SGG-REFLACT: SUB-GOAL GUIDED REFLACT WITH STRUCTURED PLANNING FOR RELIABLE LONG-HORIZON REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in reasoning backbones have empowered large language model (LLM) agents to tackle complex, multi-step tasks. However, as reasoning horizons grow, inconsistent internal beliefs induce intermediate errors that cause agents to drift from their goals. This limitation also persists in REFLACT, which reflects only on the end-goal at each step without explicitly considering intermediate sub-goals. To address this problem, we propose SGG-ReflAct (Sub-Goal Guided ReflAct), a reasoning backbone that integrates sub-goals generated through a single-path LLM planner into the reflection process. We further extend this framework to BeamSGG-ReflAct, which replaces the single-path planner with a beam search-based LLM planner for structured plan exploration. We run experiments on ALFWorld, ScienceWorld, and Jericho with multiple LLM models. SGG-ReflAct outperforms REFLACT in nearly all settings, achieving best success rate gains of 14.9 percentage points on ALFWorld and 8.0 percentage points on ScienceWorld with Llama-3.1-8B-Instruct. Our experimental analysis shows that SGG-ReflAct reduces hallucinated actions and achieves its largest gains on procedurally ordered tasks. Furthermore, experimental results with BeamSGG-ReflAct show that the backbone's effectiveness depends on plan quality: explicitly specifying the required operations recovers gains that plan searching alone cannot achieve. These results demonstrate that SGG-ReflAct offers a practical and highly effective reasoning backbone, enabling LLM agents to achieve reliable performance in complex, long-horizon tasks through easy integration.

1 INTRODUCTION

Recent advances in Large Language Models (LLMs) have expanded their capabilities, enabling LLM-based agents to tackle complex tasks that require more than a single response (Brown et al., 2020; Wang et al., 2024). Such tasks demand multi-step decision making under partial observability and sparse rewards, so the agent has to reason, act, and observe over a long horizon (Yao et al., 2023b; Shridhar et al., 2021; Wang et al., 2022). A *reasoning backbone* governs that loop by fixing how the agent thinks before it acts, and its design has therefore become a central focus of LLM agent research.

Existing research follows three primary directions. The first, such as REACT, interleaves reasoning with action, anchoring each thought to the most recent observation (Wei et al., 2022; Yao et al., 2023b). The second, such as Plan-and-Act, provides explicit structure through planning and searching over future trajectories (Wang et al., 2023; Erdogan et al., 2025; Xiong et al., 2025; Zhou et al., 2024). The third employs reflection on top of a reasoning backbone, revisiting failed episodes to inform subsequent attempts (Shinn et al., 2023).

Despite these diverse paradigms, all three share a fundamental vulnerability in long-horizon interactions: ungrounded steps enter unchecked and compound (Kim et al., 2025; Xie et al., 2023), one misaligned thought triggering a hallucinated action that yields a misleading observation and corrupts the next thought, so the agent's belief drifts from the environment until reasoning collapses. REFLACT (Kim et al., 2025) attacks this at its source, redesigning the reasoning–action cycle itself: at each timestep the agent weighs its current state against the task goal before action selection,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

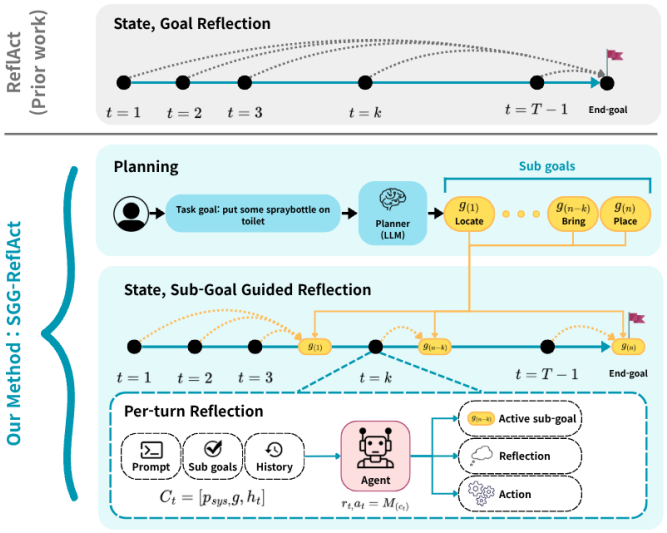

Figure 1: Overview of SGG-ReflAct on an ALFWorld task. (Top) REFLACT anchors per-turn reflections to a fixed end-goal. (Bottom) SGG-ReflAct generates a sub-goal plan $\mathcal{G} = (g_1, \ldots, g_N)$ via a single planning call, anchoring reflections to the active sub-goal to allow the target to move with the agent. The inset illustrates the inference at turn $t = k$, where context c_t (system prompt, sub-goal plan, and history) is used to predict the active sub-goal, reflection r_t, and next action a_t in a single step. SGG-ReflAct incurs minimal overhead, adding only one initial planning call.

which grounds reasoning in the environment state. This works well, but weakens in long-horizon tasks, when early in an episode the state is still far from the end-goal and reflecting on it offers little concrete guidance toward the next step. We therefore conjecture that anchoring each reflection to a nearer, intermediate target rather than the distant end-goal gives more actionable guidance while still far from completion. Our proposed anchoring directly addresses this issue, preventing early misjudgments from compounding into the ungrounded actions examined in Section 5.1.

Motivated by this, we propose **SGG-ReflAct** (**S**ub-**G**oal **G**uided **ReflAct**), a reasoning backbone that bridges structured planning and reflective acting (Figure 1). The mechanism is twofold. First, a *Planner* module runs once at episode start and decomposes the task goal into a sub-goal plan, each sub-goal serving as an explicit milestone. Second, at each timestep the agent weighs its present state against *the active sub-goal*, rather than the distant end-goal, before action selection; the reflection target thus advances with the agent and stays proximate as the horizon grows. Due to the modular nature of the planner, we can isolate the impact of search effort by keeping the reflection-and-acting phase constant. This allows us to evaluate whether increased computation on plan search yields better performance than direct planning. Specifically, we introduce **BeamSGG-ReflAct**, which replaces the single-path planner with a self-evaluation guided beam search over candidate sub-goal sequences (Xie et al., 2023).

We evaluate both methods on three models (GPT-4o, GPT-4o-mini, and Llama-3.1-8B-Instruct) and three interactive text benchmarks: ALFWorld (Shridhar et al., 2021), ScienceWorld (Wang et al., 2022), and Jericho (Hausknecht et al., 2020). SGG-ReflAct outperforms the REACT, Plan-and-Act, and REFLACT baselines in nearly all settings. The gains are largest on the resource-constrained Llama-3.1-8B-Instruct, where SGG-ReflAct reaches 50.7% on ALFWorld and 33.6% on ScienceWorld. These results surpass REFLACT by +14.9 pp and +8.0 pp. Our work makes the following contributions:

1. We propose SGG-ReflAct, a reasoning backbone that bridges sub-goal planning and per-turn reflection. Each reflection is anchored to the active sub-goal rather than the distant end-goal, keeping the reflection target proximate as the horizon grows; its planner is a *modular* component, so a different planner can be swapped in without altering the reflection-and-acting backbone, and we instantiate this with BeamSGG-ReflAct.

2. Through a *grounding* analysis we show that the gains stem not from merely adding plans to the context—as previously noted by Kim et al. (2025)—but from anchoring each reflection to the active sub-goal. This mitigates the accumulation of hallucinated actions during long-horizon episodes, reducing the invalid-action rate from 43.3% to 29.9% on Llama-3.1-8B-Instruct.
3. Through a *plan-content* analysis we show that plan quality determines the backbone's effectiveness; inadequate plans can turn gains into degradation, and successful plans must specify the operations required to achieve the goal rather than merely restating it. Explicitly prompting the planner to include these operations recovers gains that plan search alone cannot achieve.

2 Related Work

LLM agent reasoning backbones. Structured reasoning has been at the center of progress in LLM agents: Chain-of-Thought (CoT) prompting (Wei et al., 2022) first showed that reasoning step by step improves answers on hard tasks, and set the stage for agent frameworks. ReAct (Yao et al., 2023b) took the next step, interleaving reasoning and action so each thought is tied to the latest observation. ReAct, however, forms each thought from the current observation and the past history alone, without tying it to the end-goal or to how much of the task is done; this gap becomes costly as the chain grows, because early mistakes carry forward and build into a full breakdown (Xie et al., 2023; Kim et al., 2025).

Several methods add modules on top of ReAct: WKM (Qiao et al., 2024) and KnowAgent (Zhu et al., 2025) keep a world model and an action knowledge base to rule out infeasible moves, while MPO (Xiong et al., 2025) and Dynamic Cheatsheet (Suzgun et al., 2026) hand the agent an optimized meta-plan or an adaptive memory, and Trial and Error (Song et al., 2024) searches the action space directly. All add knowledge, memory, or a plan, but leave the core thought-action loop unchanged, so they help little when the backbone itself produces ungrounded thoughts.

Agent planning and search. A second line builds a plan before or during execution. Plan-and-Solve (Wang et al., 2023) and Plan-and-Act (Erdogan et al., 2025) turn the task into a high-level plan at the start and run it step by step, giving a structure that plain ReAct lacks; Pre-Act (Rawat et al., 2025) instead keeps updating the plan as each step returns new output, and SwiftSage (Lin et al., 2023) uses a hybrid approach. Like Plan-and-Solve and Plan-and-Act, we fix the plan for the whole episode; what differs is that we use it as a target for reflection rather than a script to execute, so the extra cost stays at one planning call. Others organize the plan itself—ReAcTree (Choi et al., 2026) as a control-flow tree, PlanU (Deng et al., 2025) under uncertainty over sub-goal orderings, and plans refined from retrieved experience and graph-based checks (Kagaya et al., 2024; Jeong et al., 2026)—while RAP (Hao et al., 2023) plans with a world model and RAFA (Liu et al., 2024) and LATS (Zhou et al., 2024) reason over trajectories or rollouts. Most of these plan-centric methods stay with their plan and act without per-turn state–goal reflection, so they turn fragile when an observation does not fit the plan.

Search-based methods explore several reasoning paths at inference time: Tree-of-Thoughts (ToT) (Yao et al., 2023a) and Graph-of-Thoughts (GoT) (Besta et al., 2024) grow CoT reasoning into a branching search, and Self-Evaluation Guided Beam Search (Xie et al., 2023) runs beam search over candidates scored by the LLM itself, and rStar (Qi et al., 2025) lets two LLMs reason against each other to steer the search. These work well for static or single-step reasoning, but do not carry over directly to sequential, partially observable environments. BeamSGG-ReflAct adapts self-evaluation guided beam search to plan building instead, scoring sub-goal sequences by feasibility and order rather than answer correctness, so the planner stays a drop-in part of SGG-ReflAct.

Reflection mechanisms. A growing line of work has agents reflect on and fix their own behavior. Reflexion (Shinn et al., 2023) writes a verbal self-reflection after a failed episode, Self-Refine (Madaan et al., 2023) and CRITIC (Gou et al., 2024) revise an output using self- or tool-generated feedback, and ExpeL (Zhao et al., 2024) distills reusable lessons from past runs; all reflect on *outputs* or *past episodes*, outside the per-turn loop where the errors first appear. Closer to that loop, ReBACT (Zeng et al., 2025) fixes errors before each action, and InferAct (Fang et al., 2025) checks

a proposed action before it runs; both act as a *verifier* placed after the action. REFLACT (Kim et al., 2025) instead redesigns the backbone itself, so that at each step the agent weighs its current state against the task goal rather than just predicting the next action. Grounding each turn cuts belief mismatch sharply, yet REFLACT's reflection still gives little help early in a long task, when the current state is far from the end-goal. SGG-ReflAct builds on this line, keeping per-turn reflection but anchoring it to the active sub-goal instead of the distant end-goal, so it differs from plan-injection methods such as MPO (Xiong et al., 2025), which add a plan to the context and leave the reflection backbone unchanged (Table 2, Appendix A). This is consistent with Kim et al. (2025), who found that adding MPO meta-plans to a reflection backbone yields negligible performance gains, as a plan is effective only when it redirects the per-turn reflection rather than merely filling the context.

3 METHOD

We formalize the task as a POMDP (Kaelbling et al., 1998), following the notation of Kim et al. (2025). Let u denote the task instruction, including the end-goal. At each step t, the agent receives an observation o_t, maintains a history $h_t = \{(o_1, a_1), \ldots, (o_{t-1}, a_{t-1}), o_t\}$, and selects an action a_t via a language model $\mathcal{M}$. Given the partial observability of the environment, the agent must act on a belief formed from h_t; errors in that belief lead to hallucinated actions. A reasoning backbone, following Yao et al. (2023b), adds a natural-language thought τ_t before a_t to reduce them.

All three backbones receive the task instruction u, but differ in what the per-turn instruction directs τ_t at. REACT attends to the history alone, $\tau_t,\ a_t = \mathcal{M}(h_t)$, with no link to the goal or to progress so far. REFLACT (Kim et al., 2025) attends to the state relative to the goal, producing a reflection r_t^g toward the end-goal, $\tau_t,\ r_t^g,\ a_t = \mathcal{M}(h_t)$, but in long-horizon tasks the distance between the current state and end-goal in u leaves the reflection loosely tied to the immediate sub-task. To address this, SGG-ReflAct attends to the active sub-goal of a plan $\mathcal{G} = (g_1, \ldots, g_N)$, producing a reflection r_t^{sg} toward that sub-goal, $g_t^*,\ \tau_t,\ r_t^{sg},\ a_t = \mathcal{M}(h_t,\ \mathcal{G})$: at each turn t the agent identifies the *active sub-goal* $g_t^* \in \mathcal{G}$ and reflects on it rather than the end-goal in u. This framework runs in two phases: a *planning phase* that decomposes the end-goal in u into $\mathcal{G}$ once per episode, and a *reflection-and-acting phase* that runs every turn, doing sub-goal guided reflection before acting. Figure 1 illustrates both phases and contrasts our reflection target with that of REFLACT.

3.1 SGG-REFLACT FRAMEWORK

Phase 1: Goal Decomposition. The first phase is a dedicated *Planner* module. It runs once at $t = 0$, before the agent touches the environment, and maps an initial context c_0 and a single in-context planning example $\mathcal{E}$ to an ordered sequence of N high-level sub-goals:

$$\mathcal{G} = \textsc{Planner}(c_0,\ \mathcal{E}) = (g_1,\ g_2,\ \ldots,\ g_N)\,. \tag{1}$$

We refer to $\mathcal{G}$ simply as *the plan*. The initial condition c_0 varies by environment: $c_0 = u$ for ScienceWorld, and $c_0 = \{u, o_0\}$ for ALFWorld and Jericho to provide spatial context. Each sub-goal g_i defines a high-level objective (*what to do*) rather than a specific action (*how to do*), using generalized phrasing to preserve plan generality. These sub-goals are ordered by environmental prerequisites. We set $N = 4$ based on ScienceWorld's median sub-goal sequence length; to maintain one unified architecture rather than tune N per environment, we apply the same value to ALFWorld and Jericho, where tasks also exhibit a comparable four-stage structure (Appendices B, P, Q, R). Producing this fixed sequence $\mathcal{G}$ costs one additional LLM call per episode; because the planner commits to a single candidate without search or revision, we call it a *single-path* planner. These sub-goals remain abstract and *phase-level* across all benchmarks to emphasize agent interaction.

Phase 2: Sub-Goal Guided Reflection and Acting. At every turn $t \geq 1$, the agent sees the plan $\mathcal{G}$ and the history $h_t = \{(o_1, a_1), \ldots, (o_{t-1}, a_{t-1}), o_t\}$, with the plan fixed at the front of the context:

$$c_t = \Big[\ \underbrace{p_{\text{sys}}}_{\text{system}}\ ,\quad \underbrace{\mathcal{G}}_{\text{sub-goals}}\quad ,\quad \underbrace{h_t}_{\text{history}}\ \Big]. \tag{2}$$

The system prompt p_{sys} carries the task instruction u, so the end-goal stays in the context throughout, and it asks the agent to reply in three parts: the active sub-goal $g_t^* \in \mathcal{G}$ it is working on, a one-sentence reflection on where it stands on g_t^*, and the next action a_t. Appendix N shows the full instruction and format.

