SYMBOLIC COMPOSITIONAL PLANNING FROM DEMONSTRATED EXPERIENCE FOR EMBODIED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language models (VLMs) exhibit strong in-context learning capabilities, yet embodied agents built on them struggle with complex tasks given only a few demonstrations, often omitting critical steps, violating action preconditions, or organizing the overall workflow incorrectly. These failures stem from an incomplete understanding of demonstrated action dependencies and a failure to distinguish reusable local procedures from task-specific workflow structure. To address this problem, we propose an embodied planning method that combines trajectory composition with symbolic analysis, reusing demonstrated local procedures while reorganizing them into workflows for new tasks. Demonstration fragments preserve local action structure, while symbolic analysis makes their preconditions and effects explicit to guide composition and connections between fragments. During execution, the agent maintains scene memory and uses execution feedback to revise the remaining plan. Experiments on EB-ALFRED and EB-Habitat show success-rate improvements of 21% and 22 %, respectively, over standard few-shot planning with the same backbone.

1 INTRODUCTION

Through in-context learning, Vision-language models (VLMs) enable embodied agents to convert natural-language instructions into executable action plans from only a small number of successful demonstrations (Huang et al., 2022; Song et al., 2023). However, when confronted with complex tasks, agents often omit critical steps, violate action preconditions, or misorder the overall workflow. As action sequences become longer, early-stage errors can invalidate the conditions required for subsequent operations, ultimately preventing successful task completion (Wei et al., 2026).

Despite recent progress, existing approaches have not fully resolved how procedural experience should transfer to novel complex tasks. Example retrieval provides relevant trajectories as in-context references (Song et al., 2023; Zhao et al., 2024), while experience abstraction extracts reusable workflows or augments demonstrations with cognitive annotations, such as causal relations and state transitions (Wang et al., 2025; Sarch et al., 2024). ELITE further combines experience abstraction with intent-aware retrieval, distilling reusable strategies from execution trajectories and retrieving them according to the current planning intent (Wei et al., 2026). However, retrieved or abstracted experience often remains tightly coupled to the task-specific execution order from which it was derived. Simply reusing an entire trajectory can preserve an unsuitable global workflow, whereas freely recombining actions can break the local dependencies required for successful execution.

This challenge arises because effective transfer requires reasoning at two distinct yet interdependent levels: local procedural dependencies and global workflow organization. Successful demonstrations reveal an execution order, but typically leave implicit the preconditions that each action requires and the postconditions it establishes for subsequent actions. To adapt a demonstrated procedure to new objects or contexts, an agent must infer and preserve these essential dependencies rather than merely imitate surface-level action sequences. Meanwhile, local procedures are embedded within the source task's global structure. A target task may involve similar operations but require a different ordering, number of repetitions, or hierarchical nesting. Without explicitly separating local dependency preservation from global workflow adaptation, an agent may either disrupt critical

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.

action preconditions or retain an organizational structure that no longer serves the current goal. The problem becomes even more acute for long-horizon planning, where action-induced state changes must be tracked continuously and local procedure construction must remain coordinated with the evolving global workflow.

To address these challenges, we propose COPE, a symbolic **CO**mpositional **P**lanning framework that transfers procedural experience from limited demonstrations to new **E**mbodied tasks. Prior work on skill graphs and short-trajectory composition has demonstrated how local behaviors can be assembled into long-horizon behavior (Bagaria et al., 2021; Luo et al., 2025). Building on this principle, our method performs in-context learning directly from raw demonstrations by extracting contiguous action fragments that represent reusable local procedures. To enable reliable adaptation, we use symbolic analysis to derive the preconditions and effects of each adapted fragment from action rules, exposing the state dependencies that are otherwise implicit in demonstrations. The model then uses these explicit constraints to determine fragment ordering and connections, revising the composition whenever required conditions are not satisfied. In this way, the proposed method couples the reuse of local procedural experience with the goal-directed organization of complete workflows, enabling agents to adapt demonstrated behaviors to new complex tasks without sacrificing execution feasibility.

