), and K.8 (In-Context Learning). To add description of environmental uncertainty to baseline prompts we use the same words as InfoSeeker in Listing 1.

```
You are an AI robot that generate a plan
    of actions to reach the goal.
You will be given a domain description
    and the trace of generated plans
    that lead to motion failure.
Previous actions have not completed the
    goal, possibly due to
    misunderstandings of the environment
    , insufficient exploration, or flaws
    in your earlier plan.
```

Listing 1: Prompt for Uncertainty

	robot arm control		mix color			robot navigation		stack multiple blocks		stack single block	
	basic	perturbed	basic	contaminated	wronglabel	basic	perturbed	basic	perturbed	basic	perturbed
ReAct	22	2	42	34	6	90	4	18	10	54	20
AdaPlanner	80	2	38	24	2	82	0	16	12	22	10
LLM3 (backtrack)	**100**	12	74	76	10	80	6	38	30	80	4
LLM3 (from scratch)	**100**	6	76	72	6	**96**	8	**48**	**50**	56	4
InfoSeeker (ours)	**100**	**22**	**84**	**78**	**36**	92	**44**	38	26	**84**	**34**

Table 8: **Quantitative results on GPT-4o.** Success rate (%) within 100 interaction steps per task. InfoSeeker outperforms baselines on most variants, with the largest gains on the perturbed settings.

	mix color			robot navigation		stack multiple blocks		stack single block		robot arm control	
	basic	contaminated	wronglabel	basic	perturbed	basic	perturbed	basic	perturbed	basic	perturbed
LLM3 (backtrack, traces=5)	64	60	6	96	4	50	26	76	6	**100**	6
LLM3 (backtrack, traces=10)	64	58	4	92	2	50	30	82	6	98	6
LLM3 (from scratch, traces=5)	54	44	4	**100**	4	42	**36**	**86**	0	98	2
LLM3 (from scratch, traces=10)	46	42	4	**100**	4	46	**36**	84	10	98	2
InfoSeeker (ours)	**76**	**80**	**14**	**100**	**46**	42	34	82	**62**	**100**	**80**

Table 10: **Increasing the number of LLM3 reasoning traces does not close the gap.** We rerun both LLM3 variants with 10 reasoning traces (instead of the default 5 used in Table 1) across all benchmark variants. The extra traces yield at most marginal improvement and InfoSeeker still outperforms LLM3 on the perturbed settings. Gemini 2.0 Flash.

K.1 InfoSeeker Prompts

```
You are an AI tasked with verifying your understanding of a given environment and exploring it for additional information.
Based on the provided domain description, generate a plan to ensure your understanding is aligned with the real environment and how you will explore it for further details.
Try to verifying your understanding thoroughly, and explore beyond the goal for unexpected solution.
You are expected to learn from the environment only, do not try to achieve the goal.

Important: You have VERY FEW turns left. Choose your next action carefully to maximize information.

# Domain Description
{domain_desc}

Please provide the following output:
1. **Reasoning**: Explain the strategy you will use to verify your understanding of the environment and explore it for new information.
2. **Steps**: Provide a detailed series of steps you will take to verify and explore the environment. Make sure to reset and clean the environment at the end of each step to avoid any potential interference. Each step should include:
   - **Goal**: The specific objective or target for that step (what you aim to achieve).
   - **Action Plan**: A list of actions that you will perform to accomplish the goal of that step.

Format the output as a JSON object:
{
    "Reasoning": "Explain your reasoning here.",
    "Steps": [
        {
            "Goal": "The goal of your verification or exploration",
            "Action Plan": ["action1", "action2", "action3", ...]
        },
        ...
    ]
}
```

Listing 2: Information-seeking prompt for InfoSeeker in our benchmark when no prior plan exists

```
You are an AI tasked with generating a plan to achieve a given goal.
You will be given a domain description and a history of past actions that have not yet achieved the goal.
You must analyze the interaction history, understand the environment mechanics, and determine why the goal has not been reached.
Using this insight, create a new plan to successfully achieve the goal.

# Domain Description
{domain_desc}