REFLACT tells the agent to reflect on its state against the end-goal; ours instead has the agent first name the active sub-goal, then reflect on its state against it. Since the agent can reach g_t^* from its current state, while u may remain many actions away, the reflection stays grounded in what the agent can act on next, even early in a long episode. The agent then produces:

$$g_t^*,\ \tau_t,\ r_t^{sg},\ a_t = \mathcal{M}(c_t)\,, \tag{3}$$

where $\mathcal{M}$ is the language model that runs the policy, c_t is the context from Equation 2, $g_t^* \in \mathcal{G}$ is the active sub-goal the model selects for this turn, τ_t is its thought, r_t^{sg} is its reflection on g_t^*, and a_t is the action. All four come from a single call, so identifying the active sub-goal costs no extra LLM call. Rather than four discrete fields, the model produces a single free-form response where g_t^* is described within the reflection; only a_t is parsed via the `Action:` tag.

3.2 BEAMSGG-REFLACT: SEARCH-ENHANCED PLANNING

Since reflections are anchored to the active sub-goal from $\mathcal{G}$, the agent's reasoning capability is inherently constrained by the plan's coverage. Because the plan is static, any missing task requirements cannot be addressed during the episode. To test whether searching the space of sub-goal sequences improves plan quality, BeamSGG-ReflAct replaces the single-path planner with a *Beam Search Planner*, which likewise runs once at $t = 0$, before the episode begins. This planner adapts self-evaluation guided beam search (Xie et al., 2023) to construct $\mathcal{G}$ incrementally. Unlike MCTS-based approaches such as LATS (Zhou et al., 2024), which require environment rollouts or world models, our planner relies exclusively on LLM self-evaluation, requiring no additional environment interaction.

Beam Search Planner. The planner expands a beam of K partial plans, adding one sub-goal at each depth $d \in \{1, \ldots, N\}$. At each step, it generates S candidates per beam, drawing them one at a time rather than all at once because decoding is deterministic (Appendix E): each new request is told which candidates have already been produced at that depth and is asked for a different one (see Appendix C). Writing $\rho = (g_1, \ldots, g_d)$ for a candidate's partial plan at depth d, candidates are scored by the product of their prefix self-evaluation scores:

$$\text{score}(\rho) = \prod_{i=1}^{|\rho|} \textsc{SelfEval}(\rho_{1:i} \mid u)\,, \tag{4}$$

then keeps the top K for the next depth. Here $\textsc{SelfEval}(\cdot)$ is a separate LLM call that rates a prefix on a $[0, 1]$ scale against a fixed rubric (Equation 5). It judges the whole prefix rather than the sub-goal just added, so an early sub-goal enters the product at every later depth and carries more weight than a late one. For our experiments, we set $N = 4, K = 2, S = 3$. With two LLM calls per candidate (expansion and evaluation), the search requires at most 42 calls per episode. Full details are in Appendix C.

BeamSGG-ReflAct shares SGG-ReflAct's reflection-and-acting phase and differs only in how $\mathcal{G}$ is built, so the gap between them measures the effect of the sub-goal plan itself rather than making BeamSGG-ReflAct a separate method; both planner prompts impose the same constraints and differ only in the unit they produce.

Setting $K = 1$ and $S = 6$ reduces the planner to a *greedy* search that retains one partial plan and samples six candidates per depth (up to 48 calls per episode); Section 5.2 evaluates both variants.

4 EXPERIMENTS

We ask whether sub-goal guided, per-turn reflection improves performance and which of its two ingredients—the sub-goal plan and the per-turn reflection—is responsible; the baselines therefore remove one at a time. The benchmarks are long-horizon, where a reflection target far from the agent's state has room to drift, and the models span three capacity levels, since explicit sub-goals should matter most where the model has least to spare.

Table 1: Performance comparison across benchmarks. Values are percentages. ± denotes standard deviation across each benchmark's own task groupings: ALFWorld's six task types (Appendix F) and the ScienceWorld split's nine categories (Appendix G), whose tables report the same value in their *SD* column. Jericho defines no such grouping, so none is reported for it. **Bold** indicates the best performance within each model group per column.

Model	Agent	ALFWorld	ScienceWorld		Jericho	
		SR	SR	AR	SR	AR
GPT-4o	NoThinking	78.4 ± 16.8	53.6 ± 29.4	66.6 ± 21.5	20.0	45.8
	REACT (Yao et al., 2023b)	80.6 ± 11.8	59.2 ± 35.6	71.7 ± 22.1	25.0	44.2
	Plan-and-Act	81.3 ± 9.9	54.0 ± 33.4	68.2 ± 24.3	30.0	49.2
	REFLACT (Kim et al., 2025)	89.6 ± 8.9	55.5 ± 32.7	67.2 ± 20.8	35.0	54.4
	BeamSGG-ReflAct (ours)	84.3 ± 14.7	63.5 ± 31.5	**74.9** ± 20.8	25.0	43.1
	SGG-ReflAct (ours)	**90.3** ± 8.9	**64.0** ± 37.4	73.5 ± 18.0	**40.0**	**55.0**
GPT-4o-mini	NoThinking	44.0 ± 24.0	20.9 ± 23.0	40.2 ± 24.0	10.0	36.0
	REACT (Yao et al., 2023b)	45.5 ± 17.4	40.8 ± 33.0	53.2 ± 20.7	5.0	26.1
	Plan-and-Act	43.3 ± 9.8	36.0 ± 33.4	51.0 ± 28.0	5.0	21.1
	REFLACT (Kim et al., 2025)	62.7 ± 21.3	40.3 ± 34.5	55.6 ± 23.9	15.0	34.3
	BeamSGG-ReflAct (ours)	70.1 ± 26.0	43.6 ± 31.9	55.9 ± 20.5	15.0	36.3
	SGG-ReflAct (ours)	**72.4** ± 21.3	**45.0** ± 33.8	**60.5** ± 22.8	**20.0**	**38.6**
Llama-3.1 -8B-Instruct	NoThinking	28.4 ± 25.2	13.3 ± 16.7	34.6 ± 19.5	5.0	26.3
	REACT (Yao et al., 2023b)	29.9 ± 3.8	18.5 ± 16.4	36.9 ± 19.2	0.0	15.7
	Plan-and-Act	29.9 ± 13.5	11.8 ± 11.7	33.6 ± 26.2	0.0	16.2
	REFLACT (Kim et al., 2025)	35.8 ± 27.5	25.6 ± 27.5	41.5 ± 23.5	5.0	**26.6**
	BeamSGG-ReflAct (ours)	46.3 ± 22.5	26.5 ± 24.0	41.9 ± 23.3	5.0	17.8
	SGG-ReflAct (ours)	**50.7** ± 19.6	**33.6** ± 26.2	**48.7** ± 23.4	**10.0**	24.3

4.1 BASELINES

Following Kim et al. (2025), we compare against four frameworks chosen to span the two ingredients of SGG-ReflAct. **NoThinking** and **REACT** (Yao et al., 2023b) hold neither a sub-goal plan nor a state–goal reflection and set the floor. **REFLACT** (Kim et al., 2025) supplies per-turn reflection without sub-goals, so the reflection target alone separates it from ours. **Plan-and-Act** (Erdogan et al., 2025) supplies a first-timestep plan without per-turn reflection, its plan coming from the agent's own first-turn thought rather than a dedicated planner, so what separates it is what the agent does with the plan once the episode is under way. A gain that survives both single-ingredient baselines cannot be credited to either on its own. Appendix A places all of them on the same two axes.

4.2 BENCHMARKS AND METRICS

We evaluate on three interactive text benchmarks that differ in domain and in what they demand of a sub-goal plan. **ALFWorld** (Shridhar et al., 2021) is a TextWorld (Côté et al., 2019) household environment whose standard 134-episode split has six task types. **ScienceWorld** (Wang et al., 2022) covers procedural science experiments in a 211-episode split of 24 task types (grouped into nine by the split's identifiers); many enforce a strict prerequisite order, so a sub-goal plan that names the required operation pays off. **Jericho** (Hausknecht et al., 2020) is the hardest of the three: the 20-game AgentBoard (Ma et al., 2024) set pairs large action spaces with sparse rewards, so a plausible-sounding sub-goal is often unreachable. We report **Success Rate (SR)** across all benchmarks and **Average Reward (AR)** for ScienceWorld and Jericho. Because the built-in success flags in ScienceWorld and Jericho are unreliable, we follow Kim et al. (2025) and derive success from progress rewards; Appendices E and D give the thresholds, the settings behind them, and the per-task step budgets.

4.3 MODELS AND IMPLEMENTATION

We evaluate on three models: **GPT-4o** and **GPT-4o-mini** (OpenAI, 2024) as proprietary models, and **Llama-3.1-8B-Instruct** (Dubey et al., 2024) as a resource-constrained, open-weight model. Llama-3.1-8B-Instruct is the most likely to benefit from explicit sub-goals, so it is the primary subject of the analyses in Sections 5.1 and 5.2. Appendix E gives the decoding, prompting, and serving settings, which are identical across agents.

4.4 MAIN RESULTS

Table 1 reports how SGG-ReflAct and BeamSGG-ReflAct compare with the NoThinking, REACT, Plan-and-Act, and REFLACT baselines across the three models and three benchmarks.

ALFWorld. SGG-ReflAct leads in every model group, reaching 90.3% with GPT-4o, 72.4% with GPT-4o-mini, and 50.7% with Llama-3.1-8B-Instruct. On Llama this last number is **+14.9 pp** above REFLACT. The single-path planner stays ahead of the Beam Search Planner in every model group, by 2.3 to 6.0 points, and both are well ahead of Plan-and-Act; planning and reflection thus help in different ways (Section 5.2).

ScienceWorld. SGG-ReflAct again leads every model group in SR, and in AR on all but GPT-4o, where the Beam Search Planner edges ahead (74.9 against 73.5). On Llama-3.1-8B-Instruct it improves over REFLACT by **+8.0 pp** SR and **+7.2 pp** AR. A fixed plan without per-turn reflection can also hurt weaker models: Plan-and-Act drops to 11.8% on Llama, below even NoThinking (13.3%), which agrees with Kim et al. (2025)'s finding that smaller models struggle to follow a preset plan.

Jericho. On the 20-game Jericho suite, SGG-ReflAct has the best SR in every model group (for example, 40.0% against REFLACT's 35.0% on GPT-4o) and the best AR on GPT-4o and GPT-4o-mini. The Beam Search Planner, by contrast, never beats the single-path planner on any model; on GPT-4o it falls to 25.0%, behind even REFLACT and Plan-and-Act. We revisit this beam disadvantage across all three benchmarks in Section 5.2.

5 DISCUSSION

The efficacy of SGG-ReflAct is driven by two factors: **(C1)** the necessity of reflection-target proximity for robust grounding (Section 5.1) and **(C2)** the critical role of plan content in determining backbone performance (Section 5.2).

5.1 REFLACT VS. SGG-REFLACT: TASK-LEVEL SUCCESS AND GROUNDING (C1)

Impact of sub-goal plans vs. anchoring instructions. SGG-ReflAct introduces two key modifications to REFLACT: the inclusion of a sub-goal plan in the context and the instruction to anchor each reflection to the active sub-goal. To disentangle these effects, we conduct a controlled ablation on ScienceWorld with Llama-3.1-8B-Instruct. REFLACT+Plan uses the same single-path planner as SGG-ReflAct to generate its sub-goal plan, but retains REFLACT's original instruction to reflect against the end-goal, so the only deliberate difference between the two agents is what the per-turn reflection targets (see Appendix H for verbatim instructions). Merely appending the plan to the context is not enough: without an instruction to identify the currently active sub-goal, the agent has no mechanism for using it.

As shown in Table 13 (Appendix S), of the +8.0 pp improvement of SGG-ReflAct over REFLACT, the mere presence of the plan in the context accounts for only +1.9 pp; the remaining +6.1 pp (to 33.6%) is attributed to anchoring the reflection to the active sub-goal. Average reward follows a comparable trend: the plan alone raises it by +3.3 pp, and anchoring adds a further +3.9 pp to reach 48.7%, aligning with Kim et al. (2025), who found that adding a meta-plan to a reflection backbone yields negligible improvements. Appendix H traces both agents turn by turn on a task that REFLACT+Plan fails and SGG-ReflAct solves.

What changes inside an episode: fewer hallucinated actions. To analyze the temporal effects of the anchoring instruction, we use the **invalid-action rate**, the fraction of executed actions the environment refuses to interpret or apply, as a proxy for *hallucinated actions*. We pool observations from all episodes regardless of success, and Appendix I shows that the pattern reflects grounding rather than formatting. We report this rate overall and across three temporal thirds of each episode. Table 12 gives the values for all six agents on ScienceWorld.

First, grounding errors in reflection-only backbones (REACT and REFLACT) compound over time. For REFLACT on Llama-3.1-8B-Instruct, the invalid-action rate rises sharply from 14.5% to 44.1%

to 71.4% (Figure 2, Appendix S), so ungrounded actions become more frequent, not just more numerous.

Second, SGG-ReflAct substantially mitigates this error accumulation. On Llama-3.1-8B-Instruct, SGG-ReflAct reduces the overall invalid-action rate from 43.3% (REFLACT) to 29.9% and flattens the late-stage surge (from 71.4% to 49.7%); BeamSGG-ReflAct also stays under REFLACT (34.2% overall), though above SGG-ReflAct. The improvement over REFLACT holds on GPT-4o as well, both overall (27.7% versus 34.2%) and early (11.2% versus 19.0%). On GPT-4o-mini the reduction is confined to the early third (17.7% versus 23.3%): the overall rate is unchanged (32.6% versus 32.7%) and the late third is worse (45.5% versus 39.0%). Appendix J traces one such cascade step by step.

A low invalid-action rate is necessary but not sufficient. While a low invalid-action rate might suggest an agent is merely attempting less demanding actions, Table 12 shows that this metric is only meaningful when paired with high success rates. On GPT-4o and GPT-4o-mini, agents without per-turn reflection (e.g., NoThinking and Plan-and-Act) exhibit the lowest invalid-action rates (20.9% and 23.3% on GPT-4o) but also the lowest success rates (53.6% and 54.0%, against 64.0% for SGG-ReflAct). Those low rates are not a matter of acting less: on GPT-4o, NoThinking averages 24.7 actions per episode and Plan-and-Act 28.3, against SGG-ReflAct's 20.8 (Appendix K). The difference lies in action quality, not quantity. In contrast, SGG-ReflAct and BeamSGG-ReflAct reduce invalid actions and increase success rates at the same time, offering the more informative within-family comparison. This is most evident on the constrained Llama-3.1-8B-Instruct, where our variants achieve both the lowest invalid-action rates and the highest success rates.

Where the gains show up: procedurally ordered tasks. The benefits of improved grounding are non-uniform across tasks. Figure 3 (Appendix S) decomposes ScienceWorld into ten task buckets on Llama-3.1-8B-Instruct (Table 7 gives the per-row values), revealing that gains are most pronounced in tasks requiring strict prerequisite ordering. For instance, REFLACT achieves only 0% on Boil and 5.6% on Paint/Mix—the latter of which underperforms even the baseline REACT (22.2%). In contrast, by anchoring reflections to task-relevant sub-goals, SGG-ReflAct achieves 55.6% and 61.1% on these same tasks, whereas end-goal reflection can lead to misguided trajectories. The performance margin narrows for Find/classification (77.5% for SGG-ReflAct vs. 70.0% for REFLACT), where sub-goal structuring offers minimal advantage for one-step decisions. However, REFLACT outperforms SGG-ReflAct on knowledge-retrieval Lifespan tasks (66.7% vs. 53.3%), suggesting that SGG-ReflAct tends to over-decompose tasks: these tasks require only two ordered actions (Appendix B), yet the SGG-ReflAct planner still emits four. For long-horizon exploration (Grow), no agent exceeds 5.0%, indicating that the step budget, rather than the planning mechanism, is the primary bottleneck.

5.2 PLAN CONTENT AS A FUNDAMENTAL DETERMINANT OF BACKBONE PERFORMANCE (C2)

The Primacy of Plan Content. Since the two share a backbone (§3.2), Table 1 isolates the planner's impact. Ranked by success rate, SGG-ReflAct is ahead of BeamSGG-ReflAct in all nine (model, benchmark) cells; against REFLACT, SGG-ReflAct is ahead in all nine, whereas BeamSGG-ReflAct is ahead in five, which suggests that the planner limits the backbone's potential: if candidate plans miss necessary steps, increasing the search budget will not help. We turn to plan content for the rest of this section (Table 14).

Structural Deficiencies in Planning. Qualitative analysis attributes these failures to a structural deficiency: deficient plans often merely restate the goal rather than prescribing physical transformations. A "Melt" plan may include a sub-goal such as "change its state to liquid," restating the task description instead of naming the required operation (applying a heat source). Corroborated by our observations on Llama-3.1-8B-Instruct, where the agent earns partial reward on 14 of the 18 Freeze and Melt episodes yet solves none, the failure lies in the plan's content rather than execution errors. Because these critical steps are entirely absent, search-based reordering cannot resolve the failure.