During execution, the agent follows the composed plan while updating its scene memory after each interaction. This scene memory grounds planned object references in concrete environmental instances and maintains the current world state needed to determine whether subsequent actions remain applicable. When an action outcome deviates from the plan, the agent leverages scene memory and execution feedback to revise the remaining action sequence.

Our main contributions are as follows:

- We propose an embodied planning framework that transfers action fragments from limited demonstrations to new complex tasks. The framework integrates trajectory composition with closed-loop execution, coupling complete-plan generation with feedback-guided object grounding and local plan adaptation.
- We introduce a symbolic feedback mechanism that derives and exploits fragment preconditions and effects to guide fragment selection and composition. By verifying goal achievement and tracking state dependencies across fragments, it enables the agent to revise fragment choices, execution order, and connecting actions when necessary.
- We evaluated our method with VLM backbones of varying scales on EB-ALFRED and EB-Habitat. Compared with standard few-shot planning with the same backbone, our method improves success rates by 21% and 22 % on the two benchmarks, respectively.

2 Related Work

2.1 Embodied Task Planning

Embodied task planning translates natural-language goals into actions in an environment. Foundation models provide task understanding and commonsense knowledge for generating high-level plans and invoking available actions. Huang et al. (2022) map language-model outputs to executable operations through semantic matching. SayCan combines language-model scores with skill value functions to account for both goal relevance and execution feasibility (Ahn et al., 2022). Interactive approaches incorporate environmental feedback into planning. ReAct alternates reasoning and action, while Inner Monologue supplies subsequent decisions with success signals and scene descriptions (Yao et al., 2023; Huang et al., 2023).

Long-horizon tasks also require persistent use of environmental state and execution history. Mimir maintains separate world and task memories and binds active goals to retrieved environmental evidence before acting. GiG organizes scenes and execution history as graphs, using memory retrieval and lookahead to support action generation (Xu et al., 2026a; Li et al., 2026). These methods strengthen the relationship between task goals, environmental information, and action decisions. We focus on the use of experience in planning: constructing the complete action sequence required by a new task from limited successful demonstrations.

2.2 In-Context Learning for Embodied Planning

In-context learning guides a model's planning through a small number of task demonstrations supplied at inference time, without updating its parameters. LLM-Planner combines demonstrations with environmental information to generate and update plans. ProgPrompt represents demonstrations as programs, using action calls and control structures to express task execution (Song et al., 2023; Singh et al., 2023). These methods allow models to draw on the relationship between task instructions and action procedures presented in context.

Related work on language-model agents explores which experiences to include in the model's context and how to represent them. Voyager stores and retrieves reusable skill programs, while Agent Workflow Memory (AWM) induces workflows from execution trajectories and composes existing workflows into more complex ones (Wang et al., 2024; 2025). ICAL revises demonstrations and annotates causal relations, state changes, and subgoals, enriching the procedural experience available for in-context learning (Sarch et al., 2024). ELITE retrieves strategies distilled from execution trajectories according to the current planning intent. HP-KG organizes tasks, steps, and actions in a hierarchical knowledge graph to support experience retrieval and reuse (Wei et al., 2026; Zhou et al., 2025).

A related line of work transforms execution experience into programs or action models. Demo2Code synthesizes control programs by summarizing demonstrations, while BLADE and PDDLLM derive action representations that support compositional planning (Wang et al., 2023; Liu et al., 2025; Huang et al., 2026). These approaches encode execution structure and action constraints in reusable programs or models, providing another way to organize new plans from prior experience.

We study composition over raw demonstrations within in-context learning. With model parameters and the demonstration set held fixed, the model selects and adapts contiguous action fragments from different demonstrations for the current task. Symbolic analysis uses action rules to make their preconditions and effects explicit. The local action structure supplied by demonstrations and the symbolically computed execution constraints jointly guide workflow construction, making fragment selection and composition explicit parts of experience reuse.

3 Problem Formulation

We consider embodied task planning in a partially observable environment. Given a natural-language instruction g, an agent interacts with the environment by selecting actions a_t from an action space $\mathcal{A}$, which includes navigation and object-manipulation operations. At each time step t, the agent receives a visual observation o_t and selects an action based on the available observations and interaction history, with the objective of completing the instructed task.