```

```
# Interaction History
{interaction_history}

# Information
{information}

Please provide the following output:
1. **Reasoning**: Describe the strategy you will use to achieve the goal, taking into account the past actions and information in the interaction history. Consider how the environment's response to these actions affects your new plan.
2. **Solution Plan**: List the series of actions you will take to achieve the goal, ensuring that your actions align with the reasoning you provided. Remember to check for the final result.

Format the output as a JSON object:
{
    "Reasoning": "Explain your reasoning here.",
    "Solution Plan": ["action1", "action2", "action3", ...]
}
```

Listing 3: Task-Oriented Planning prompt for InfoSeeker in our benchmark

```
You are an AI tasked with revising your plan to achieve a goal.
Previous actions have not completed the goal, possibly due to misunderstandings of the environment, insufficient exploration, or flaws in your earlier plan.
You can conduct simple tests to identify issues with the past approach, verify your understanding, and explore the environment for more information if needed.
Using the provided domain description and interaction history, generate an action plan to verify and explore the environment, identifying flaws in previous actions.
Try to verify the failures by simple unit tests, or explore beyond past plans for unexpected solution.

Important: You have VERY FEW turns left. Choose your next action carefully to maximize information.

# Domain Description
{domain_desc}

# Interaction History
{interaction_history}

Please provide the following output:
1. **Reasoning**: Explain your strategy for verifying your understanding of the environment, exploring it for new insights, and addressing issues in the previous plan.
2. **Steps**: Provide a detailed series of steps you will take to verify and explore the environment. Make sure to reset and clean the environment at the end of each step to avoid any potential interference. Each step should include:
   - **Goal**: The specific objective or target for that step (what you aim to achieve).
   - **Action Plan**: A list of actions that you will perform to accomplish the goal of that step.

Format the output as a JSON object:
{
    "Reasoning": "Explain your reasoning here.",
    "Steps": [
        {
            "Goal": "The goal of your verification or exploration",
            "Action Plan": ["action1", "action2", "action3", ...]
        },
        ...
    ]
}
```

Listing 4: Information Seeking prompt for InfoSeeker in our benchmark when prior plan exists

K.2 ReAct Prompt

```
Interact with materials to solve a task. Here are two examples.
# Example 1
Your task is to: move the gripper to [1.0, 0.0], while avoid collision with the obstacles at, [2.0, -2.0] radius 0.5
Initial State:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]
> think: To solve the task, I need to first check the action options.
OK.
> help
1) Move x y: Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics
2) Check: Check Robot arm joint positions
3) Help: View the available action options.
4) think: <your_thought>: Output your thinking and reasoning.
5) End: End the task when you finished, or the task is not solvable.
> think: To solve the task, I need to move gripper to [1.0, 0.0].
OK.
> Move 1.0 0.0
Success!
> Check
```

```
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [1.00, 0.00]
> think: Now the gripper is at [1.0, 0.0], the task is complete.
OK.
> End
# Now it's your turn
Your task is to: move the gripper to [1.0, 0.0], while avoid collision with the obstacles at, [0.0, 3.0] radius 0.5, [2.0, 2.0] radius 0.5, [-2.0, 2.0] radius 0.5, [2.5, 1.0] radius 0.5, [-2.5, 1.0] radius 0.5, [0.0, -3.0] radius 0.5, [2.0, -2.0] radius 0.5, [-2.0, -2.0] radius 0.5, [2.5, -1.0] radius 0.5, [-2.5, -1.0] radius 0.5
Initial State:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]
> think: To solve the task, I need to first check the action options.
OK.
> help
1) Move x y: Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics
2) Check: Check Robot arm joint positions
3) Help: View the available action options.
4) think: <your_thought>: Output your thinking and reasoning.
5) End: End the task when you finished, or the task is not solvable.
> think: To solve the task, I need to move the gripper to [1.0, 0.0] while ensuring it does not collide with any obstacles. The initial position of the gripper is [3.0, 0.0]. I will perform the move operation to see if it can directly reach the target without colliding.
OK.
> Move 1.0 0.0
Failed! Collision detected along the path. Move aborted.
> think: Since a direct move to [1.0, 0.0] results in a collision, I need to plan an alternative path to avoid obstacles. Considering the obstacles' positions and their radii, a potential solution is to move the gripper around the obstacles with minimal path adjustments.
OK.
> Move 2.0 0.0
Success!
> Check
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [1.75, -0.97] 'Gripper': [2.00, 0.00]
> think: With the gripper now at [2.0, 0.0], the next step is to move closer to the target at [1.0, 0.0] while avoiding obstacles. Moving directly from [2.0, 0.0] to [1.0, 0.0] seems feasible without hitting any obstacles given their positions and radii. I will attempt this move.
OK.
> Move 1.0 0.0
Success!
```