Procedural Guidance as a Remedy. To address this, we investigated whether explicit procedural prompting could recover these missing steps. This procedural prompting is applied *only to the Plan-*

ner, never to the agent prompt, so the comparison stays fair. As shown in Table 14 (Appendix S), introducing it for state-change tasks (Appendix Q) yielded clear improvements, increasing success on the targeted state-change episodes by up to 16.7 points on GPT-4o. Because the agent never sees the procedure, these improvements are driven by the Planner through the sub-goals it generates. Llama-3.1-8B-Instruct's single-path planner is the exception, losing 1.4 points overall and 5.6 on state-change, because the small LLM cannot turn the added rule into a correctly ordered sub-goal plan; the Beam Search Planner instead achieves this on the targeted episodes, matching or surpassing single-path performance across all tested models. The bottleneck therefore lies in the absence of the required operations, not the search mechanism: explicit prompting supplies the operation, though the search mechanism remains necessary for the weakest model to use it.

Limitations of Search-Based Optimization. Without the explicit rule, BeamSGG-ReflAct does not outperform the single-path baseline, which we attribute not to the search mechanism itself but to how candidates are scored. Our analysis points to two issues. First, the self-evaluation judge (Equation 5) lacks environmental grounding, relying exclusively on the LLM's internal knowledge. Second, the scorer is backward-looking: its rubric (Appendix C) prioritizes the local consistency of existing sub-goals over future task completion, which is assessed only at the final depth. Locally well-formed prefixes can therefore score highly even without essential future operations, so high-scoring but flawed prefixes are preserved and increasing the beam width merely reorders existing candidates rather than introducing the missing operations required for success. This bottleneck is most evident in Jericho, where the gap between plausible-sounding sub-goals and executable actions is widest, not from the beam width itself: a budget-comparable greedy search (up to 48 calls, against beam's 42) lands at 27.0% vs. BeamSGG-ReflAct's 26.5% on Llama-3.1-8B-Instruct (Table 9), though we tested no other configuration on the larger models. While search may occasionally recover isolated operations (e.g., `melt caesium`; Appendix M), it cannot enhance aggregate performance independently.

6 Conclusion

In this paper, we presented SGG-ReflAct, a reasoning backbone that decomposes task goals into a sub-goal plan and anchors each reflection to the active sub-goal. To investigate the impact of planning quality, we also introduced BeamSGG-ReflAct, which uses a self-evaluation-guided beam search. Across ALFWorld, ScienceWorld, and Jericho, SGG-ReflAct outperformed ReAct, Plan-and-Act, and ReflAct in nearly all settings, with the largest gains on the smallest model (e.g., $+14.9$ pp on ALFWorld and $+8.0$ pp on ScienceWorld using Llama-3.1-8B-Instruct).

Our analysis reveals two primary drivers of this improvement. First, anchoring reflections to sub-goals substantially enhances agent grounding, reducing the invalid-action rate and improving performance especially on procedurally ordered tasks. Ablation studies confirm that this benefit stems from the anchoring instruction itself rather than the mere presence of a sub-goal plan. Second, we suggest that plan content is a fundamental determinant of backbone performance. Search-based optimization only yields benefits when candidate plans explicitly include the required operations. Consequently, improving the descriptive quality of sub-goals is a necessary prerequisite for effective plan search, rather than a substitute for it.

These findings suggest a dual design principle for long-horizon agents: (i) strengthening reasoning backbones requires refining their reflection targets, and (ii) optimizing planners necessitates prioritizing sub-goal content to enable effective search.

Limitations and Future Work. Several limitations remain. First, SGG-ReflAct fixes the sub-goal plan at $t = 0$, meaning it cannot dynamically generate new sub-goals mid-episode; implementing a re-planning trigger is a natural next step. Second, some of our margins rest on few episodes: a single game is worth 5.0 points on the 20-game Jericho suite, and a single episode moves rates from splits as small as nine by 11.1 points. We also fix the plan length at $N = 4$ for every task (Section 5.1, Appendix B). Finally, our modular planner can be replaced by environment-aware models, though it incurs up to 42 LLM calls per episode.

REPRODUCIBILITY STATEMENT

All agents, environments, and evaluation code are re-implemented in a single codebase and run under identical conditions, described in Section 4. Every run shares the same base LLM per group, the same stop-word set, the same flat-text in-context learning format, and a single fixed random seed (42). Appendices E and D give the exact settings, the corrected success criteria, and the per-task step budgets. Appendix C gives the full beam-search planning algorithm, its hyperparameters ($N = 4$, $K = 2$, $S = 3$), the self-evaluation scoring values, and the self-evaluation prompt. Every system prompt, in-context example, and planner prompt is listed in Appendices N, O, P, Q, and R. The per-group results behind every aggregate in Table 1, and the dispersions it reports, are given in Appendices F and G.

ETHICS STATEMENT

This work evaluates LLM agents in three simulated text environments (ALFWorld, ScienceWorld, and Jericho). It involves no human subjects, no personal or sensitive data, and no deployment in a physical or safety-critical setting. The method changes how an agent reasons; it does not grant the agent new capabilities or tool access. We use publicly released benchmarks under their original licenses, together with commercially available or openly released models. We see no direct path from this work to harmful application beyond the general risks already associated with more capable LLM agents.

AI USE STATEMENT

We used LLM-based assistants during this project, and disclose their role below. We did not use them to generate synthetic data, to develop theoretical results or proofs, or to implement the agent methods and evaluation harness; the remaining disclosure categories in the ICLR AI policy do not apply to this work. We have reviewed all AI-assisted work. The authors take full responsibility for all content, including any text, figures, or analysis that an assistant helped produce.

Research methodology and analysis. An assistant was used to discuss and stress-test the framing of the analyses in Section 5. It proposed robustness checks—such as stratifying the invalid-action rate by episode outcome and aggregating the four state-change task types—wrote the scripts that computed them, and helped interpret the resulting numbers. The authors reviewed those scripts and checked their output against the run logs. All agent runs were designed, executed, and verified by the authors, and every number reported in this paper was computed from our own experiment logs.

Literature retrieval and citation checking. An assistant was used to retrieve and summarize the abstracts of the works we cite and to flag mismatches between a citation and the claim it supported. Every flagged case was checked by the authors against the source, and any replacement citation was verified against that source before use.

Writing. An assistant was used to shorten and simplify prose, to split long sentences, to align terminology, and to check internal consistency between the text, the tables, and the figures. It did not generate any claim, result, or citation that the authors did not verify against the underlying data or the cited source.

Code. An assistant wrote most of the analysis and plotting scripts, including the one that computes the ordered-chain length behind our choice of $N = 4$ (Appendix B); the authors reviewed them and verified their output. The agent implementations and the evaluation harness were written and checked by the authors.

REFERENCES

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings*

of the AAAI Conference on Artificial Intelligence, 2024. URL https://arxiv.org/abs/2308.09687.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.14165.

Jae-Woo Choi, Hyungmin Kim, Hyobin Ong, Youngwoo Yoon, Minsu Jang, Dohyung Kim, and Jaehong Kim. ReAcTree: Hierarchical LLM agent trees with control flow for long-horizon task planning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2026. URL https://arxiv.org/abs/2511.02424.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. TextWorld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with IJCAI 2018, Revised Selected Papers*, 2019. URL https://arxiv.org/abs/1806.11532.

Ziwei Deng, Mian Deng, Chenjing Liang, Zeming Gao, Chennan Ma, Chenxing Lin, Haipeng Zhang, Songzhu Mei, Siqi Shen, and Cheng Wang. PlanU: Large language model reasoning through planning under uncertainty. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2510.18442.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2503.09572.

Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. Preemptive detection and correction of misaligned actions in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 222–244, 2025. URL https://aclanthology.org/2025.emnlp-main.12/.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.11738.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8154–8173, 2023. URL https://arxiv.org/abs/2305.14992.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7903–7910, 2020. URL https://arxiv.org/abs/1909.05398.

Jongwon Jeong, Jungtaek Kim, and Kangwook Lee. TAPE: Tool-guided adaptive planning and constrained execution in language model agents. *arXiv preprint arXiv:2602.19633*, 2026. URL https://arxiv.org/abs/2602.19633.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. URL https://www.sciencedirect.com/science/article/pii/S000437029800023X.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. RAP: Retrieval-augmented planning with contextual memory for multimodal LLM agents. *arXiv preprint arXiv:2402.03610*, 2024. URL https://arxiv.org/abs/2402.03610.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. ReflAct: World-grounded decision making in LLM agents via goal-state reflection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 33421–33453, 2025. URL https://arxiv.org/abs/2505.15182.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. SwiftSage: A generative agent with fast and slow thinking for complex interactive tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.17390.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled architecture for autonomous LLM agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2309.17382.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2024. URL https://arxiv.org/abs/2401.13178.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.17651.

OpenAI. GPT-4o system card. https://openai.com/index/gpt-4o-system-card/, 2024. Accessed: 2026.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller LLMs stronger problem-solvers. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2408.06195.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2405.14205.

Mrinal Rawat, Ambuje Gupta, Rushil Goomer, Alessandro Di Bari, Neha Gupta, and Roberto Pieraccini. Pre-Act: Multi-step planning and reasoning improves acting in LLM agents. *arXiv preprint arXiv:2505.09970*, 2025. URL https://arxiv.org/abs/2505.09970.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.11366.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.03768.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7584–7600, 2024. URL https://arxiv.org/abs/2403.02502.

Mirac Suzgun, Mert Yüksekgönül, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 7080–7106, 2026. URL https://arxiv.org/abs/2504.07952.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 2609–2634, 2023. URL https://aclanthology.org/2023.acl-long.147/.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 2024. URL https://arxiv.org/abs/2308.11432.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022. URL https://arxiv.org/abs/2203.07540.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2201.11903.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.00633.

Weimin Xiong, Yifan Song, Qingxiu Dong, Bingchan Zhao, Feifan Song, Xun Wang, and Sujian Li. MPO: Boosting LLM agents with meta plan optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 3914–3935, 2025. URL https://arxiv.org/abs/2503.02682.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. URL https://arxiv.org/abs/2210.03629.

Qiuhai Zeng, Sarvesh Rajkumar, Di Wang, Narendra Gyanchandani, and Wenbo Yan. Reflect before act: Proactive error correction in language models. *arXiv preprint arXiv:2509.18607*, 2025. URL https://arxiv.org/abs/2509.18607.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://arxiv.org/abs/2308.10144.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2310.04406.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. KnowAgent: Knowledge-augmented planning for LLM-based agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3709–3732, 2025. URL https://aclanthology.org/2025.findings-naacl.205/.

A Framework Comparison

Table 2 places SGG-ReflAct and BeamSGG-ReflAct among the planning and reflection frameworks reviewed in Section 2. It separates the two components that prior work supplies on their own—an upfront plan and a per-turn reflection—from the coupling that makes the plan the target of that reflection, which is the axis on which our two agents differ from the rest.

Table 2: We compare agent frameworks along three design axes: **Plan**, whether the agent performs structured upfront goal decomposition; **Turn-Reflect**, whether it performs per-turn state–goal reflection; and **Plan+Reflect**, whether the two components are tightly coupled so the plan anchors the reflection. MPO provides a meta plan as context but does not modify the reflection target, so it has Plan but not Turn-Reflect or Plan+Reflect.

Method	Plan	Turn-Reflect	Plan+Reflect
ReAct (Yao et al., 2023b)	×	×	×
Reflexion (Shinn et al., 2023)	×	×	×
WKM (Qiao et al., 2024)	×	×	×
Plan-and-Act (Erdogan et al., 2025)	√	×	×
MPO (Xiong et al., 2025)	√	×	×
LATS (Zhou et al., 2024)	Tree/MCTS	×	×
ReflAct (Kim et al., 2025)	×	√	×
ToT / GoT (Yao et al., 2023a; Besta et al., 2024)	Tree/Graph	×	×
SelfEval-Beam (Xie et al., 2023)	Beam	×	×
SGG-ReflAct (ours)	Single-pass	Sub-goal	√
BeamSGG-ReflAct (ours)	Beam+SelfEval	Sub-goal	√

B Choosing the Plan Length N

For each of ScienceWorld's 24 task types, we take the environment's own gold action sequence (`generateGoldPath=True`) and compute the length of the longest chain of actions that the sequence forces into a fixed order: the sub-sequence of distinct actions that cannot be reordered without breaking a prerequisite. This length is the natural analogue of the number of ordered milestones a sub-goal plan should name, since our sub-goals are likewise ordered to respect the environment's prerequisites (Section 3.1). Table 3 gives this length for every type. Weighted by the number of test episodes each type contributes, the mean is 3.79 and the median and mode are both 4, which is why we set $N = 4$. The values range from 2 (the Lifespan and Life-Stage tasks, which only require locating and comparing organisms) to 8 (Chemistry Mix, which chains several sequential mixing steps), so $N = 4$ is a compromise tuned to the center of this distribution rather than a value that fits every type exactly.

C BeamSGG-ReflAct Planning Details

Algorithm 1 gives the full beam-search planning procedure of BeamSGG-ReflAct (Section 3.2). The self-evaluation score used in the multiplicative plan score (Equation 4) asks the judge to rate the partial plan on a continuous scale in $[0, 1]$ against a fixed rubric:

$$\textsc{SelfEval}(\rho) = \begin{cases} 0.01 & \text{duplicate} \\ \mathrm{clip}_{[0.05,\,1.0]}(v) & \text{parsed rating } v \\ 0.30 & \text{no rating parsed} \end{cases} \tag{5}$$

A candidate scores 0.01 when g_d duplicates any sub-goal in $\rho_{1:d-1}$, which is detected without an LLM call. A rating given on a 0–10 or 0–100 scale is rescaled, and the result is clipped to $[0.05, 1.0]$ so that no single weak step can zero out an entire plan. We ask for a continuous rating rather than a binary verdict because a binary judge collapses every acceptable candidate onto one value and so cannot rank candidates against one another. The rubric asks the LLM to check several criteria: (1) the sub-goal describes *what* to achieve and follows any ordering instructions in u; (2) no prerequisite step is skipped; (3) a "focus on" sub-goal appears only after the target object has been obtained; (4) the final sub-goal achieves the stated objective; and (5) variable elements are described generically. The closing instruction that requests the rating is reproduced verbatim below, and the full per-environment prompts are in Appendices P, Q, and R.

Table 3: Length of the longest forced-order action chain in the ScienceWorld gold solution, by task type, alongside the number of test episodes each type contributes.

Task type	n	Chain length
Boil	9	4
Change state of matter	9	4
Freeze	9	4
Melt	9	5
Measure melting point	10	4
Use thermometer	10	3
Power component	5	3
Power (renewable vs. non-renewable)	5	5
Test conductivity	10	6
Test conductivity (unknown)	10	6
Find animal	10	4
Find living thing	10	4
Find non-living thing	10	4
Find plant	10	4
Grow fruit	10	3
Grow plant	10	3
Chemistry mix	8	8
Mix paint (secondary)	9	4
Mix paint (tertiary)	9	2
Lifespan (longest-lived)	10	2
Lifespan (longest then shortest)	10	2
Lifespan (shortest-lived)	10	2
Identify life stages 1	5	2
Identify life stages 2	4	2

Self-evaluation: closing instruction

```
Rate how well the Partial plan satisfies the criteria above on a continuous scale from
0.0 to 1.0 (1.0 = fully correct and well-ordered, 0.0 = clearly wrong). Use the full
range: a plan that satisfies most but not all criteria should receive an intermediate
value.
Answer with a single decimal number only (e.g., 0.82). No explanation.
```

How the S candidates differ. Decoding is deterministic, so a beam's S candidates are drawn one at a time rather than in parallel, and each request names what has already been produced. The expansion prompt is followed by the in-context example, the task, the sub-goals fixed so far, up to two avoidance lines, and a closing request for step d of N:

Avoidance lines in the expansion prompt

```
Do not repeat any sub-goal already in the current plan: {sub-goals fixed so far}
Do not repeat any of the following proposed candidates: {candidates already drawn at
this depth}
```

The i-th request therefore lists the first $i-1$ candidates, which is what makes them differ under deterministic decoding. Self-evaluation is independent across candidates and runs in parallel. Before the top-K cut, candidates that reached the same sub-goal sequence are collapsed to the highest-scoring one.