The agent has access to a limited set of successful demonstrations $\mathcal{D} = \{(g_i, \tau_i)\}_{i=1}^{N}$, where g_i denotes a demonstration instruction and τ_i is the corresponding high-level action trajectory. The action-rule set $\mathcal{R}$ provides domain-level preconditions and effects for validating candidate action sequences, but does not specify task-dependent procedural structure or a complete symbolic world state. Given the task instruction g, initial observation o_0, action-rule set $\mathcal{R}$, and demonstrations $\mathcal{D}$, the model first generates a complete initial plan $\pi_0 = \pi^{\text{init}} = (\bar{a}_1, \ldots, \bar{a}_K)$, where $\bar{a}_k \in \mathcal{A}$ and K denotes the plan length. This plan defines the intended global workflow but is not treated as a fixed sequence to be executed. Instead, its object references may be grounded again and its remaining actions are adjusted through interaction with the environment.

During execution, the agent maintains an interaction history H_t, comprising previous observations, executed actions, and environmental feedback, as well as a pending plan π_t containing the actions yet to be executed. Initially, $H_0 = (o_0)$ and $\pi_0 = \pi^{\text{init}}$. At each time step, a decision policy μ_θ selects the next action conditioned on the task instruction, interaction history, demonstrations, and current pending plan:

$$a_t \sim \mu_\theta(\cdot \mid g, H_t, \mathcal{D}, \pi_t)\,. \tag{1}$$

After executing a_t, the agent incorporates the resulting observation and environmental feedback into H_{t+1} and updates the execution progress in π_{t+1}. If the remaining plan remains applicable, execution proceeds without revision. Otherwise, the agent revises only its unexecuted portion while

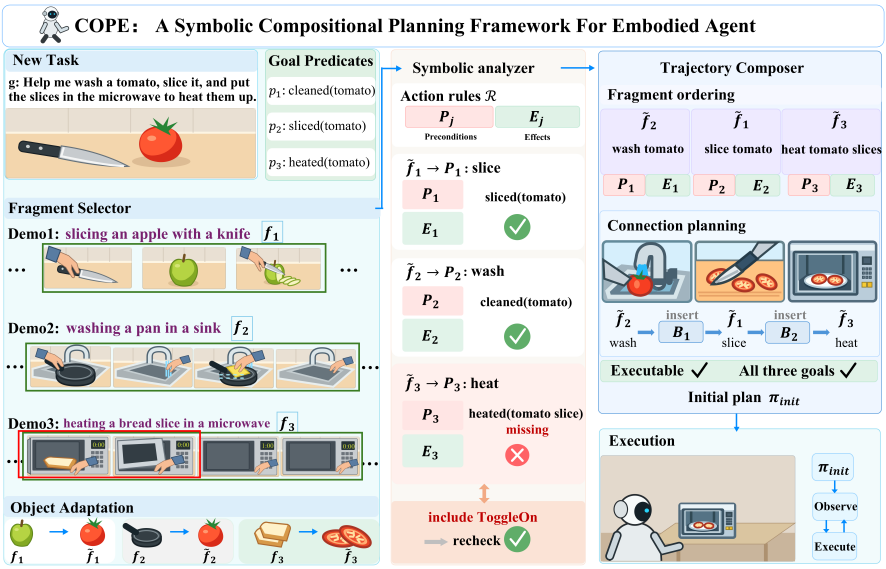

Figure 1: **Overview of trajectory composition and closed-loop execution.** Goal predicates guide fragment selection and object adaptation. Symbolic preconditions, effects, and validation feedback guide fragment revision and composition.

preserving completed actions and their observed outcomes. Execution terminates when the task goal is achieved or the interaction budget $T_{\max}$ is exhausted.

4 Our COPE Approach

We propose COPE, an embodied planning framework that enables agents to transfer procedural experience from a limited set of demonstrations to novel complex tasks. As illustrated in Figure 1, COPE comprises two complementary components: trajectory composition and closed-loop adjustment. Trajectory composition leverages symbolic analysis to select and assemble action fragments from demonstrations into a complete, executable plan tailored to the current task. During execution, closed-loop adjustment incorporates visual observations and environmental feedback to refine object grounding and revise the remaining plan when deviations occur.