Listing 5: Prompt for ReAct in the Robot Arm Control Tasks

K.3 AdaPlanner Prompt

```
basic_info = '''
# You are a household agent. Here is some Python code defining a household environment:

# Agent class represents the state of the agent,
# including what materials are available as well as the actions it can take.
class Agent:
    def __init__(self, joint0_pos, joint1_pos, gripper_pos):
        self.joint0_pos = joint0_pos
        self.joint1_pos = joint1_pos
        self.gripper_pos = gripper_pos

    # Here are the admissible actions the agent can take:

    # Check Robot arm joint positions.
    # For example, "Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [1.00, 0.00]" = check()
    def check(self):
        ...

    # Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics.
    # For example, 'Success!' = move(3.0, 0.0)
    # For example, 'Failed! Collision detected along the path. Move aborted.' = move(3.0, 0.0)
    # For example, 'Failed! Target is out of reach. Move aborted.' = move(3.0, 0.0)
    def move(self, x, y):
        ...
'''.strip()

get_solution_prompt = f'''
{basic_info}

# Now complete the function solution() below to solve the task by composing the agent's methods to interact with the environment.
# For each step you plan to take, 1) mark with '[Step xx]', 2) give a reason why you think it is a good step to take 3) write an assertion to check if the step is successful.

```

```
# Here is an example of a solution to the task:

<example>

# Here is the actual task.
# define environment and agent
joint0_pos = <joint0_pos_list>
joint1_pos = <joint1_pos_list>
gripper_pos = <gripper_pos_list>
agent = Agent(joint0_pos, joint1_pos, gripper_pos)

# The robot arm has two joints and a gripper, the goal is to <task> .
# You should complete your solution function below:
def solution(agent, start_from=1):
'''.strip()

simple_example = '''
# define environment and agent
joint0_pos = [0.00, 0.00]
joint1_pos = [2.00, 0.00]
gripper_pos = [3.00, 0.00]
agent = Agent(joint0_pos, joint1_pos, gripper_pos)

# Your task is to: move the gripper to [1.0, 0.0], while avoid collision with the obstacles at, [2.0, -2.0] radius 0.5.
# here is a solution:
def solution(agent, start_from=1):
    # General Plan: To solve the task, I need to move gripper to [1.0, 0.0].
    if start_from <= 1:
        # [Step 1] Move gripper from [3.0, 0.0] to [1.0, 0.0].
        # Move gripper to [1.0, 0.0].
        observation = agent.move(1.0, 0.0)
        # expectation: I should be able to move gripper to [1.0, 0.0].
        assert "Success" in observation, f'ERROR in [Step 1]. {observation}'

    if start_from <= 2:
        # [Step 2] Check if the gripper has been moved correctly.
        # Check the gripper position.
        observation = agent.check()
        # expectation: I should be able to see gripper position at [1.0, 0.0].
        assert "'Gripper': [1.00, 0.00]" in observation, f'ERROR in [Step 2]. {observation}'
'''.strip()

feedback_fix_prompt = '''
{basic_info}

# Here is a example of successful solution for solving a similar task:
[Successful example]
joint0_pos = [0.00, 0.00]
joint1_pos = [2.00, 0.00]
gripper_pos = [3.00, 0.00]
agent = Agent(joint0_pos, joint1_pos, gripper_pos)
<example>

# Here is the actual task.
# define environment and agent
joint0_pos = <joint0_pos_list>
joint1_pos = <joint1_pos_list>
gripper_pos = <gripper_pos_list>
agent = Agent(joint0_pos, joint1_pos, gripper_pos)

# The robot arm has two joints and a gripper, the goal is to <task> .
You have generated code of solution() to solve the task. However, you executed the solution() function and get an error message:
<error_msg>

Let's think step by step. Referring to the successful case and the error message, you should complete the solution function with the correct code.
def solution(agent, start_from=1):
'''.strip()
```