D Step Budgets and Episode Termination

An episode ends when the agent solves the task, when it reaches the step budget, or—on ScienceWorld—when it emits ten consecutive actions the environment cannot parse. We cap ALFWorld and Jericho at **30** steps per episode, following AgentBoard (Ma et al., 2024) for Jericho. ScienceWorld uses a per-task budget, listed in Table 4, because its task types differ widely in how many actions a gold solution needs.

Algorithm 1 BeamSGG-ReflAct Planning

Require: Task instruction u, beam width K, samples per beam S, depth N

$\mathcal{B}_0 \leftarrow \{()\}$
for $d = 1$ **to** N **do**
  $\mathcal{C} \leftarrow \emptyset$
  for each partial plan $\rho \in \mathcal{B}_{d-1}$ **do**
    avoid_set $\leftarrow \emptyset$
    for $i = 1$ **to** S **do**
      $g_d^{(i)} \leftarrow \mathcal{M}(\text{expand_prompt},\ u,\ \rho,\ \text{avoid_set})$
      avoid_set $\leftarrow$ avoid_set $\cup \{g_d^{(i)}\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\rho,\ g_d^{(i)})\}$
    end for
  end for
  // Score all candidates in parallel via self-evaluation
  for each $\rho' \in \mathcal{C}$ **do**
    $\text{score}(\rho') \leftarrow \text{score}(\rho'_{1:d-1}) \times \textsc{SelfEval}(\rho')$
  end for
  $\mathcal{B}_d \leftarrow \text{top-}K(\mathcal{C},\ \text{score})$
end for
return $\arg\max_{\rho \in \mathcal{B}_N} \text{score}(\rho)$

ScienceWorld additionally terminates an episode after **ten consecutive** observations of `No known action matches that input`; a single valid action resets the counter. This rule caps how long an agent can loop on a phrasing the parser rejects, and it bounds the late-episode invalid-action rates reported in Section 5.1.

Responses missing the `Action:` tag trigger an `Error Input` observation rather than an environment execution, and this is uniform across all three benchmarks. Such turns consume step budget without advancing the environment state. On ScienceWorld they do not count toward the ten consecutive unparsable actions above, which are the environment's own refusals.

E Experimental Settings

This appendix records the settings summarized in Section 4. All models decode with `temperature=0` and `max_tokens=512`, and no method updates any model weights, so agents differ from one another only in their prompts and control flow. To ensure consistency, all agents share the same base LLM, stop-word set, and in-context format within a single codebase. Environments and test splits follow Kim et al. (2025) and Xiong et al. (2025) for ALFWorld and ScienceWorld and Ma et al. (2024) for Jericho, re-implemented in our own codebase rather than reusing the original code. To mitigate known success-flag bugs in ScienceWorld and Jericho (Kim et al., 2025) (ScienceWorld's `getCompleted()` can return `True` at low reward values, and Jericho shows the same false positive after the agent has died), we define success via progress rewards: at least 0.7 for ScienceWorld and exactly 1.0 for Jericho; ALFWorld uses its native binary flag. Appendix D gives the per-task step budgets, and BeamSGG-ReflAct is the one agent that issues extra model calls beyond the per-turn policy, at most 42 planner calls per episode (Appendix C).

We exclude Reflexion (Shinn et al., 2023) due to its multi-episode reset mechanism, which is incompatible with our single-episode online setting. MPO and LATS are discussed in Section 2 rather than run: Kim et al. (2025) already measure MPO in this setting and report that adding meta-plans to a reflection backbone barely changes performance, with a small drop on Llama-3.1-8B-Instruct; LATS is excluded because its environment-rollout search exceeds our budget of one call per episode with no environment interaction.

Table 4: Per-task step budgets for ScienceWorld's 24 task types. An episode also ends early on success or after ten consecutive unparsable actions.

Task Type	Max Steps
task-10-measure-melting-point-(known-substance)	120
task-1-boil	100
task-1-change-the-state-of-matter-of	80
task-1-freeze	80
task-1-melt	80
task-4-grow-fruit	60
task-5-chemistry-mix	60
task-10-use-thermometer	30
task-2-power-component-(renewable-vs-nonrenewable-energy)	30
task-2a-test-conductivity	30
task-2a-test-conductivity-of-unknown-substances	30
task-4-grow-plant	30
task-5-chemistry-mix-paint-(tertiary-color)	30
task-7-identify-life-stages-1	30
task-7-identify-life-stages-2	30
task-2-power-component	20
task-3-find-animal	15
task-3-find-living-thing	15
task-3-find-non-living-thing	15
task-3-find-plant	15
task-5-chemistry-mix-paint-(secondary-color)	15
task-6-lifespan-(longest-lived-then-shortest-lived)	12
task-6-lifespan-(longest-lived)	10
task-6-lifespan-(shortest-lived)	10

F ALFWorld Success Rate by Task Type

Table 5 breaks the ALFWorld results of Table 1 down by the benchmark's six standard task types, which we recover from the goal text of each episode. The *SD* column is the standard deviation of the six rates and is the dispersion reported alongside the ALFWorld column of Table 1. Pick Two & Place is the hardest type for every model, and it is where the weakest backbones collapse.

Table 5: We report ALFWorld success rate (%) by task type, for all three models. Columns abbreviate the six standard types: *Pick* is Pick & Place (n=24), *Look* is Look at Object in Light (n=18), *Clean* is Clean & Place (n=31), *Heat* is Heat & Place (n=23), *Cool* is Cool & Place (n=21), and *Two* is Pick Two & Place (n=17). *All* is the overall rate over the 134 episodes and matches Table 1, and *SD* is the standard deviation across the six types. We bold the best result per column within each model group.

Model	Agent	Pick	Look	Clean	Heat	Cool	Two	All	SD
GPT-4o	NoThinking	87.5	**100.0**	83.9	52.2	85.7	58.8	78.4	16.8
	ReAct	91.7	94.4	80.6	73.9	81.0	58.8	80.6	11.8
	Plan-and-Act	87.5	94.4	77.4	65.2	90.5	**76.5**	81.3	9.9
	ReflAct	**95.8**	**100.0**	**90.3**	78.3	95.2	**76.5**	89.6	8.9
	BeamSGG-ReflAct	**95.8**	88.9	87.1	78.3	95.2	52.9	84.3	14.7
	SGG-ReflAct	**95.8**	**100.0**	87.1	**82.6**	**100.0**	**76.5**	90.3	8.9
GPT-4o-mini	NoThinking	83.3	61.1	29.0	52.2	19.0	17.6	44.0	24.0
	ReAct	75.0	61.1	38.7	34.8	38.1	23.5	45.5	17.4
	Plan-and-Act	50.0	50.0	54.8	34.8	33.3	**29.4**	43.3	9.8
	ReflAct	79.2	66.7	58.1	**82.6**	61.9	17.6	62.7	21.3
	BeamSGG-ReflAct	79.2	66.7	**83.9**	69.6	**90.5**	11.8	70.1	26.0
	SGG-ReflAct	**87.5**	**77.8**	77.4	73.9	81.0	23.5	72.4	21.3
Llama-3.1 -8B	NoThinking	58.3	66.7	19.4	21.7	0.0	5.9	28.4	25.2
	ReAct	33.3	27.8	**32.3**	26.1	33.3	23.5	29.9	3.8
	Plan-and-Act	50.0	38.9	29.0	8.7	33.3	17.6	29.9	13.5
	ReflAct	66.7	**77.8**	19.4	4.3	14.3	**47.1**	35.8	27.5
	BeamSGG-ReflAct	**75.0**	72.2	16.1	**43.5**	57.1	23.5	46.3	22.5
	SGG-ReflAct	62.5	61.1	29.0	39.1	**85.7**	35.3	50.7	19.6

G SCIENCEWORLD SUCCESS RATE BY TASK CATEGORY

Table 6 breaks the ScienceWorld results of Table 1 down by the nine categories of the ScienceWorld split, which group its 24 task types by task identifier (`task-1` through `task-10`), for all three models. The *All* column and the *SD* column are the two numbers Table 1 reports for ScienceWorld: *SD* is the dispersion printed after each success rate there. Two patterns recur across models. The state-change category, which pools the Boil, Freeze, Melt, and Change-State task types, is where the spread between methods is widest, and Grow stays near zero for every agent and model, confirming that its bottleneck is the step budget rather than the reasoning backbone.

Table 6: We report ScienceWorld success rate (%) by task category, for all three models. The nine columns group the 24 ScienceWorld task types by task identifier: *St-Chg* (n=36), *Elec* (10), *Cond* (20), *Find* (40), *Grow* (20), *Chem* (26), *Lifesp* (30), *LifeSt* (9), and *Meas* (20). *All* is the overall rate over the 211 episodes and *SD* the standard deviation across the nine categories; both match the corresponding ScienceWorld entry of Table 1. We bold the best result per category column within each model group, with ties bolded both, and leave columns on which every agent ties unmarked.

Model	Agent	St-Chg	Elec	Cond	Find	Grow	Chem	Lifesp	LifeSt	Meas	All	SD
GPT-4o	NoThinking	47.2	**10.0**	15.0	82.5	0.0	73.1	80.0	**44.4**	60.0	53.6	29.4
	REACT	55.6	0.0	10.0	97.5	0.0	76.9	**86.7**	**44.4**	70.0	59.2	35.6
	Plan-and-Act	41.7	0.0	5.0	97.5	5.0	65.4	80.0	**44.4**	65.0	54.0	33.4
	REFLACT	61.1	0.0	15.0	87.5	0.0	46.2	**86.7**	**44.4**	75.0	55.5	32.7
	BeamSGG-ReflAct	**66.7**	**10.0**	**50.0**	87.5	0.0	73.1	80.0	33.3	**90.0**	63.5	31.5
	SGG-ReflAct	55.6	0.0	**50.0**	**100.0**	**10.0**	**84.6**	76.7	0.0	**90.0**	64.0	37.4
GPT-4o-mini	NoThinking	5.6	0.0	0.0	55.0	0.0	3.8	60.0	11.1	0.0	20.9	23.0
	REACT	25.0	0.0	**15.0**	**100.0**	0.0	30.8	73.3	0.0	20.0	40.8	33.0
	Plan-and-Act	16.7	0.0	5.0	97.5	0.0	7.7	**76.7**	**22.2**	15.0	36.0	33.4
	REFLACT	41.7	0.0	10.0	**100.0**	0.0	3.8	73.3	0.0	25.0	40.3	34.5
	BeamSGG-ReflAct	25.0	0.0	10.0	85.0	0.0	61.5	73.3	0.0	**45.0**	43.6	31.9
	SGG-ReflAct	**52.8**	0.0	0.0	95.0	0.0	**65.4**	60.0	11.1	10.0	45.0	33.8
Llama-3.1 -8B	NoThinking	2.8	0.0	10.0	55.0	0.0	0.0	3.3	0.0	10.0	13.3	16.7
	REACT	2.8	0.0	15.0	45.0	**5.0**	15.4	40.0	0.0	0.0	18.5	16.4
	Plan-and-Act	0.0	0.0	**20.0**	37.5	**5.0**	7.7	3.3	0.0	10.0	11.8	11.7
	REFLACT	11.1	0.0	0.0	70.0	**5.0**	3.8	**66.7**	0.0	0.0	25.6	27.5
	BeamSGG-ReflAct	16.7	0.0	15.0	70.0	0.0	3.8	53.3	0.0	10.0	26.5	24.0
	SGG-ReflAct	**22.2**	0.0	0.0	**77.5**	**5.0**	**42.3**	53.3	0.0	**20.0**	33.6	26.2

Figure 3 regroups the same episodes into ten reporting buckets rather than these nine categories, so that the state-change category appears as its four task types and the paint-mixing episodes appear on their own; the remaining categories are pooled into *Other*. Table 7 gives the finer split the figure is built from, for the same model and the same four backbones. Reading it beside the figure makes two things visible that the pooled *Other* bar hides. First, BeamSGG-ReflAct's nonzero rate on *Other* comes from four episodes out of forty-seven: three electrical or conductivity episodes and one non-paint chemistry episode. Second, no agent solves a single life-stage episode, so that category does not separate the two planners at all. On the state-change tasks, BeamSGG-ReflAct solves one of nine Melt episodes and SGG-ReflAct none; these are the tasks where Section 5.2 shows that stating the required procedure in the plan, rather than the planner that produced it, drives the difference.

H REFLECTION ANCHORING: REFLACT+PLAN VS. SGG-REFLACT (EXAMPLE)

This example accompanies the ablation of Section 5.1. Both agents receive a sub-goal plan for the same ScienceWorld episode (Llama-3.1-8B-Instruct, `boil mercury`) from the same single-path planner, generated independently for each agent, and differ in how the per-turn reflection is instructed and demonstrated. REFLACT+Plan scores 0.05 and SGG-ReflAct solves the task. The instructions below are the ScienceWorld ones (Appendix N), quoted without the output-format clause they share. REFLACT+Plan carries REFLACT's released wording unchanged, so the contrast is between two complete per-turn instructions given plans from the same planner.

The per-turn instruction.

Table 7: We report ScienceWorld success rates (%) with Llama-3.1-8B-Instruct at the granularity Figure 3 is drawn from: the state-change category as its four task types, the chemistry category as its paint and non-paint episodes, and every other row one of those categories. *Electricity* pools the power-component and conductivity categories, as the figure does. The figure's *Paint / Mix* bar is the *Paint / Mix* row here, and its *Other* bar pools *Electricity*, *Life Stages*, and *Chemistry mix*. Totals match Table 1; # zero counts the rows an agent never solves. We bold the best result per row and leave rows on which no agent succeeds unmarked.

Task Type	n	REACT	REFLACT	BeamSGG-ReflAct	SGG-ReflAct
Boil	9	0.0	0.0	22.2	**55.6**
Freeze	9	0.0	0.0	0.0	0.0
Melt	9	0.0	**33.3**	11.1	0.0
Change-State	9	11.1	11.1	**33.3**	**33.3**
Paint / Mix	18	22.2	5.6	0.0	**61.1**
Chemistry mix	8	0.0	0.0	**12.5**	0.0
Find (classification)	40	45.0	70.0	70.0	**77.5**
Measure	20	0.0	0.0	10.0	**20.0**
Electricity	30	**10.0**	0.0	**10.0**	0.0
Grow	20	**5.0**	**5.0**	0.0	**5.0**
Lifespan	30	40.0	**66.7**	53.3	53.3
Life Stages	9	0.0	0.0	0.0	0.0
Total	211	18.5	25.6	26.5	**33.6**
SD across rows		15.5	25.1	21.8	28.0
# zero rows		6	6	**4**	5

REFLACT+Plan (end-goal reflection)

"For each of your turns, you will be given the observation from the previous turn. You should first reflect on the agent's state, including the location, inventory, and focused object, ***in relation to the task goal****. Then, output the action for this turn. . . . "*

SGG-ReflAct (sub-goal anchored)

"For each of your turn, you will be given the observation of the last turn. ***Referring to the provided sub-goals, determine which sub-goal you are currently working on based on your progress so far.*** *Then reflect in one sentence on your current state and* ***progress toward the current sub-goal****, and then output the action for this turn. . . . "*

Resulting reflections. The plans the two agents receive are near-identical in content, both ending in a heating step. Their reflections are not.

REFLACT+Plan rollout, fails (score 0.05)

Plan given: 1. Find the container holding mercury; 2. Focus on the mercury; 3. Take actions that will cause the mercury to boil or combust; 4. Observe the resulting state of the mercury.

1. "To solve this task, I need to find a container with mercury, focus on it, and then take actions to change its state of matter . . . " *(restates the whole plan)*
2. "Currently, I am in the bathroom, holding nothing and focused on nothing. There's no sign of mercury here . . . "

⋮

22–26. "Currently, I am in the foundry, holding nothing and focused on the mercury . . . The blast furnace is activated but still closed." *(identical reflection repeated for the last five turns)*

The agent never names the active sub-goal. From turn 22 it freezes on one reflection and emits several actions in a single turn, and the episode ends without heating the mercury.

SGG-ReflAct rollout, succeeds (score 1.00)

Plan given: 1. Find and obtain mercury; 2. Focus on the mercury; 3. Cause the mercury to boil or combust; 4. Focus on the resulting vapor or ash.

1. "**My current sub-goal is (1) Find and obtain mercury.** Currently, I am in the art studio . . . "

⋮

11. "**My current sub-goal is (3) Cause the mercury to boil or combust.** . . . I should use heat — I see a blast furnace in the foundry."
13. "**My current sub-goal is (3)** . . . With the blast furnace activated, I should place the mercury inside it to heat it up."
22. "**My current sub-goal is (3)** . . . With the blast furnace open, I should place the mercury inside it."