4.1 Trajectory Composition

Our trajectory composition module transfers local procedures from successful demonstrations according to the goal conditions of the target task. It consists of two stages. First, a fragment selector retrieves and adapts demonstrated action fragments, while a symbolic analyzer verifies whether their effects support the intended goals. Second, a trajectory composer orders the verified fragments and generates connecting actions to form an executable, goal-complete plan.

Fragment selection. Given the task instruction g, a VLM-based fragment selector decomposes the task objective into a set of goal predicates $G = \{p_1, \ldots, p_m\}$ that must jointly hold upon task completion. For each predicate p_j, the selector identifies a contiguous fragment from the successful demonstrations $\mathcal{D}$ whose operation pattern is relevant to achieving that predicate. It then adapts the objects in the retrieved fragment to the target task. If the fragment is retrieved from demonstration i_j, its source and adapted sequences are given by

$$f_j = \tau_{i_j}[\ell_j : r_j], \qquad \widetilde{f}_j = \phi_j(f_j), \tag{2}$$

where $[\ell_j, r_j]$ denotes an inclusive interval in the source trajectory and ϕ_j is an object mapping from the demonstrated task to the target task. The adaptation preserves the action types and internal action order of the source fragment while maintaining consistent references to shared target objects across fragments.

Symbolic fragment analysis. To determine whether an adapted fragment can reliably support its assigned goal, the symbolic analyzer derives its preconditions and effects from an action-rule set $\mathcal{R}$:

$$(P_j, E_j) = \mathrm{Analyze}_{\mathcal{R}}(\widetilde{f}_j), \tag{3}$$

where P_j contains the conditions that must hold before the fragment is executed, and E_j characterizes the state changes induced by its execution. For any symbolic state $\widehat{s}$ satisfying the fragment preconditions, a valid fragment for p_j should satisfy

$$\widehat{s} \models P_j \implies \mathrm{Exec}_{\mathcal{R}}(\widehat{s}, \widetilde{f}_j) \models p_j. \tag{4}$$

Here, $\mathrm{Exec}_{\mathcal{R}}$ returns the symbolic state reached by executing a sequence under $\mathcal{R}$. Although $\mathcal{R}$ specifies the preconditions and effects of individual actions, it is not intended to serve as a complete generative task model. In particular, it does not encode task-specific procedural preferences, semantic correspondences between natural-language goals and environment objects, or the high-level organization of actions into meaningful procedures. Moreover, under partial observability, the agent generally does not have access to a complete symbolic initial state from which a classical planner could reliably synthesize an entire plan. Direct search over primitive actions would therefore require resolving substantial semantic and combinatorial ambiguity. Demonstrations and symbolic rules instead provide complementary forms of knowledge. Demonstrated fragments supply procedural priors that constrain planning to locally coherent behaviors observed in successful executions, whereas $\mathcal{R}$ exposes the execution constraints needed to verify, connect, and reorganize these fragments for the target task. Accordingly, we use symbolic reasoning primarily as a verifier and feedback mechanism, rather than as a standalone plan generator.

If the adapted fragment fails to establish p_j, the analyzer returns the unsatisfied goal and relevant state discrepancies to the selector. The selector then revises the source fragment or object mapping, and the resulting fragment is analyzed again. This iterative process produces a candidate set

$$\mathcal{C} = \{(\widetilde{f}_j, P_j, E_j)\}_{j=1}^{m} \tag{5}$$

for subsequent composition.

Fragment ordering. The VLM-based trajectory composer receives the instruction g, goal predicates G, and candidate set $\mathcal{C}$. Treating each fragment as an indivisible procedural unit, it determines an execution order $\sigma = (\sigma_1, \ldots, \sigma_m)$ based on the task requirements and inter-fragment state dependencies, where σ is a permutation of $\{1, \ldots, m\}$. The ordered fragments $(\widetilde{f}_{\sigma_1}, \ldots, \widetilde{f}_{\sigma_m})$ define the global task workflow, while any state gaps between consecutive fragments are resolved through connection planning.