Listing 6: Prompt for AdaPlanner in the Robot Arm Control Tasks

K.4 LLM3 Prompt

```
You are an AI robot that generate a plan of actions to reach the goal. You will be given a domain description and the trace of generated plans that lead to motion failure.
You are expected to correct the plan incrementally (on top of the last plan) to avoid the motion failure. This may involve sample new parameters for the failed action, or reverse one or more succeeded actions for backtracking. Make your decision based on the trace provided.

The tabletop environment has a robot arm, several obstacles and a goal location.
The robot arm has two joints and a gripper, the goal is to  move the gripper to [1.0, 2.0], while avoid collision with the obstacles at, [0.0, 3.0] radius 0.5, [2.0, 2.0] radius 0.5, [-2.0, 2.0] radius 0.5, [2.5, 1.0] radius 0.5, [-2.5, 1.0] radius 0.5, [0.0, -3.0] radius 0.5, [2.0, -2.0] radius 0.5, [-2.0, -2.0] radius 0.5, [2.5, -1.0] radius 0.5, [-2.5, -1.0] radius 0.5 .

The robot arm has the following primitive actions:
1) Move x y:  Move the gripper to (x, y). The rotation of the joints will be
```

```
     calculated by inverse kinematics.
2) Check: Check Robot arm joint positions

The current environment state is:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]

The trace is:
No previous plan

Please generate output step-by-step, which includes:
1. Reasoning: Your reasoning for the failure of last plan if the last plan exists, and the strategy to accomplish the task goal. Make sure you account for the position of robot arm and obstacles. Try to answer the questions: (i) what is the cause of the failure of last plan? (ii) can altering action parameters for the failed action solve the problem? if yes, what feasible action parameters should we use? (iii) do we need to reverse one or more succeeded actions executed before the failed action? if yes, which actions should be reversed? (iv) if the task goal is not achieved, how can we revise the plan to achieve the goal?
2. Full Plan: The new full plan that you generate based on the last plan. Make sure you properly reflect the above reasoning in the new plan. The plan should be a full plan that includes all the actions from the beginning to the end.
Please organize the output following the json format below:
{
    "Reasoning": "My reasoning for the failure of last plan is ...",
    "Full Plan": ["pick A", "place B", "check", ...]
}
```

Listing 7: Prompt for LLM3 (backtrack) in the Robot Arm Control Tasks

```
You are an AI robot that generate a plan of actions to reach the goal. You will be given a domain description and the trace of generated plans that lead to motion failure. You are expected to generate a plan from scratch.

The tabletop environment has a robot arm, several obstacles and a goal location.
The robot arm has two joints and a gripper, the goal is to move the gripper to [1.0, 2.0], while avoid collision with the obstacles at, [0.0, 3.0] radius 0.5, [2.0, 2.0] radius 0.5, [-2.0, 2.0] radius 0.5, [2.5, 1.0] radius 0.5, [-2.5, 1.0] radius 0.5, [0.0, -3.0] radius 0.5, [2.0, -2.0] radius 0.5, [-2.0, -2.0] radius 0.5, [2.5, -1.0] radius 0.5, [-2.5, -1.0] radius 0.5 .

The robot arm has the following primitive actions:
1) Move x y:  Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics.
2) Check: Check Robot arm joint positions

The current environment state is:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]

The trace is:
No previous plan

Please generate output step-by-step, which includes:
1. Reasoning: Your reasoning for the failure of last plan if the last plan exists, and the strategy to generate a new plan from scratch to accomplish the task goal. Please be specific on the strategy, such as what actions to take and what parameters to use. Make sure you account for the position of robot arm and obstacles.
2. Full Plan: The new full plan that you generate based on the last plan. Make sure you properly reflect the above reasoning in the new plan. The plan should be a full plan that includes all the actions from the beginning to the end.
Please organize the output following the json format below:
{
    "Reasoning": "My reasoning for the failure of last plan is ...",
    "Full Plan": ["pick A", "place B", "check", ...]
}
```