The agent names the active sub-goal on every turn and advances through the plan. It is not flawless, and turns 15–21 struggle with how to operate the furnace switch, but because the reflection stays fixed on a named target it keeps proposing *different* actions toward that target instead of freezing, and it recovers and completes the task.

Across the run, REFLACT+Plan's reflections restate the overall task in this way, while SGG-ReflAct's name the active sub-goal.

I ADDITIONAL GROUNDING AND PLAN-QUALITY ANALYSES

Invalid-Action Metric as a Grounding Proxy. A rejected action might be a formatting slip rather than a genuine grounding error, so we verify four things. First, the pattern holds under Kim et al. (2025)'s broader hallucinated-action definition, which counts *all* erroneous observations, covering ambiguous, malformed, inapplicable, and precondition-violating actions: pooled across the three models, the rate still orders REACT (39.0%) $\approx$ REFLACT (38.8%) above SGG-ReflAct (32.5%) and BeamSGG-ReflAct (32.3%), with parse failures the dominant contributor. Second, it varies sharply across backbones on the *same* base model (42.9% for REFLACT versus 30.3% for SGG-ReflAct on Llama-3.1-8B-Instruct) and rises over the course of an episode—neither pattern is explained by formatting competence, which is fixed by the base model and does not degrade mid-episode. Third, 22–32% of invalid actions on GPT-4o and Llama-3.1-8B-Instruct repeat an action the environment has *already* rejected, which shows that the agent treats an infeasible action as valid rather than mistyping a one-off command. Finally, the rise over an episode is not an artifact of which episodes succeed: within REFLACT's successful episodes the rate goes from 2.6% to 37.0%, and within its failed ones from 16.3% to 76.8%. We nonetheless read the metric as a proxy, and leave fine-grained annotation of syntax-versus-grounding failures to future work. Formally, for a set $\mathcal{D}$ of executed-action observations, the rate is $\mathrm{IA}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{o \in \mathcal{D}} 1[\, o \text{ is a refusal} \,]$.

The Ordering Rule's Effect. We examine whether the ordering rule affects only the 36 targeted state-change episodes or spills over into the remaining 175 (Tables 14 and 8). In three of the six model/planner pairs, changes occur primarily within the targeted episodes. GPT-4o's single-path planner gains six target episodes and loses one non-target episode, and its Beam Search Planner gains three target and one non-target episode. Llama-3.1-8B-Instruct's single-path planner loses two target and one non-target episode. In the other three pairs, most of the change falls outside the targeted episodes. GPT-4o-mini's single-path planner is the sharpest case, gaining one target episode against nine non-target episodes (ten additional successes overall). Llama-3.1-8B-Instruct's Beam Search Planner gains four target and six non-target episodes, and GPT-4o-mini's Beam Search Planner gains eleven target episodes while giving back three non-target episodes.

Category Volatility and Cancellation. Even when the net change outside state-change tasks is near zero, sizeable internal shifts occur. GPT-4o's single-path planner loses four *Find* episodes while gaining three *Lifespan* and two *Life Stages* episodes. Llama-3.1-8B-Instruct's single-path planner loses six *Find* episodes while gaining four *Measurement* and three *chemistry-mixing* episodes. Which categories move is inconsistent across models: GPT-4o's single-path planner picks up *Life Stages* (0.0 to 22.2), *Lifespan* (76.7 to 86.7), and *Electricity* (0.0 to 10.0), while its Beam Search

Planner shows minimal non-target movement. The sharpest divergence is on *Measurement* for GPT-4o-mini, where the single-path planner rises from 10.0% to 50.0% while the Beam Search Planner falls from 45.0% to 15.0%; we attribute this reversal to the category's small sample size (20 episodes), where a handful of flipped episodes moves the rate by ten points or more at a time.

Interpretation and Limitations. These results suggest that the ordering rule acts as a general prompt for sub-goal specificity and structure, rather than only teaching one specific procedure. Our current ablation cannot, however, decouple whether the rule teaches the model a specific state-change fact or improves its general planning competence, and we leave this distinction as a question for future work.

Table 8: We report the effect of the state-change ordering rule (Table 14) on every ScienceWorld category. The columns are the same nine categories as Table 6: *St-Chg* pools the four Boil/Freeze/Melt/Change-State task types (36 episodes), matching Table 14's State-change column, and *Chem* pools all 26 chemistry-mixing episodes, paint and non-paint together. *base* and *+proc* repeat Table 14's columns, and we bold the higher of each base/+proc pair, with ties bolded both.

Model	Planner	St-Chg	Elec	Cond	Find	Grow	Chem	Lifesp	LifeSt	Meas	All
GPT-4o	SGG base	55.6	0.0	**50.0**	**100.0**	**10.0**	**84.6**	76.7	0.0	**90.0**	64.0
	SGG +proc	**72.2**	**10.0**	40.0	90.0	5.0	**84.6**	**86.7**	**22.2**	**90.0**	**66.4**
	Beam base	66.7	**10.0**	**50.0**	**87.5**	0.0	**73.1**	80.0	**33.3**	**90.0**	63.5
	Beam +proc	**75.0**	**10.0**	45.0	**87.5**	**10.0**	**73.1**	**83.3**	**33.3**	85.0	**65.4**
GPT-4o-mini	SGG base	52.8	**0.0**	0.0	**95.0**	**0.0**	**65.4**	60.0	**11.1**	10.0	45.0
	SGG +proc	**55.6**	**0.0**	**10.0**	90.0	**0.0**	**65.4**	**66.7**	0.0	**50.0**	**49.8**
	Beam base	25.0	**0.0**	**10.0**	85.0	**0.0**	61.5	**73.3**	**0.0**	**45.0**	43.6
	Beam +proc	**55.6**	**0.0**	**10.0**	**87.5**	**0.0**	**69.2**	**73.3**	**0.0**	15.0	**47.4**
Llama-3.1 -8B	SGG base	**22.2**	**0.0**	0.0	**77.5**	**5.0**	42.3	**53.3**	**0.0**	20.0	**33.6**
	SGG +proc	16.7	**0.0**	**10.0**	62.5	0.0	**53.8**	43.3	**0.0**	**40.0**	32.2
	Beam base	16.7	**0.0**	**15.0**	70.0	0.0	3.8	**53.3**	**0.0**	10.0	26.5
	Beam +proc	**27.8**	**0.0**	5.0	**75.0**	**5.0**	**19.2**	**53.3**	**0.0**	**15.0**	**31.3**

Search Configurations at a Comparable Budget. Comparison of the greedy and Beam Search Planners at a comparable budget (Table 9) shows near-identical performance across most categories: they agree exactly on six of the nine and differ by at most four points on the rest. The main exception is chemistry mixing, where the single-path planner (42.3%) substantially outperforms the Beam Search Planner (3.8%) and its greedy variant (7.7%). Invalid-action rates follow the same ranking as success rates (Beam: 34.2%, Greedy: 32.4%, single-path: 29.9%). Given how little the greedy and beam variants differ, we read them as one observation rather than two, since a single seed separates their success rates of 27.0% and 26.5%.

Table 9: We compare greedy versus beam search at a comparable planner budget, on ScienceWorld with Llama-3.1-8B-Instruct. The Beam Search Planner ($K = 2$, $S = 3$) spends at most 42 planner calls per episode, and its greedy variant ($K = 1$, $S = 6$) at most 48 (Section 3.2). All three rows use the base planner prompt, so the two rows that also appear in Table 1 match it, and column names follow Table 8. *IA* is the invalid-action rate, where lower is better; we bold the best result per column, with ties bolded both, and leave columns on which all three planners tie unmarked.

Planner	St-Chg	Elec	Cond	Find	Grow	Chem	Lifesp	LifeSt	Meas	SR / AR	IA
Beam (K=2, S=3)	16.7	0.0	**15.0**	70.0	0.0	3.8	53.3	0.0	10.0	26.5 / 41.9	34.2
Greedy (K=1, S=6)	13.9	0.0	**15.0**	72.5	0.0	7.7	53.3	0.0	10.0	27.0 / 41.3	32.4
Single pass	**22.2**	0.0	0.0	**77.5**	**5.0**	**42.3**	53.3	0.0	**20.0**	**33.6 / 48.7**	**29.9**

Trajectory Length. To ensure SGG-ReflAct's lower invalid-action rate is not simply due to shorter trajectories, we compare mean steps to success on shared episodes (Table 10), reported across all models and benchmarks on episodes that *both* REFLACT and SGG-ReflAct solve. Step counts remain comparable, within about one step, across models, though SGG-ReflAct occasionally requires fewer (e.g., ScienceWorld on GPT-4o-mini and ALFWorld on Llama-3.1-8B-Instruct). This indicates that SGG-ReflAct succeeds by completing *more* tasks rather than merely solving them faster; results for Jericho are inconclusive given its limited episode overlap.

Table 10: We report steps to success, the mean number of environment steps, on episodes solved by *both* REFLACT and SGG-ReflAct. n is the number of such commonly-solved episodes.

Model	Benchmark	n	REFLACT	SGG-ReflAct
GPT-4o	ALFWorld	115	11.2	11.5
	ScienceWorld	105	13.6	14.0
	Jericho	7	17.0	19.3
GPT-4o-mini	ALFWorld	78	10.6	10.7
	ScienceWorld	69	17.1	14.4
	Jericho	2	13.5	12.5
Llama-3.1 -8B-Instruct	ALFWorld	35	10.1	9.0
	ScienceWorld	33	9.1	9.7
	Jericho	1	23.0	8.0

J GROUNDING: A HALLUCINATED-ACTION CASCADE (EXAMPLE)

This ScienceWorld episode (Llama-3.1-8B-Instruct, `make violet paint`) traces step by step the compounding of hallucinated actions quantified in Section 5.1. REFLACT and SGG-ReflAct face the identical task and initial state; only the reasoning backbone differs. The task is to use chemistry to create violet paint and then focus on it.

REFLACT rollout, fails with 9 of 15 actions rejected

1. `teleport to art studio` → ok
2. `look around` → ok (paints are in cups)
3. `open large cupboard` → ok
4. `look at drawer` → ok
5. `pick up cup containing yellow paint` → **Ambiguous request**; the agent disambiguates on the next turn
6. `1` → the wood cup moves to the inventory

7–15. nine consecutive `pour`/`use` commands, each malformed and each rejected with **No known action matches that input**; no paint is ever mixed.

REFLACT fixates on a phrasing the environment cannot parse and repeats it for the rest of the episode, so its per-turn reflection never recovers the agent to a valid action.

SGG-ReflAct rollout, succeeds with 0 of 8 actions rejected

Plan: *(1) teleport to the room with the materials; (2) gather the materials; (3) mix them into violet paint; (4) focus on the result.*

1. `teleport to art studio` → ok
2. `look around` → ok
3. `pick up wood cup containing red paint` → moved to inventory
4. `pick up wood cup containing blue paint` → moved to inventory
5. `pour wood cup containing blue paint into bowl` → ok
6. `pour wood cup containing red paint into bowl` → ok
7. `mix bowl` → "Red and blue paint mix to produce violet paint."
8. `focus on violet paint` → **success**

The active sub-goal anchors the agent, first to *gather the materials* and then to *mix them*. SGG-ReflAct therefore issues only well-formed, grounded actions and finishes in eight steps. Nine rejected actions versus none, on the same task, is the episode-level signature of the aggregate invalid-action gap in Table 12.

K ACTIONS PER EPISODE

Table 11 gives the mean number of actions each agent executes per ScienceWorld episode, over the same 211 episodes as Table 1. It supports the argument in Section 5.1 that the agents with the lowest invalid-action rates are not the ones that interact least: on every model, NoThinking and Plan-and-Act take more actions than SGG-ReflAct, by 3.7 to 7.5. BeamSGG-ReflAct takes as few actions as SGG-ReflAct or fewer, yet solves less, which is why a low count is not on its own a sign of an efficient agent.

Table 11: Mean actions executed per episode on ScienceWorld (211 episodes). Lower is not better on its own; read alongside the success rates in Table 1 and the invalid-action rates in Table 12.

Model	Agent	Actions / episode
GPT-4o	NoThinking	24.7
	REACT	22.6
	Plan-and-Act	28.3
	REFLACT	20.2
	BeamSGG-ReflAct	21.6
	SGG-ReflAct	20.8
GPT-4o-mini	NoThinking	25.6
	REACT	23.2
	Plan-and-Act	25.3
	REFLACT	23.1
	BeamSGG-ReflAct	20.4
	SGG-ReflAct	20.3
Llama-3.1 -8B-Instruct	NoThinking	22.8
	REACT	21.5
	Plan-and-Act	23.4
	REFLACT	19.9
	BeamSGG-ReflAct	17.5
	SGG-ReflAct	19.1

L INVALID-ACTION RATE FOR ALL AGENTS

Table 12 gives the full invalid-action rates behind Figure 2, for all six agents on all three models, overall and by episode position.

M SUB-GOAL PLANS: SGG-REFLACT VS. BEAMSGG-REFLACT (EXAMPLES)

The following are real plans (Llama-3.1-8B-Instruct, ScienceWorld) that illustrate why the two planners induce different success distributions (Section 5.2). Consider `melt caesium`, a task type on which SGG-ReflAct scores 0% while BeamSGG-ReflAct succeeds; the two plans differ only in how the transformation is expressed.

`melt caesium`: SGG-ReflAct (fails) vs. BeamSGG-ReflAct (succeeds)

SGG-ReflAct (single-path planner), *fails*:

1. Find and obtain caesium
2. Focus on the caesium
3. *Take actions to increase the temperature of the caesium*
4. *Focus on the melted caesium*

BeamSGG-ReflAct (Beam Search Planner), *succeeds*:

1. Find and obtain the caesium
2. Focus on the caesium
3. **Find the heat source to melt the caesium**

Table 12: We report the invalid-action rate (%) on ScienceWorld for all six agents: the fraction of executed actions the environment refuses to interpret or apply, where lower is better, overall and by early/mid/late episode thirds (equal thirds of the actions each episode executes, pooled across episodes). **Bold** marks the lowest rate per model group per column; a low rate marks the least-rejected agent, not the best one (Section 5.1).

Model	Agent	Overall	Early	Mid	Late
GPT-4o	NoThinking	**20.9**	15.1	**22.5**	25.0
	REACT	35.3	23.1	41.1	41.7
	Plan-and-Act	23.3	17.8	27.1	**24.8**
	REFLACT	34.2	19.0	40.7	42.9
	BeamSGG-ReflAct	24.9	14.6	30.8	29.3
	SGG-ReflAct	27.7	**11.2**	35.6	36.3
GPT-4o-mini	NoThinking	26.6	21.8	28.5	29.5
	REACT	31.9	22.9	37.8	35.0
	Plan-and-Act	**24.4**	19.7	**28.2**	**25.4**
	REFLACT	32.7	23.3	35.7	39.0
	BeamSGG-ReflAct	30.0	19.0	31.9	39.0
	SGG-ReflAct	32.6	**17.7**	34.4	45.5
Llama-3.1 -8B-Instruct	NoThinking	45.5	20.4	43.8	72.2
	REACT	44.9	18.6	47.4	68.7
	Plan-and-Act	39.5	17.2	39.0	62.4
	REFLACT	43.3	14.5	44.1	71.4
	BeamSGG-ReflAct	34.2	9.3	36.6	56.9
	SGG-ReflAct	**29.9**	**8.7**	**31.2**	**49.7**

4. **Apply heat to cause the state change of the caesium**

Here SGG-ReflAct's plan restates the goal ("increase the temperature") and then passively focuses on the result, without ever naming an executable transformation. In the corresponding rollout, the agent locates the blast furnace but, lacking a stated "apply heat" step, wanders (teleport / look / examine) and exhausts its step budget without melting the substance. BeamSGG-ReflAct instead decomposes the transformation into "find a heat source" and "apply heat," which the agent can directly execute.

The reverse holds on task types where SGG-ReflAct already excels, which exposes the *cost* of the search. On `boil mercury`, for instance, SGG-ReflAct's concise plan succeeds while BeamSGG-ReflAct's more elaborate one fails.

`boil mercury`: SGG-ReflAct (succeeds) vs. BeamSGG-ReflAct (fails)

SGG-ReflAct, *succeeds*:

1. Find and obtain mercury
2. Focus on the mercury
3. **Cause the mercury to boil or combust**
4. Focus on the resulting vapor or ash

BeamSGG-ReflAct, *fails*:

1. Find and obtain mercury
2. *Find the container holding mercury*
3. Focus on the mercury
4. *Reach or obtain the heat source to apply to the mercury*

Here the search *over*-decomposes: extra "find the container" and "find a heat source" steps mislead the agent and give up the outcome that SGG-ReflAct's direct plan attains. The two cases point the same way. What the plan says, not whether it was found by search, decides the outcome; naming the missing operation helps (Melt), while adding needless steps hurts (Boil).