Connection planning. Starting from the initially known symbolic state $\widehat{s}_0$, the analyzer propagates the state through the ordered fragments and checks the entry conditions of each fragment. Let $\widehat{s}_{k-1}$ denote the accumulated symbolic state before the kth ordered fragment. Its unmet preconditions are

$$\Delta_k = \{p \in P_{\sigma_k} \mid \widehat{s}_{k-1} \not\models p\}. \tag{6}$$

If $\Delta_k \neq \varnothing$, the analyzer provides the composer with the current state and missing conditions. The composer then generates a connecting action sequence B_k from the available action space to establish the required entry conditions. When all preconditions already hold, B_k is empty. Connecting sequences may be generated for multiple fragment boundaries in a single revision, producing the candidate plan

$$\pi = B_1 \oplus \widetilde{f}_{\sigma_1} \oplus \cdots \oplus B_m \oplus \widetilde{f}_{\sigma_m}, \tag{7}$$

where $\oplus$ denotes sequence concatenation.

The symbolic analyzer then verifies the complete candidate plan. It checks the preconditions of each connecting action and propagates the resulting symbolic state. Let $\bar{s}_k$ denote the state reached after executing B_k. If $\bar{s}_k$ satisfies the entry conditions of the subsequent fragment, the analyzer applies the fragment effects to obtain the next accumulated state:

$$\begin{aligned} \bar{s}_k &= \mathrm{Exec}_{\mathcal{R}}(\widehat{s}_{k-1}, B_k), \\ \widehat{s}_k &= \mathrm{Apply}_{\mathcal{R}}(\bar{s}_k, E_{\sigma_k}), \qquad \bar{s}_k \models P_{\sigma_k}. \end{aligned} \tag{8}$$

Any executability violation is returned to the composer for revising the fragment order or connecting actions. Once the complete sequence is executable, the analyzer verifies that the predicted final state

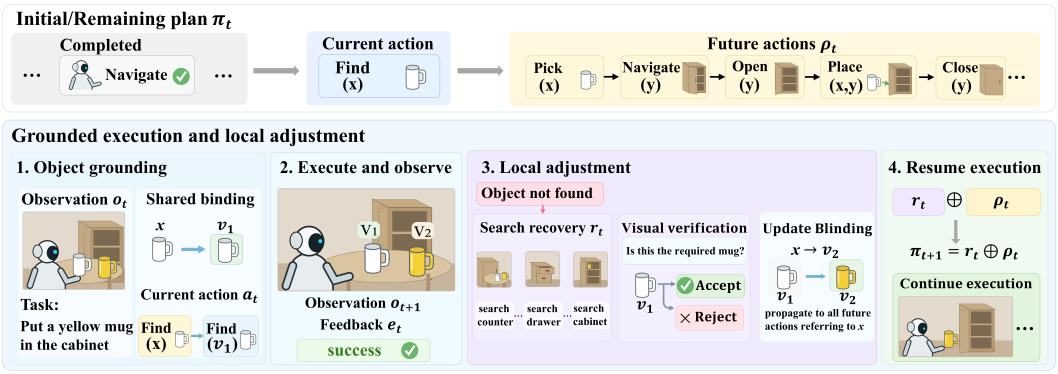

Figure 2: **Closed-loop plan execution with local grounding adjustments.** The agent grounds each planned action to a concrete object instance and executes it sequentially. Environmental feedback triggers local search recovery, while visual queries verify appearance or spatial requirements. When an instance is rejected, the updated binding is propagated to the remaining plan before execution resumes.

jointly satisfies all task goals:

$$\widehat{s}_m \models \bigwedge_{p \in G} p. \tag{9}$$

A candidate plan that passes both the executability and goal-satisfaction checks is adopted as the initial plan π^{init} for closed-loop execution.

4.2 Grounded Execution and Local Adjustment

The initial plan π^{init} specifies the task workflow, while closed-loop execution grounds this workflow in concrete objects in the environment (Figure 2). At each step, the agent binds the current action target to an observed instance, executes the action, and updates the remaining plan from the resulting observation and feedback. If the target is not found or requires visual verification, the agent performs local search or replaces the object binding and propagates the new binding to future actions before resuming the plan.