Listing 8: Prompt for LLM3 (from scratch) in the Robot Arm Control Tasks

K.5 Reflexion Prompt

```
Your task is to: move the gripper to [1.0, 0.0], while avoid collision with the obstacles at, [2.0, -2.0] radius 0.5
Initial State:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]
> think: To solve the task, I need to first check the action options.
OK.
```

```
> help
1) Move x y: Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics
2) Check: Check Robot arm joint positions
3) Help: View the available action options.
4) think: <your_thought>: Output your thinking and reasoning.
5) End: End the task when you finished, or the task is not solvable.
> think: To solve the task, I need to move gripper to [1.0, 0.0].
OK.
> Move 1.0 0.0
Success!
> Check
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [1.00, 0.00]
> think: Now the gripper is at [1.0, 0.0], the task is complete.
OK.
> End
# Now it's your turn

Your memory for the task below:
Trial 0:
I repeatedly tried to move the gripper to the target location [1.0, 0.0] using intermediate points like [2.0, 0.0], [1.5, 0.0], and [1.1, 0.0]. Each time, the 'Move' command resulted in a location slightly off from the intended target, and ultimately failed to reach [1.0, 0.0] exactly. I also tried to 'End' the task after getting close, but the command failed, probably because I need to reach the target exactly. The inverse kinematics solver seems to be very sensitive.

Plan: Since directly moving to the target and using intermediate points along the x-axis failed, I will try a different strategy: moving along a circular path. I will move the gripper to [3.0, 1.0] first, then [2.0, 1.0], then [1.0, 1.0], and then finally [1.0, 0.0]. This approach might allow the inverse kinematics to find a solution more easily by gradually adjusting the joint angles.
Your task is to: move the gripper to [1.0, 0.0], while avoid collision with the obstacles at, [0.0, 3.0] radius 0.5, [2.0, 2.0] radius 0.5, [-2.0, 2.0] radius 0.5, [2.5, 1.0] radius 0.5, [-2.5, 1.0] radius 0.5, [0.0, -3.0] radius 0.5, [2.0, -2.0] radius 0.5, [-2.0, -2.0] radius 0.5, [2.5, -1.0] radius 0.5, [-2.5, -1.0] radius 0.5
Initial State:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]
> think: To solve the task, I need to first check the action options.
OK.
> help
1) Move x y: Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics
2) Check: Check Robot arm joint positions
3) Help: View the available action options.
4) think: <your_thought>: Output your thinking and reasoning.
5) End: End the task when you finished, or the task is not solvable.
> think: Okay, I remember my previous failed attempts to move directly to [1.0, 0.0] and along the x-axis. My plan now is to try moving the gripper along a different path to see if it helps the inverse kinematics to find a solution while avoiding obstacles. I'll try moving the gripper to [3.0, 1.0] first.
OK.
```

Listing 9: Prompt for Reflexion in the Robot Arm Control Tasks

K.6 CoT-SC Prompt

```
You are an AI robot that generate a plan of actions to reach the goal. You will be given a domain description and the trace of generated plans that lead to motion failure.
You are expected to correct the plan incrementally (on top of the last plan) to avoid the motion failure. This may involve sample new parameters for the failed action, or reverse one or more succeeded actions for backtracking. Make your decision based on the trace provided.

The tabletop environment has a robot arm, several obstacles and a goal location.
The robot arm has two joints and a gripper, the goal is to  move the gripper to [1.0, 0.0], while avoid collision with the obstacles at, [0.0, 3.0] radius 0.5, [2.0, 2.0] radius 0.5, [-2.0, 2.0] radius 0.5, [2.5, 1.0] radius 0.5, [-2.5, 1.0] radius 0.5, [0.0, -3.0] radius 0.5, [2.0, -2.0] radius 0.5, [-2.0, -2.0] radius 0.5, [2.5, -1.0] radius 0.5, [-2.5, -1.0] radius 0.5 .

The robot arm has the following primitive actions:
1) Move x y:  Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics.
```

```
2) Check: Check Robot arm joint positions

The current environment state is:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]

The trace is:
No previous plan

Please generate output step-by-step, which includes:
1. Reasoning: Your reasoning for the failure of last plan if the last plan exists, and the strategy to accomplish the task goal. Make sure you account for the position of robot arm and obstacles. Try to answer the questions: (i) what is the cause of the failure of last plan? (ii) can altering action parameters for the failed action solve the problem? if yes, what feasible action parameters should we use? (iii) do we need to reverse one or more succeeded actions executed before the failed action? if yes, which actions should be reversed? (iv) if the task goal is not achieved, how can we revise the plan to achieve the goal?
2. Full Plan: The new full plan that you generate based on the last plan. Make sure you properly reflect the above reasoning in the new plan. The plan should be a full plan that includes all the actions from the beginning to the end.
Please organize the output following the json format below:
{
    "Reasoning": "My reasoning for the failure of last plan is ...",
    "Full Plan": ["pick A", "place B", "check", ...]
}
```