SGG-ReflAct does not always restate the goal. On `make violet paint`, a task type where it scores highly, its plan names the operation ("*Mix the materials to create violet paint*") and it succeeds, so what its plans supply varies by task type. The search in BeamSGG-ReflAct is itself imperfect as well: on `freeze tin` it retrieves "*find… a cold source*" but then instructs "*Apply heat to cause the state change*," and this wrong-direction step is one reason BeamSGG-ReflAct still fails a majority of Freeze episodes.

N AGENT PROMPTS (ALFWORLD)

All prompts are reproduced verbatim from our source code, preserving all original typographical errors. For ALFWorld, agents use the template below with a {`agent_instruction`} slot; SGG-ReflAct and BeamSGG-ReflAct share the same agent instruction, differing only in how the sub-goal plan is generated (Appendices P, Q, and R). ScienceWorld and Jericho follow analogous templates, their action lists omitted here for space. We also reproduce Kim et al. (2025)'s instructions as released, noting that our ScienceWorld instruction reads "for each of your turns … from the previous turn" against their "for each of your turn … of the last turn" (Appendix H). Our instructions remain consistent across ALFWorld and ScienceWorld, adding only a one-line role prefix for Jericho to ensure the backbone is not tuned per environment. That Jericho variant is the only instruction of ours not shown below, and it is given at the end of this appendix.

Shared household prompt template

```
Interact with a household to solve a task. Imagine you are an intelligent agent in a
household environment and your target is to perform actions to complete the task goal.
At the beginning of your interactions, you will be given the detailed description of
the current environment and your goal to accomplish.
{agent_instruction}
The available actions are:
1. go to {recep}
2. take {obj} from {recep}
3. put {obj} in/on {recep}
4. open {recep}
5. close {recep}
6. use {obj}
7. clean {obj} with {recep}
8. heat {obj} with {recep}
9. cool {obj} with {recep}
where {obj} and {recep} correspond to objects and receptacles.
After your each turn, the environment will give you immediate feedback based on which
you plan your next few steps. if the envrionment output "Nothing happened", that means
the previous action is invalid and you should try more options.
Reminder:
The action must be chosen from the given available actions. Any actions except provided
available actions will be regarded as illegal.
```

Plan-and-Act uses a step-0 variant of this template that omits the "After your each turn…" feedback sentence.

NoThinking instruction

```
For each of your turn, you will be given the observation of the last turn.
You should directly output the action in this turn. Your output must strictly follow
this format:"Action: your next action".
```

REACT instruction

```
For each of your turn, you will be given the observation of the last turn. You should
first think about the current condition and plan for your future actions, and then
output your action in this turn. Your output must strictly follow this format:"Thought:
your thoughts.
Action: your next action". Remember that you can only output one "Action:" in per
response.
```

Plan-and-Act instruction (step ≥ 1)

```
For each of your turn, you will be given the observation of the last turn.
You should directly output the action in this turn. Your output must strictly follow
this format:"Action: your next action".
```

Plan-and-Act planning instruction (step 0)

```
You should first think about the given task and plan your approach to the task, and
then output the action for this turn.
```

REFLACT instruction

```
For each of your turn, you will be given the observation of the last turn. You should
first reflect in one sentence on the agent's state in relation to the task goal, and
then output the action for this turn. Your output must strictly follow this
format:"Reflection: your reflection.
Action: your next action".
```

SGG-ReflAct / BeamSGG-ReflAct instruction

```
For each of your turn, you will be given the observation of the last turn. Referring to
the provided sub-goals, determine which sub-goal you are currently working on based on
your progress so far. Then reflect in one sentence on your current state and progress
toward the current sub-goal, and then output the action for this turn. Your output must
strictly follow this format:"Reflection: your reflection.
Action: your next action".
```

Jericho instruction.

Our agents' instruction is unchanged from the ALFWorld one above on ScienceWorld; on Jericho it gains the role prefix that benchmark's prompts carry.

SGG-ReflAct / BeamSGG-ReflAct — Jericho

```
You are a game master in fictional text games. For each of your turn, you will be given
the observation of the last turn. Referring to the provided sub-goals, determine which
sub-goal you are currently working on based on your progress so far. Then reflect in
one sentence on your current state and progress toward the current sub-goal, and then
output the action for this turn. Your output must strictly follow this
format:"Reflection: your reflection.
Action: your next action".
```

O SGG-REFLACT IN-CONTEXT EXAMPLES

The single-shot in-context example given to the SGG-ReflAct agent (shared with BeamSGG-ReflAct), one per environment; ALFWorld provides one example per task type.

ALFWorld — `pick_and_place`

```
USER: You are in the middle of a room. Looking quickly around you, you see a cabinet 4,
a cabinet 3, a cabinet 2, a cabinet 1, a countertop 1, a garbagecan 1, a
handtowelholder 2, a handtowelholder 1, a sinkbasin 2, a sinkbasin 1, a toilet 1, a
toiletpaperhanger 1, and a towelholder 1.
Your task is to: put some spraybottle on toilet.

USER: Here is your sub goals to complete the task:
1. Locate the required object
2. Obtain the required object
3. Bring the object to the target receptacle
4. Place the object to complete the task

ASSISTANT: Reflection: My current sub-goal is Locate the required object. Currently, I
am in the middle of the room, not holding anything. A spraybottle is likely in the
cabinets -- I should check cabinet 1 first.
Action: go to cabinet 1
```

```
USER: Observation: On the cabinet 1, you see a cloth 1, a soapbar 1, a soapbottle 1.

ASSISTANT: Reflection: My current sub-goal is Locate the required object. Currently, I
am at cabinet 1, not holding anything. There is no spraybottle here -- I should check
the next cabinet.
Action: go to cabinet 2

USER: Observation: The cabinet 2 is closed.

ASSISTANT: Reflection: My current sub-goal is Locate the required object. Currently, I
am at cabinet 2, not holding anything. The cabinet is closed -- I need to open it to
look inside.
Action: open cabinet 2

USER: Observation: You open the cabinet 2. The cabinet 2 is open. In it, you see a
candle 1, and a spraybottle 2.

ASSISTANT: Reflection: The sub-goal is complete -- I found a spraybottle 2 in cabinet
2. My current sub-goal is Obtain the required object. Currently, I am at cabinet 2, not
holding anything -- I should take the spraybottle now.
Action: take spraybottle 2 from cabinet 2

USER: Observation: You pick up the spraybottle 2 from the cabinet 2.

ASSISTANT: Reflection: The sub-goal is complete -- I now hold the spraybottle 2. My
current sub-goal is Bring the object to the target receptacle. Currently, I am at
cabinet 2, holding the spraybottle -- I should go to the toilet, the target receptacle.
Action: go to toilet 1

USER: Observation: On the toilet 1, you see a soapbottle 2.

ASSISTANT: Reflection: The sub-goal is complete -- I have arrived at toilet 1 with the
spraybottle. My current sub-goal is Place the object to complete the task. Currently, I
am at toilet 1, holding the spraybottle -- I should place it now.
Action: put spraybottle 2 in/on toilet 1

USER: Observation: You put the spraybottle 2 in/on the toilet 1.
```

ALFWorld — `pick_clean_then_place`

```
USER: You are in the middle of a room. Looking quickly around you, you see a cabinet
13, a cabinet 12, a cabinet 11, a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, a
cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a
coffeemachine 1, a countertop 1, a diningtable 1, a drawer 1, a fridge 1, a garbagecan
1, a microwave 1, a shelf 3, a shelf 2, a shelf 1, a sinkbasin 1, a stoveburner 4, a
stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: put a clean lettuce in diningtable.

USER: Here is your sub goals to complete the task:
1. Find and obtain the required object
2. Clean the object at the cleaning station
3. Bring the cleaned object to the target receptacle
4. Place the object to complete the task

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am in the middle of the room, not holding anything. Lettuce is often kept
cold, so I should check the fridge first.
Action: go to fridge 1

USER: Observation: The fridge 1 is closed.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at fridge 1, not holding anything. The fridge is closed -- I need to
open it to look inside.
Action: open fridge 1

USER: Observation: You open the fridge 1. The fridge 1 is open. In it, you see a cup 3,
a egg 2, a potato 3, and a potato 2.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at fridge 1, not holding anything. There is no lettuce in the fridge --
I should check the diningtable next.
Action: go to diningtable 1
```

```
USER: Observation: On the diningtable 1, you see a apple 1, a bread 1, a butterknife 2,
a cup 2, a fork 2, a knife 2, a knife 1, a ladle 1, a lettuce 1, a mug 2, a mug 1, a
pan 2, a peppershaker 1, a spatula 3, a tomato 2, and a tomato 1.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at diningtable 1, not holding anything. I see the lettuce 1 here -- I
should take it.
Action: take lettuce 1 from diningtable 1

USER: Observation: You pick up the lettuce 1 from the diningtable 1.

ASSISTANT: Reflection: The sub-goal is complete -- I now hold the lettuce 1. My current
sub-goal is Clean the object at the cleaning station. Currently, I am at diningtable 1,
holding the lettuce -- I should go to the sinkbasin to clean it.
Action: go to sinkbasin 1

USER: Observation: On the sinkbasin 1, you see a apple 2, a ladle 2, a spoon 1, and a
tomato 3.

ASSISTANT: Reflection: My current sub-goal is Clean the object at the cleaning station.
Currently, I am at sinkbasin 1, holding the lettuce -- I should clean it now.
Action: clean lettuce 1 with sinkbasin 1

USER: Observation: You clean the lettuce 1 using the sinkbasin 1.

ASSISTANT: Reflection: The sub-goal is complete -- the lettuce is now clean. My current
sub-goal is Bring the cleaned object to the target receptacle. Currently, I am at
sinkbasin 1, holding the clean lettuce -- I should go to the diningtable.
Action: go to diningtable 1

USER: Observation: On the diningtable 1, you see a apple 1, a bread 1, a butterknife 2,
a cup 2, a fork 2, a knife 2, a knife 1, a ladle 1, a mug 2, a mug 1, a pan 2, a
peppershaker 1, a spatula 3, a tomato 2, and a tomato 1.

ASSISTANT: Reflection: The sub-goal is complete -- I have arrived at diningtable 1 with
the clean lettuce. My current sub-goal is Place the object to complete the task.
Currently, I am at diningtable 1, holding the clean lettuce -- I should place it now.
Action: put lettuce 1 in/on diningtable 1

USER: Observation: You put the lettuce 1 in/on the diningtable 1.
```

ALFWorld — `pick_heat_then_place`

```
USER: You are in the middle of a room. Looking quickly around you, you see a cabinet
10, a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a
cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 3, a countertop 2,
a countertop 1, a diningtable 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a
drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin 1, a
stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: heat some egg and put it in diningtable.

USER: Here is your sub goals to complete the task:
1. Find and obtain the required object
2. Heat the object at the heating appliance
3. Bring the heated object to the target receptacle
4. Place the object to complete the task

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am in the middle of the room, not holding anything. An egg might be in the
fridge -- I should open it to check.
Action: open fridge 1

USER: Observation: You open the fridge 1. The fridge 1 is open. In it, you see a
lettuce 2, a mug 2, and a potato 3.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at fridge 1, not holding anything. There is no egg in the fridge -- I
should check the countertops.
Action: go to countertop 1

USER: Observation: On the countertop 1, you see a bread 1, a fork 1, and a saltshaker 1.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at countertop 1, not holding anything. No egg here -- I should check
another countertop.
Action: go to countertop 3
```

```
USER: Observation: On the countertop 3, you see a bowl 1, a butterknife 1, a egg 2, a
kettle 2, a plate 1, a sink 1, and a spatula 2.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at countertop 3, not holding anything. I see the egg 2 here -- I should
take it.
Action: take egg 2 from countertop 3

USER: Observation: You pick up the egg 2 from the countertop 3.

ASSISTANT: Reflection: The sub-goal is complete -- I now hold the egg 2. My current
sub-goal is Heat the object at the heating appliance. Currently, I am at countertop 3,
holding the egg -- I should go to the microwave to heat it.
Action: go to microwave 1

USER: Observation: The microwave 1 is closed.

ASSISTANT: Reflection: My current sub-goal is Heat the object at the heating appliance.
Currently, I am at microwave 1, holding the egg -- I should heat the egg with the
microwave now.
Action: heat egg 2 with microwave 1

USER: Observation: You heat the egg 2 using the microwave 1.

ASSISTANT: Reflection: The sub-goal is complete -- the egg is now heated. My current
sub-goal is Bring the heated object to the target receptacle. Currently, I am at
microwave 1, holding the heated egg -- I should go to the diningtable.
Action: go to diningtable 1

USER: Observation: On the diningtable 1, you see a apple 2, a bread 3, a egg 1, a
kettle 1, a knife 1, a mug 1, a papertowelroll 1, a peppershaker 2, a potato 1, a
soapbottle 1, and a spatula 1.

ASSISTANT: Reflection: The sub-goal is complete -- I have arrived at diningtable 1 with
the heated egg. My current sub-goal is Place the object to complete the task.
Currently, I am at diningtable 1, holding the heated egg -- I should place it now.
Action: put egg 2 in/on diningtable 1

USER: Observation: You put the egg 2 in/on the diningtable 1.
```

ALFWorld — `pick_cool_then_place`

```
USER: You are in the middle of a room. Looking quickly around you, you see a cabinet
16, a cabinet 15, a cabinet 14, a cabinet 13, a cabinet 12, a cabinet 11, a cabinet 10,
a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet
3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 2, a countertop 1, a
diningtable 1, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a fridge 1,
a garbagecan 1, a microwave 1, a safe 1, a sinkbasin 1, a stoveburner 4, a stoveburner
3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: cool some pan and put it in stoveburner.

USER: Here is your sub goals to complete the task:
1. Find and obtain the required object
2. Cool the object at the cooling appliance
3. Bring the cooled object to the target receptacle
4. Place the object to complete the task

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am in the middle of the room, not holding anything. A pan is often on a
stoveburner -- I should check stoveburner 1.
Action: go to stoveburner 1

USER: Observation: On the stoveburner 1, you see nothing.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at stoveburner 1, not holding anything. Nothing here -- I should check
another stoveburner.
Action: go to stoveburner 3

USER: Observation: On the stoveburner 3, you see a pan 1.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the required object.
Currently, I am at stoveburner 3, not holding anything. I see the pan 1 here -- I
should take it.
Action: take pan 1 from stoveburner 3

USER: Observation: You pick up the pan 1 from the stoveburner 3.
```

```
ASSISTANT: Reflection: The sub-goal is complete -- I now hold the pan 1. My current
sub-goal is Cool the object at the cooling appliance. Currently, I am at stoveburner 3,
holding the pan -- I should go to the fridge to cool it.
Action: go to fridge 1

USER: Observation: The fridge 1 is closed.

ASSISTANT: Reflection: My current sub-goal is Cool the object at the cooling appliance.
Currently, I am at fridge 1, holding the pan -- I should cool the pan with the fridge
now.
Action: cool pan 1 with fridge 1

USER: Observation: You cool the pan 1 using the fridge 1.

ASSISTANT: Reflection: The sub-goal is complete -- the pan is now cooled. My current
sub-goal is Bring the cooled object to the target receptacle. Currently, I am at fridge
1, holding the cooled pan -- I should go to the stoveburner.
Action: go to stoveburner 1

USER: Observation: On the stoveburner 1, you see nothing.

ASSISTANT: Reflection: The sub-goal is complete -- I have arrived at stoveburner 1 with
the cooled pan. My current sub-goal is Place the object to complete the task.
Currently, I am at stoveburner 1, holding the cooled pan -- I should place it now.
Action: put pan 1 in/on stoveburner 1

USER: Observation: You put the pan 1 in/on the stoveburner 1.
```