During execution, an object binding β_t maps task objects in the plan to the currently selected environmental instances. The first action of the pending sequence π_t is grounded through this binding:

$$a_t = \text{Bind}(\text{head}(\pi_t), \beta_t). \tag{10}$$

Actions referring to the same task object share its binding. The action, observation, and environmental feedback from each interaction are added to the history H_t, informing subsequent search and instance adjustment.

Search planner. The search planner, implemented with a VLM, generates search actions based on the task requirements, interaction history, and environmental feedback when an object search fails. Once the object is located, execution resumes with the remaining actions of the original plan.

Visual verifier. For appearance or spatial requirements in the instruction, the visual verifier identifies in advance the objects to inspect and the corresponding queries, placing checkpoints after the relevant Find actions. At each checkpoint, the verifier answers the query using the current image and determines whether the found instance satisfies the requirement. If it does not, another instance of the same class is selected as a candidate, searched for, and verified.

Instance replacement affects the remaining plan through an update to the object binding. For a task object x whose instance is to be replaced and a new candidate instance v', the updated binding is

$$\beta_{t+1} = \beta_t[x \mapsto v']. \tag{11}$$

All remaining actions involving x are grounded through the updated binding, propagating a single instance adjustment to subsequent relevant operations.

Table 1: **Success rates (%) on EB-ALFRED and EB-Habitat.** We compare our method with standard VLM planning and training-free methods. Avg averages Base and Long Horizon. Subscripts indicate the absolute performance improvement relative to the base model.

	EB-ALFRED			EB-Habitat		
Method	**Avg**	**Base**	**Long**	**Avg**	**Base**	**Long**
Closed-source VLMs						
GPT-4o	59	64	54	75	86	64
Claude-3.7-Sonnet	69	68	70	68	90	46
Claude-3.5-Sonnet	62	72	52	77	**96**	58
Open-source VLMs						
Qwen2.5-VL-72B-Ins	42	50	34	46	74	18
InternVL3-78B	37	38	36	62	84	40
Gemma-3-27B-it	34	42	26	45	68	22
Qwen3-VL-32B-Ins	52	56	48	57	76	38
Training-free Methods						
ELITE (Qwen2.5-VL-72B-Ins)	61	60	62	63	86	40
BrainMem (Qwen2.5-VL-72B-Ins)	**73**	72	**74**	77	92	62
Ours (Qwen3-VL-32B-Ins)	**73** +21	**76** +20	70 +22	**79** +22	90 +14	**68** +30

Plan progression and local adjustment are represented through updates to the remaining sequence. Let $\rho_t = \mathrm{tail}(\pi_t)$ denote the actions pending after the current action, and let r_t denote the local action sequence required to locate the current object, either by recovering the search or by searching for a replacement instance. The remaining plan is updated as

$$\pi_{t+1} = \begin{cases} \rho_t, & \text{if the action succeeds and no adjustment is needed,} \\ r_t \oplus \rho_t, & \text{if search fails or visual verification fails.} \end{cases} \tag{12}$$

Here, $\oplus$ denotes sequence concatenation. This update preserves the overall task workflow while focusing closed-loop decisions on object localization and instance verification.

5 Experiments

We evaluate the method on EB-ALFRED and EB-Habitat to examine whether it improves task completion across vision-language backbones, whether it can reuse relatively simple demonstrations for complex long-horizon tasks, and how demonstration-fragment reuse and symbolic analysis contribute.

5.1 Experimental Setup

Benchmarks. We use EB-ALFRED and EB-Habitat from EmbodiedBench (Yang et al., 2025). Both provide egocentric images and natural-language instructions and support interaction through high-level actions in household environments. EB-ALFRED adapts ALFRED tasks (Shridhar et al., 2020) in AI2-THOR (Kolve et al., 2017) covers 7 household task types utilizing 8 high-level skills.EB-Habitat is built on Habitat 2.0 (Szot et al., 2021) and focuses on object rearrangement. Its navigation targets are restricted to receptacles and supporting surfaces, requiring search to locate objects. Our main comparisons focus on the Base and Long Horizon subsets of both benchmarks. We additionally report results across all EB-ALFRED subsets: Base, Common Sense, Complex Instruction, Visual Appearance, Spatial Awareness, and Long Horizon.