Listing 10: Prompt for CoT-SC in the Robot Arm Control Tasks

K.7 Tree-of-Thoughts Prompt

```
You are an AI robot that generate a plan of actions to reach the goal. You will be given a domain description and the trace of generated plans that lead to motion failure.
You are expected to correct the plan incrementally (on top of the last plan) to avoid the motion failure. This may involve sample new parameters for the failed action, or reverse one or more succeeded actions for backtracking. Make your decision based on the trace provided.

The tabletop environment has a robot arm, several obstacles and a goal location.
The robot arm has two joints and a gripper, the goal is to  move the gripper to [1.0, 0.0], while avoid collision with the obstacles at, [0.0, 3.0] radius 0.5, [2.0, 2.0] radius 0.5, [-2.0, 2.0] radius 0.5, [2.5, 1.0] radius 0.5, [-2.5, 1.0] radius 0.5, [0.0, -3.0] radius 0.5, [2.0, -2.0] radius 0.5, [-2.0, -2.0] radius 0.5, [2.5, -1.0] radius 0.5, [-2.5, -1.0] radius 0.5 .

The robot arm has the following primitive actions:
1) Move x y:  Move the gripper to (x, y). The rotation of the joints will be calculated by inverse kinematics.
2) Check: Check Robot arm joint positions

The current environment state is:
Joint positions: 'Joint 0': [0.00, 0.00] 'Joint 1': [2.00, 0.00] 'Gripper': [3.00, 0.00]

The trace is:
No previous plan

Current New Plan: ['Move 2.0 0.0', 'Move 1.0 0.0', 'Check', 'Check']

Please generate output step-by-step, which includes:
1. Reasoning: Your reasoning for the failure of last plan if the last plan exists, and the strategy to accomplish the task goal based on current new plan. Make sure you account for the position of robot arm and obstacles. Try to answer the questions: (i) what is the cause of the failure of last plan? (ii) can altering action parameters for the failed action solve the problem? if yes, what feasible action parameters should we use? (iii) do we need to reverse one or more succeeded actions executed before the failed action? if yes, which actions should be reversed? (iv) if the task goal is not achieved, how can we revise the plan to achieve the goal?
2. Next Action: Based on your reasoning, what is the next best action for current new plan? The action you choose will be append to the current new plan. Make sure you account for the actions in the current new plan. Output a single action only.
Please organize the output following the json format below:
{
    "Reasoning": "My reasoning for the failure of last plan is ...",
    "Next Action": "pick A"
}
```

```
Your output in json is (please don't output ```json):
```

Listing 11: Prompt for Tree-of-Thoughts in the Robot Arm Control Tasks

K.8 In-Context Learning Prompt

```
# Example Interaction History 1
Task: "Create 2 ml of red paint in container B".

## Step 1: Check red tube color.
- Act: Clean A
- Act: Add red to A
- Obs: You add 1 ml of paste from red tube into container A.
- Act: Check A
- Obs: Container A has 1 ml of blue paint.

## Step 2: Find red color.
- Act: Clean A
- Act: Add blue to A
- Obs: You add 1 ml of paste from blue tube into container A.
- Act: Check A
- Obs: Container A has 1 ml of red paint.

## Step 3: Complete the task using blue label tube for red paint.
- Act: Clean B
- Act: Add blue to B
- Obs: You add 1 ml of paste from blue tube into container B.
- Act: Add blue to B
- Obs: You add 1 ml of paste from blue tube into container B.

# Example Interaction History 2
Task: "Create 2 ml of blue paint in container B".

## Step 1: Check container B.
- Act: Check B
- Obs: Container B has 1 ml of blue paint.

## Step 2: Clean container B.
- Act: Clean B

## Step 3: Complete the task by adding 2 ml of red paint in container B.
- Act: Clean B
- Act: Add blue to B
- Obs: You add 1 ml of paste from blue tube into container B.
- Act: Add blue to B
- Obs: You add 1 ml of paste from blue tube into container B.

Please generate output step-by-step, which includes:
1. Reasoning: Your reasoning for the actions to take. You are expected to follow the strategy in the example interaction history.
2. Full Plan: The plan should be a full plan that includes all the actions you want to take.
```

Listing 12: Prompt for In-Context LLM planner in the Mix Color Tasks