ALFWorld — `pick_two_obj`

```
USER: You are in the middle of a room. Looking quickly around you, you see a armchair
2, a armchair 1, a bed 1, a countertop 1, a diningtable 1, a drawer 2, a drawer 1, a
dresser 1, a garbagecan 1, a laundryhamper 1, and a sidetable 1.
Your task is to: put two creditcard in dresser.

USER: Here is your sub goals to complete the task:
1. Find and obtain the first required object
2. Bring the first object to the target receptacle and place it
3. Find and obtain the second required object
4. Bring the second object to the target receptacle and place it

ASSISTANT: Reflection: My current sub-goal is Find and obtain the first required
object. Currently, I am in the middle of the room, not holding anything. A creditcard
might be in a drawer -- I should check drawer 1.
Action: go to drawer 1

USER: Observation: The drawer 1 is closed.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the first required
object. Currently, I am at drawer 1, not holding anything. The drawer is closed -- I
need to open it.
Action: open drawer 1

USER: Observation: You open the drawer 1. The drawer 1 is open. In it, you see a book
1, a cd 1, and a pen 1.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the first required
object. Currently, I am at drawer 1, not holding anything. No creditcard here -- I
should check the countertop.
Action: go to countertop 1

USER: Observation: On the countertop 1, you see a cellphone 2, a creditcard 4, a
creditcard 3, a creditcard 2, a mirror 1, a pencil 2, and a pencil 1.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the first required
object. Currently, I am at countertop 1, not holding anything. I see creditcards here
-- I should take the first one.
Action: take creditcard 2 from countertop 1

USER: Observation: You pick up the creditcard 2 from the countertop 1.

ASSISTANT: Reflection: The sub-goal is complete -- I now hold the first creditcard. My
current sub-goal is Bring the first object to the target receptacle and place it.
Currently, I am at countertop 1, holding creditcard 2 -- I should go to the dresser.
Action: go to dresser 1
```

```
USER: Observation: On the dresser 1, you see a mug 1, and a television 1.

ASSISTANT: Reflection: My current sub-goal is Bring the first object to the target
receptacle and place it. Currently, I am at dresser 1, holding creditcard 2 -- I should
place it now.
Action: put creditcard 2 in/on dresser 1

USER: Observation: You put the creditcard 2 in/on the dresser 1.

ASSISTANT: Reflection: The sub-goal is complete -- the first creditcard is in the
dresser. My current sub-goal is Find and obtain the second required object. Currently,
I am at dresser 1, not holding anything -- I should return to the countertop for the
second creditcard.
Action: go to countertop 1

USER: Observation: On the countertop 1, you see a cellphone 2, a creditcard 4, a
creditcard 3, a mirror 1, a pencil 2, and a pencil 1.

ASSISTANT: Reflection: My current sub-goal is Find and obtain the second required
object. Currently, I am at countertop 1, not holding anything. I see another creditcard
here -- I should take it.
Action: take creditcard 3 from countertop 1

USER: Observation: You pick up the creditcard 3 from the countertop 1.

ASSISTANT: Reflection: The sub-goal is complete -- I now hold the second creditcard. My
current sub-goal is Bring the second object to the target receptacle and place it.
Currently, I am at countertop 1, holding creditcard 3 -- I should go to the dresser.
Action: go to dresser 1

USER: Observation: On the dresser 1, you see a creditcard 2, a mug 1, and a television
1.

ASSISTANT: Reflection: My current sub-goal is Bring the second object to the target
receptacle and place it. Currently, I am at dresser 1, holding creditcard 3 -- I should
place it now to complete the task.
Action: put creditcard 3 in/on dresser 1

USER: Observation: You put the creditcard 3 in/on the dresser 1.
```

ALFWorld — `look_at_obj`

```
USER: You are in the middle of a room. Looking quickly around you, you see a bed 1, a
desk 1, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a safe 1, a shelf 5, a
shelf 4, a shelf 3, a shelf 2, a shelf 1, a sidetable 2, and a sidetable 1.
Your task is to: look at bowl under the desklamp.

USER: Here is your sub goals to complete the task:
1. Locate the required object
2. Obtain the required object
3. Locate the light source for examination
4. Examine the object under the light source to complete the task

ASSISTANT: Reflection: My current sub-goal is Locate the required object. Currently, I
am in the middle of the room, not holding anything. A bowl might be on a shelf -- I
should check shelf 1.
Action: go to shelf 1

USER: Observation: On the shelf 1, you see nothing.

ASSISTANT: Reflection: My current sub-goal is Locate the required object. Currently, I
am at shelf 1, not holding anything. Nothing here -- I should check shelf 2.
Action: go to shelf 2

USER: Observation: On the shelf 2, you see a bowl 1.

ASSISTANT: Reflection: The sub-goal is complete -- I found the bowl 1 on shelf 2. My
current sub-goal is Obtain the required object. Currently, I am at shelf 2, not holding
anything -- I should take the bowl.
Action: take bowl 1 from shelf 2

USER: Observation: You pick up the bowl 1 from the shelf 2.

ASSISTANT: Reflection: The sub-goal is complete -- I now hold the bowl 1. My current
sub-goal is Locate the light source for examination. Currently, I am at shelf 2,
holding the bowl -- a desklamp is often on a sidetable, so I should check sidetable 1.
Action: go to sidetable 1
```

```
USER: Observation: On the sidetable 1, you see a cd 1, a pen 1, and a pencil 1.

ASSISTANT: Reflection: My current sub-goal is Locate the light source for examination.
Currently, I am at sidetable 1, holding the bowl. No desklamp here -- I should check
sidetable 2.
Action: go to sidetable 2

USER: Observation: On the sidetable 2, you see a alarmclock 1, a desklamp 1, and a pen
2.

ASSISTANT: Reflection: The sub-goal is complete -- I found the desklamp 1 on sidetable
2. My current sub-goal is Examine the object under the light source to complete the
task. Currently, I am at sidetable 2, holding the bowl -- I should use the desklamp to
examine it.
Action: use desklamp 1

USER: Observation: You turn on the desklamp 1.
```

ScienceWorld

```
USER: Your task is to use chemistry to create green paint. When you are done, focus on
the green paint.

USER: Here is your sub goals to complete the task:
1. Teleport to the room where the needed materials are found
2. Gather the materials needed for green paint
3. Mix the materials to create green paint
4. Focus on the resulting green paint

ASSISTANT: Reflection: My current sub-goal is Teleport to the room where the needed
materials are found. Currently, I am in the starting room, holding nothing and focused
on nothing. Paint materials are likely to be found in the art studio -- I should
teleport there immediately to begin gathering.
Action: teleport to art studio

USER: Observation: You teleport to the art studio.

ASSISTANT: Reflection: The sub-goal is complete -- I have arrived at the art studio. My
current sub-goal is Gather the materials needed for green paint. Currently, I am in the
art studio, holding nothing and focused on nothing. I need to identify what materials
are available here -- I should look around first.
Action: look around

USER: Observation: This room is called the art studio. In it, you see:
  the agent
  a substance called air
  a large cupboard. The large cupboard door is open. In the large cupboard is: a
drawer, a wood cup (containing yellow paint), a wood cup (containing blue paint), a
wood cup (containing red paint).
  a table. On the table is: a glass cup (containing nothing).
  a wood cup (containing yellow paint)
  a wood cup (containing blue paint)
  a wood cup (containing red paint)
You also see:
  A door to the hallway (that is open)

ASSISTANT: Reflection: My current sub-goal is Gather the materials needed for green
paint. Currently, I am in the art studio, holding nothing and focused on nothing. Blue
and yellow make green, and both are available here -- I should pick up the blue paint
first to start gathering the required materials.
Action: pick up wood cup containing blue paint

USER: Observation: You move the wood cup to the inventory.

ASSISTANT: Reflection: My current sub-goal is Gather the materials needed for green
paint. Currently, I am in the art studio, holding the blue paint and focused on
nothing. I have the blue paint -- I now need to pick up the yellow paint to complete
gathering both required materials.
Action: pick up wood cup containing yellow paint

USER: Observation: You move the wood cup to the inventory.

ASSISTANT: Reflection: The sub-goal is complete -- I now hold both blue and yellow
paint. My current sub-goal is Mix the materials to create green paint. Currently, I am
in the art studio, holding blue and yellow paint and focused on nothing. I need to
start mixing -- I should pour the blue paint into the empty glass cup first.
```

```
Action: pour wood cup containing blue paint into glass cup

USER: Observation: You pour the contents of the wood cup into the glass cup.

ASSISTANT: Reflection: My current sub-goal is Mix the materials to create green paint.
Currently, I am in the art studio, holding the yellow paint and focused on nothing. The
blue paint is now in the glass cup -- I need to add the yellow paint now to have both
colors together for mixing.
Action: pour wood cup containing yellow paint into glass cup

USER: Observation: You pour the contents of the wood cup into the glass cup.

ASSISTANT: Reflection: My current sub-goal is Mix the materials to create green paint.
Currently, I am in the art studio, holding nothing and focused on nothing. Both blue
and yellow paint are now in the glass cup -- I need to mix them now to produce the
green paint.
Action: mix glass cup

USER: Observation: Blue and yellow paint mix to produce green paint.

ASSISTANT: Reflection: The sub-goal is complete -- the green paint has been created. My
current sub-goal is Focus on the resulting green paint. Currently, I am in the art
studio, holding nothing and focused on nothing. The green paint is ready -- I need to
focus on it now to complete the task.
Action: focus on green paint

USER: Observation: You focus on the green paint.
```

Jericho

```
USER: Goal: You are the warrior Link that needs to save the princess from the castle.
Observation: You are at the path leading to the castle. The castle is to your north.
There is a barrel in front of you.

USER: Here is your sub goals to complete the task:
1. Find and obtain any useful item available in the starting area
2. Navigate into the castle
3. Reach the princess's location within the castle
4. Wake the princess to complete the rescue

ASSISTANT: Reflection: My current sub-goal is Find and obtain any useful item available
in the starting area. Currently, I am at the path leading to the castle, holding
nothing. There is a barrel in front of me -- I should examine it to find any useful
item before heading into the castle.
Action: examine barrel

USER: Observation: There is a gleaming elvish sword.

ASSISTANT: Reflection: My current sub-goal is Find and obtain any useful item available
in the starting area. Currently, I am at the path leading to the castle, holding
nothing. I found a gleaming elvish sword in the barrel -- I need to take it now to be
equipped before entering the castle.
Action: take the sword

USER: Observation: The sword is taken

ASSISTANT: Reflection: The sub-goal is complete -- I now hold the elvish sword. My
current sub-goal is Navigate into the castle. Currently, I am at the path, holding the
elvish sword. The castle entrance is to the north -- I should move there now.
Action: north

USER: Observation: Main hall
The castle is dimly lit, with lava flowing in many places. There is a staircase leading
up to princess bedroom, a door to your north leading to the kitchen hall, and the door
you just entered on your south

ASSISTANT: Reflection: The sub-goal is complete -- I have entered the castle. My
current sub-goal is Reach the princess's location within the castle. Currently, I am in
the castle main hall, holding the elvish sword. A staircase leads up to the princess's
bedroom -- I should check the valid actions to confirm the correct path.
Action: check valid actions

USER: Observation: south, north, up, look, examine staircase

ASSISTANT: Reflection: My current sub-goal is Reach the princess's location within the
castle. Currently, I am in the castle main hall, holding the elvish sword. The valid
actions confirm 'up' leads to the princess's bedroom -- I should go up now to reach her.
```

```
Action: up

USER: Observation: Princess's bedroom. Princess is sleeping in bed.

ASSISTANT: Reflection: The sub-goal is complete -- I have reached the princess's
bedroom. My current sub-goal is Wake the princess to complete the rescue. Currently, I
am in the princess's bedroom, holding the elvish sword, and the princess is sleeping --
I should check the valid actions to find how to wake her.
Action: check valid actions

USER: Observation: wake up the princess, take out sword, down

ASSISTANT: Reflection: My current sub-goal is Wake the princess to complete the rescue.
Currently, I am in the princess's bedroom, holding the elvish sword. The action "wake
up the princess" is available and directly achieves this final sub-goal -- I should use
it now.
Action: wake up the princess

USER: Observation: The princess wake up from the coma. Thank you my knight, she says.
The task is finished.
```

P ALFWORLD PLANNER PROMPTS AND IN-CONTEXT EXAMPLE

The SGG-ReflAct planner for ALFWorld produces phase-level sub-goals. We show a representative in-context example (`pick_and_place`). The other five task types follow the same format.

One-shot planner prompt

```
Generate exactly 4 high-level sub-goals for a task in a text-based household simulation
environment (ALFWorld). Each sub-goal is an outcome or milestone -- WHAT to achieve,
not HOW -- and may require multiple actions; do NOT specify exact action commands.
Rules: (1) Order sub-goals by dependency: the object must be found and obtained before
it is transformed (cleaned, heated, or cooled) or placed. (2) If the task requires a
transformation (clean/heat/cool), the transformation sub-goal MUST come before the
placement sub-goal. (3) Use generic descriptions for variable elements (e.g., 'the
required object', 'the target receptacle', 'the appropriate appliance'); only name
objects or locations explicitly stated in the task. (4) The final sub-goal must place
or use the object to directly achieve the task goal. (5) Write each sub-goal as a short
imperative phrase starting with a verb, with no trailing period. When an object is
named in the task, weave it inline (e.g., 'Find the peppershaker'), not as a trailing
clause. Format: 'Sub goals:\n1. ...\n2. ...\n...'
```

One-shot planner in-context example

```
USER: You are in the middle of a room. Looking quickly around you, you see a cabinet 4,
a cabinet 3, a cabinet 2, a cabinet 1, a countertop 1, a garbagecan 1, a
handtowelholder 2, a handtowelholder 1, a sinkbasin 2, a sinkbasin 1, a toilet 1, a
toiletpaperhanger 1, and a towelholder 1.
Your task is to: put some spraybottle on toilet.

ASSISTANT: Sub goals:
1. Locate the required object
2. Obtain the required object
3. Bring the object to the target receptacle
4. Place the object to complete the task
```

Beam Search Planner prompt (per-step candidate generation)

```
Generate the next high-level sub-goal to extend a partial plan for a task in a
text-based household simulation environment (ALFWorld). Each sub-goal is an outcome or
milestone -- WHAT to achieve, not HOW -- and may require multiple actions; do NOT
specify exact action commands. Rules: (1) Order by dependency: the object must be found
and obtained before it is transformed (cleaned, heated, or cooled) or placed; do not
depend on something a later sub-goal will provide. (2) If the task requires a
transformation (clean/heat/cool), the transformation sub-goal MUST come before the
placement sub-goal. (3) Use generic descriptions for variable elements (e.g., 'the
required object', 'the target receptacle', 'the appropriate appliance'); only name
objects or locations explicitly stated in the task. (4) If this is the 4th (last)
sub-goal, it must place or use the object to directly achieve the task goal. (5) Write
the sub-goal as a short imperative phrase starting with a verb, with no trailing
period. When an object is named in the task, weave it inline (e.g., 'Find the
```

```
peppershaker'), not as a trailing clause. The total plan must consist of exactly 4
sub-goals. Output ONLY the sub-goal description on a single line. No explanation, no
prefix.
```

Here {expand_prompt} is the Beam Search Planner prompt above.