Baselines. We compare four categories of baselines: (1) *Closed-source VLMs*: GPT-4o, Claude-3.7-Sonnet, and Claude-3.5-Sonnet; (2) Open-source VLMs: Qwen2.5-VL-72B-Instruct, InternVL3-78B, Gemma-3-27B-it, and Qwen3-VL-32B-Instruct; (3) Training-based methods: ERA (Chen et al., 2025) and RoboAgent (Xu et al., 2026b); (4) Training-free methods: ELITE (Wei et al., 2026), BrainMem (Ma et al., 2026), and WorldMind (Ren et al., 2026).

Table 2: **Success rates (%) across the six EB-ALFRED task categories.** We compare our method with standard VLM planning and existing embodied-agent methods. Avg averages all six categories; dashes indicate unavailable results. Subscripts indicate the absolute performance improvement relative to the base model.

Method	Avg	Base	Common	Complex	Visual	Spatial	Long
Closed-source VLMs							
GPT-4o	56.3	64	54	68	46	52	54
Claude-3.7-Sonnet	67.7	68	**68**	70	68	62	70
Claude-3.5-Sonnet	64.0	72	66	**76**	60	58	52
Open-source VLMs							
Qwen2.5-VL-72B-Ins	39.7	50	42	42	36	34	34
InternVL3-78B	39.0	38	34	46	42	38	36
Gemma-3-27B-it	37.0	42	40	48	30	36	26
Qwen3-VL-32B-Ins	45.7	56	50	46	42	32	48
Training-based Methods							
ERA (Qwen2.5-VL-3B-Ins)	–	72	54	72	62	**66**	–
RoboAgent (Qwen2.5-VL-3B-Ins)	67.0	72	48	64	**78**	60	**80**
Training-free Methods							
ELITE (Qwen2.5-VL-72B-Ins)	–	55	42	55	50	47	–
WorldMind (GPT-4.1-mini)	–	50	58	54	42	42	–
Ours (Qwen3-VL-32B-Ins)	**68.0** +22.3	**76** +20	66 +16	70 +24	62 +20	64 +32	70 +22

Evaluation Settings. We use the demonstrations provided by each benchmark. We measure success rate (SR), the proportion of tasks satisfying all environmental goal conditions, and goal completion (GC), the mean fraction of goal conditions satisfied at termination. Our evaluations use the same action space, success criteria, and interaction budget across compared variants. Input images have resolution 500×500. Each task permits at most 30 high-level actions and terminates after 10 cumulative invalid actions. We set `temperature` 0 and `seed` 0 to reduce randomness in our evaluations. Methods sharing a backbone use the same per-call output limit.

5.2 Main Results

Table 1 compares our method with standard VLM planning and training-free methods. With the same Qwen3-VL-32B-Ins backbone, our method raises average SR from 52% to 73% on EB-ALFRED and from 57% to 79% on EB-Habitat, gains of 21 and 22 percentage points. The largest subset gain occurs on EB-Habitat Long Horizon, where SR rises from 38% to 68%. Among the reported methods, ours ties BrainMem for the highest EB-ALFRED average and achieves the highest EB-Habitat average.

Table 2 shows gains across all six EB-ALFRED categories. The six-category average rises from 45.7% to 68.0% (+22.3 points). Spatial Awareness and Complex Instruction show the largest gains, at 32 and 24 points, respectively. Our method achieves the highest Base SR (76%) and the second-highest results in Common Sense (66%), Spatial Awareness (64%), and Long Horizon (70%).

5.3 Long-Horizon Planning with Different Demonstrations

We evaluate whether demonstrations of simpler tasks can support planning for EB-ALFRED Long Horizon tasks. Using Qwen3-VL-32B-Instruct as the backbone, we compare standard planning and our method with three demonstration sets: five Base examples, the standard ten Base examples, and seven dedicated Long Horizon examples. The Base examples contain simpler workflows, whereas the Long Horizon examples are closer in complexity to the evaluation tasks.

Table 3 shows that our method improves SR from 16% to 50% with Base 5-shot demonstrations and from 20% to 54% with Base 10-shot demonstrations, a gain of 34 percentage points in both settings.