Self-evaluation prompt

```
Evaluate whether a partial plan is appropriate given the following planning constraints:

{expand_prompt}

Given the Task and its Partial plan, assess how well the plan satisfies ALL of the
following:
  1. Each sub-goal describes WHAT to achieve (an outcome or milestone), not HOW -- it
may require multiple actions.
  2. Each sub-goal logically follows from the previous ones without skipping
prerequisites.
  3. An object must be located and obtained before any sub-goal that transforms
(cleans, heats, cools) or places it.
  4. Sub-goals use generic descriptions for variable elements (e.g., 'the required
object', 'the target receptacle', 'the available cleaning station').
  5. If it is the 4th (last) sub-goal, completing it achieves the final goal stated in
the Task.
Rate how well the Partial plan satisfies the criteria above on a continuous scale from
0.0 to 1.0 (1.0 = fully correct and well-ordered, 0.0 = clearly wrong). Use the full
range: a plan that satisfies most but not all criteria should receive an intermediate
value.
Answer with a single decimal number only (e.g., 0.82). No explanation.
```

Beam Search Planner in-context example

```
USER: Task: You are in the middle of a room. Looking quickly around you, you see a
cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a countertop 1, a garbagecan 1, a
handtowelholder 2, a handtowelholder 1, a sinkbasin 2, a sinkbasin 1, a toilet 1, a
toiletpaperhanger 1, and a towelholder 1.
Your task is to: put some spraybottle on toilet.
Current plan so far: []
Generate the next sub-goal (step 1 of 4):

ASSISTANT: Locate the required object

USER: Task: You are in the middle of a room. Looking quickly around you, you see a
cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a countertop 1, a garbagecan 1, a
handtowelholder 2, a handtowelholder 1, a sinkbasin 2, a sinkbasin 1, a toilet 1, a
toiletpaperhanger 1, and a towelholder 1.
Your task is to: put some spraybottle on toilet.
Current plan so far: ['Locate the required object']
Generate the next sub-goal (step 2 of 4):

ASSISTANT: Obtain the required object

USER: Task: You are in the middle of a room. Looking quickly around you, you see a
cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a countertop 1, a garbagecan 1, a
handtowelholder 2, a handtowelholder 1, a sinkbasin 2, a sinkbasin 1, a toilet 1, a
toiletpaperhanger 1, and a towelholder 1.
Your task is to: put some spraybottle on toilet.
Current plan so far: ['Locate the required object', 'Obtain the required object']
Generate the next sub-goal (step 3 of 4):

ASSISTANT: Bring the object to the target receptacle

USER: Task: You are in the middle of a room. Looking quickly around you, you see a
cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a countertop 1, a garbagecan 1, a
handtowelholder 2, a handtowelholder 1, a sinkbasin 2, a sinkbasin 1, a toilet 1, a
toiletpaperhanger 1, and a towelholder 1.
Your task is to: put some spraybottle on toilet.
Current plan so far: ['Locate the required object', 'Obtain the required object',
'Bring the object to the target receptacle']
Generate the next sub-goal (step 4 of 4):

ASSISTANT: Place the object to complete the task
```

Q SCIENCEWORLD PLANNER PROMPTS AND IN-CONTEXT EXAMPLE

One-shot planner prompt

```
Generate exactly 4 high-level sub-goals to help an agent complete a task in a
text-based science simulation environment (ScienceWorld). Each sub-goal describes WHAT
to achieve, not HOW -- it may require multiple actions. Rules: (1) Follow ordering
instructions in the task description exactly. CRITICAL: if the task says 'First, focus
on the substance' (or 'first focus on X'), the 'focus on' sub-goal MUST come
immediately after finding/obtaining the substance (around sub-goal 2) and BEFORE any
sub-goal that changes, heats, cools, boils, or otherwise transforms it. NEVER place
such a 'focus on' sub-goal last. (2) Any 'focus on' sub-goal must appear after the
sub-goal that finds or obtains the object. (3) Use generic descriptions for variable
elements (e.g., 'the room where X can be found', 'the container holding X'). Only name
objects or substances explicitly stated in the task. Format: 'Sub goals:\n1. ...\n2.
...\n...'
```

One-shot planner in-context example

```
USER: Your task is to use chemistry to create green paint. When you are done, focus on
the green paint.

ASSISTANT: Sub goals:
1. Teleport to the room where the needed materials are found
2. Gather the materials needed for green paint
3. Mix the materials to create green paint
4. Focus on the resulting green paint
```

Beam Search Planner prompt (per-step candidate generation)

```
Generate the next candidate sub-goal to extend a partial plan for a task in a
text-based science simulation environment (ScienceWorld). Each sub-goal describes WHAT
to achieve, not HOW -- it may require multiple actions. Rules: (1) The sub-goal content
MUST be derived from the task description, not from any example. If the task
description explicitly states what to do next (e.g., 'First, focus on the substance',
'First, focus on the thing'), that action MUST become the next sub-goal -- expressed
without the 'First,' prefix. (2) A 'focus on' sub-goal MUST appear after finding the
object AND before moving or placing it. CRITICAL: Even if the task description says
'First, focus on X', when the current plan is empty, the first sub-goal MUST be finding
or locating X -- never focusing. (3) Use generic descriptions for variable elements
(e.g., 'the room where X can be found', 'the container holding X'). Only name
substances or objects explicitly stated in the task. The total plan must consist of
exactly 4 sub-goals. Output ONLY the sub-goal description on a single line. No
explanation, no prefix.
```

Here {expand_prompt} is the Beam Search Planner prompt above.

Self-evaluation prompt

```
Evaluate whether a partial plan is appropriate given the following planning constraints:

{expand_prompt}

Given the Task and its Partial plan, assess how well the plan satisfies ALL of the
following:
  1. Each sub-goal describes WHAT to achieve (not HOW) and follows any ordering
instructions in the task description.
  2. Each sub-goal logically follows from the previous ones without skipping
prerequisites.
  3. Any 'focus on' sub-goal appears after the sub-goal that finds or obtains the
object.
  4. If it is the 4th (last) sub-goal, completing it achieves the final goal stated in
the Task.
  5. Sub-goals use generic descriptions for variable elements (e.g., 'the room where X
can be found').
Rate how well the Partial plan satisfies the criteria above on a continuous scale from
0.0 to 1.0 (1.0 = fully correct and well-ordered, 0.0 = clearly wrong). Use the full
range: a plan that satisfies most but not all criteria should receive an intermediate
value.
Answer with a single decimal number only (e.g., 0.82). No explanation.
```

Beam Search Planner in-context example

```
USER: Task: Your task is to use chemistry to create green paint. When you are done,
focus on the green paint.
Current plan so far: []
Generate the next sub-goal for THIS task only.

ASSISTANT: Teleport to the room where the needed materials are found

USER: Task: Your task is to use chemistry to create green paint. When you are done,
focus on the green paint.
Current plan so far: ['Teleport to the room where the needed materials are found']
Generate the next sub-goal for THIS task only.

ASSISTANT: Gather the materials needed for green paint

USER: Task: Your task is to use chemistry to create green paint. When you are done,
focus on the green paint.
Current plan so far: ['Teleport to the room where the needed materials are found',
'Gather the materials needed for green paint']
Generate the next sub-goal for THIS task only.

ASSISTANT: Mix the materials to create green paint

USER: Task: Your task is to use chemistry to create green paint. When you are done,
focus on the green paint.
Current plan so far: ['Teleport to the room where the needed materials are found',
'Gather the materials needed for green paint', 'Mix the materials to create green
paint']
Generate the next sub-goal for THIS task only.

ASSISTANT: Focus on the resulting green paint
```

State-change planner prompts (+proc).

The *+proc* condition of Table 14 inserts the rule below into both planner prompts above as rule (3), moving each prompt's generic-description rule to (4). The rule is identical in both.

Added rule for the *+proc* condition

```
(3) ONLY for tasks explicitly involving boiling, melting, freezing, or changing the
state of matter of a substance -- NOT for chemistry, mixing, or paint creation tasks --
NEVER use 'take actions to change state of matter' as a sub-goal -- it is too vague.
The correct sub-goal order is: (a) find and obtain the substance, (b) focus on the
substance, (c) reach or obtain the heat or cold source, (d) apply heat or cold to cause
the state change. The 'focus on the substance' sub-goal MUST come at step (b) -- never
as the last sub-goal.
```

For the Beam Search Planner that is the only change. The single-path variant additionally takes rules (1) and (2) from the Beam Search Planner prompt, dropping the two phrases that apply only during incremental expansion: `the next` in rule (1), and `when the current plan is empty` in rule (2).

R JERICHO PLANNER PROMPTS AND IN-CONTEXT EXAMPLE

One-shot planner prompt

```
Generate exactly 4 high-level sub-goals for a task in a text-based interactive fiction
adventure game (Jericho). Each sub-goal is an outcome or milestone -- WHAT to achieve,
not HOW -- and may require several actions including exploration and navigation. Rules:
(1) Order sub-goals by dependency: obtain an item before using or giving it; if the
goal mentions darkness, obtain a light source before entering dark areas. (2) Use the
names and hints stated in the goal text (a specific room, where an item is hidden, a
required tool or method); for anything not stated, use a generic description and never
invent specific items, locations, or methods. (3) The final sub-goal must directly
achieve the objective (obtain X, give X to Y, or reach the exit); add no steps after
it. Format: 'Sub goals:\n1. ...\n2. ...\n...'
```

One-shot planner in-context example

```
USER: Goal: You are the warrior Link that needs to save the princess from the castle.

Starting room:
You are at the path leading to the castle. The castle is to your north. There is a
barrel in front of you.

ASSISTANT: Sub goals:
1. Find and obtain any useful item available in the starting area
2. Navigate into the castle
3. Reach the princess's location within the castle
4. Wake the princess to complete the rescue
```

Beam Search Planner prompt (per-step candidate generation)

```
Generate the next high-level sub-goal to extend a partial plan for a task in a
text-based interactive fiction adventure game (Jericho). Each sub-goal is an outcome or
milestone -- WHAT to achieve, not HOW -- and may require several actions including
exploration and navigation. Rules: (1) Derive the sub-goal from the task goal and
starting observation, not from the example. (2) Order by dependency: obtain an item or
light source before using it or entering dark areas; do not depend on something a later
sub-goal will provide. (3) Use the names and hints stated in the goal text; for
anything not stated, use a generic description and never invent specific items,
locations, or methods. (4) If this is the 4th (last) sub-goal, it must directly achieve
the objective (obtain X, give X to Y, or reach the exit). The total plan must consist
of exactly 4 sub-goals. Output ONLY the sub-goal description on a single line. No
explanation, no prefix.
```

Here {expand_prompt} is the Beam Search Planner prompt above.

Self-evaluation prompt

```
Evaluate whether a partial plan is appropriate given the following planning constraints:

{expand_prompt}

Given the Task (goal + starting room) and its Partial plan, assess how well the plan
satisfies ALL of the following:
  1. Each sub-goal describes WHAT to achieve (an outcome or milestone), not HOW.
  2. No sub-goal contains a bare navigation direction (north, south, east, west, up,
down) as its main action -- destinations must be described by what they contain or
their purpose instead.
  3. Each sub-goal logically follows from the previous ones without skipping
prerequisites.
  4. An item must be found or obtained before any sub-goal that uses or gives it.
  5. If the goal mentions darkness or a required light source, the sub-goal to obtain
the light source appears before any sub-goal that enters a dark area.
  6. Sub-goals use generic descriptions for items or locations not explicitly stated in
the goal or starting observation (e.g., 'the required tool', 'the target location')
rather than hallucinating specific names.
  7. If it is the 4th (last) sub-goal, completing it directly achieves the final
objective stated in the Task (e.g., 'deliver X to Y', 'open the gate'), not merely
gather information.
Rate how well the Partial plan satisfies the criteria above on a continuous scale from
0.0 to 1.0 (1.0 = fully correct and well-ordered, 0.0 = clearly wrong). Use the full
range: a plan that satisfies most but not all criteria should receive an intermediate
value.
Answer with a single decimal number only (e.g., 0.82). No explanation.
```

Beam Search Planner in-context example

```
USER: Task: Goal: You are the warrior Link that needs to save the princess from the
castle.

Starting room:
You are at the path leading to the castle. The castle is to your north. There is a
barrel in front of you.
Current plan so far: []
Generate the next sub-goal (step 1 of 4):

ASSISTANT: Find and obtain any useful item available in the starting area
```

```
USER: Task: Goal: You are the warrior Link that needs to save the princess from the
castle.

Starting room:
You are at the path leading to the castle. The castle is to your north. There is a
barrel in front of you.
Current plan so far: ['Find and obtain any useful item available in the starting area']
Generate the next sub-goal (step 2 of 4):

ASSISTANT: Navigate into the castle

USER: Task: Goal: You are the warrior Link that needs to save the princess from the
castle.

Starting room:
You are at the path leading to the castle. The castle is to your north. There is a
barrel in front of you.
Current plan so far: ['Find and obtain any useful item available in the starting area',
'Navigate into the castle']
Generate the next sub-goal (step 3 of 4):

ASSISTANT: Reach the princess's location within the castle

USER: Task: Goal: You are the warrior Link that needs to save the princess from the
castle.

Starting room:
You are at the path leading to the castle. The castle is to your north. There is a
barrel in front of you.
Current plan so far: ['Find and obtain any useful item available in the starting area',
'Navigate into the castle', "Reach the princess's location within the castle"]
Generate the next sub-goal (step 4 of 4):

ASSISTANT: Wake the princess to complete the rescue
```

S ABLATION, GROUNDING, AND PROCEDURAL-PROMPTING FIGURES/TABLES

Table 13 accompanies the reflection-anchoring ablation in Section 5.1; Figure 2 and Figure 3 accompany the grounding analysis in the same section; and Table 14 accompanies the procedural-prompting analysis in Section 5.2.

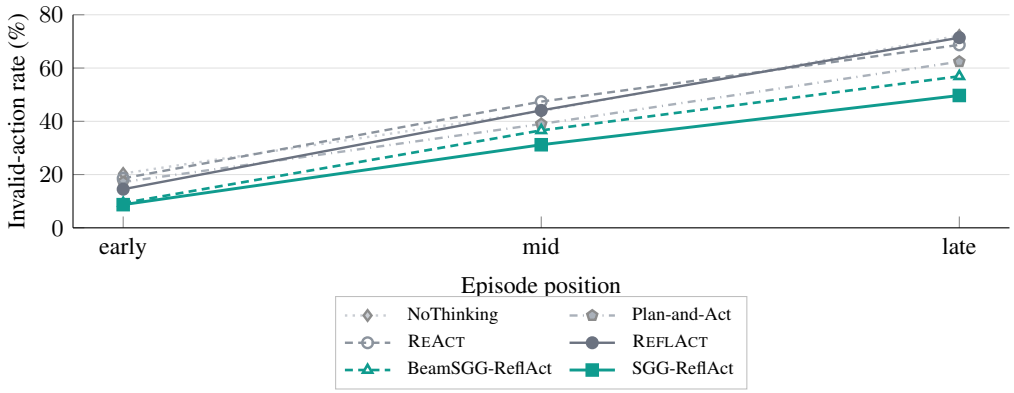

Figure 2: Invalid-action rate by episode third on ScienceWorld (Llama-3.1-8B-Instruct) for all six agents; see Appendix L for GPT-4o and GPT-4o-mini results.

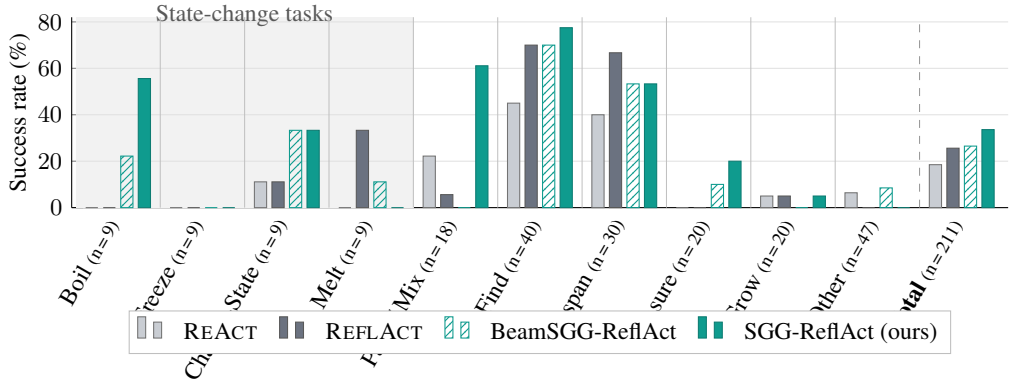

Figure 3: ScienceWorld success rates by task bucket across four reasoning backbones (Llama-3.1-8B-Instruct). The ten buckets regroup ScienceWorld's nine categories for readability: the state-change category is shown as its four task types (shaded), the chemistry category as its paint-mixing episodes, and the rest are pooled into "Other" (electrical/power, conductivity testing, non-paint chemistry mixing, and life-stage identification). n is the number of episodes per bucket, and "Total" is the rate over 211 episodes, matching Table 1. Table 7 gives the per-row values behind these bars, and Table 6 the same episodes by the nine categories of the ScienceWorld split.

Table 13: Ablation isolating the reflection-anchoring instruction on ScienceWorld (Llama-3.1-8B-Instruct). *Plan* indicates whether a sub-goal plan is present in the context. Values are percentages.

Agent	Plan	Reflects on	SR / AR
REFLACT	×	end-goal	25.6 / 41.5
REFLACT + Plan	√	end-goal	27.5 / 44.8
SGG-ReflAct	√	active sub-goal	**33.6 / 48.7**

Table 14: ScienceWorld success rates (%) for both planners, comparing the base ScienceWorld planner prompt with its state-change variant (+*proc*; Appendix Q). *Overall* includes 211 episodes; *State-change* includes the 36 targeted episodes (Boil/Freeze/Melt/Change-State). SGG/Beam denote SGG-ReflAct/BeamSGG-ReflAct. *base* matches Table 1. Bold indicates the best performance in each pair.

		Overall SR		State-change SR	
Model	Planner	base	+proc	base	+proc
GPT-4o	SGG	64.0	**66.4**	55.6	**72.2**
	Beam	63.5	**65.4**	66.7	**75.0**
GPT-4o-mini	SGG	45.0	**49.8**	52.8	**55.6**
	Beam	43.6	**47.4**	25.0	**55.6**
Llama-3.1 -8B	SGG	**33.6**	32.2	**22.2**	16.7
	Beam	26.5	**31.3**	16.7	**27.8**