Table 3: Success rate (SR) and goal completion (GC) on EB-ALFRED Long Horizon with three demonstration sets.

	Base 5-shot		Base 10-shot		Long Horizon 7-shot	
Method	**SR**	**GC**	**SR**	**GC**	**SR**	**GC**
Qwen3-VL-32B-Ins	16	28.4	20	35.2	38	47.6
Ours (Qwen3-VL-32B-Ins)	50	63.3	54	66.2	70	76.8

Table 4: Ablations on EB-ALFRED (SR, %). Parentheses show percentage-point changes from the full method.

Variant	**Base**	**Long Horizon**
Full method	**76**	**70**
Without fragment reuse	70 (−6)	64 (−6)
Without symbolic analysis	68 (−8)	60 (−10)

GC increases from 28.4% to 63.3% and from 35.2% to 66.2%, respectively. With the dedicated Long Horizon 7-shot set, our method achieves 70% SR and 76.8% GC, compared with 38% and 47.6% for standard planning. Both methods perform best with the dedicated Long Horizon examples, while our method retains a large advantage when planning from simpler Base demonstrations.

5.4 Ablation Studies

We ablate two components on EB-ALFRED Base and Long Horizon using Qwen3-VL-32B-Instruct. Base uses the standard 10-shot demonstrations, and Long Horizon uses its dedicated seven-shot set. All variants retain goal-predicate generation, the planning revision budget, and the closed-loop execution mechanism in Section 4.2.

Without fragment reuse. The VLM generates local action sequences for the same goals using the same demonstrations, rather than selecting contiguous demonstrated fragments. Symbolic analysis, composition, and feedback-guided revision remain unchanged.

Without symbolic analysis. Fragment selection, object adaptation, and composition remain unchanged. The VLM judges goal support, sequence connections, and plan completion without symbolically derived preconditions, effects, or validation feedback.

Table 4 shows that removing fragment reuse reduces SR by 6% on both task sets. Replacing symbolic analysis with VLM reasoning reduces SR by 8% on Base and 10% on Long Horizon. Both components improve planning, with the larger reduction on Long Horizon occurring when symbolic analysis is removed.

6 Conclusion

We presented COPE, a symbolic compositional planning framework that transfers procedural experience from limited demonstrations to new embodied tasks. It preserves the action dependencies within contiguous demonstrated fragments while using symbolic preconditions and effects to reorganize their order and connections around the target goal. During execution, feedback-guided grounding adjusts the plan to the observed environment. On EB-ALFRED and EB-Habitat, COPE improves the average success rate by 21% and 22% over standard planning with the same backbone. Together, these results highlight how limited demonstrations can support new complex tasks: preserve local procedural dependencies while rebuilding the global workflow for the target goal.

AI USE STATEMENT

Generative AI tools assisted with parts of the code implementation and with language polishing of the manuscript.

REPRODUCIBILITY STATEMENT

Section 4 specifies trajectory composition, symbolic analysis, and grounded execution. Section 5.1 describes the benchmarks, demonstration sets, model backbones, and interaction budgets. Sections 5.3 and 5.4 define the main comparisons and ablations. Appendices A and B provide the EB-ALFRED implementation details and prompt templates.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Akhil Bagaria, Jason K. Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 521–531. PMLR, 2021.

Hanyang Chen, Mark Zhao, Rui Yang, Qinwei Ma, Ke Yang, Jiarui Yao, Kangrui Wang, Hao Bai, Zhenhailong Wang, Rui Pan, Mengchao Zhang, Jose Barreiros, Aykut Onol, ChengXiang Zhai, Heng Ji, Manling Li, Huan Zhang, and Tong Zhang. ERA: Transforming VLMs into embodied agents via embodied prior learning and online reinforcement learning. *arXiv preprint arXiv:2510.12693*, 2025. doi: 10.48550/arXiv.2510.12693.

Jinbang Huang, Yixin Xiao, Zhanguang Zhang, Mark Coates, Jianye Hao, and Yingxue Zhang. One demo is all it takes: Planning domain derivation with LLMs from a single demonstration. In *International Conference on Learning Representations*, 2026.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 2022.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An interactive 3D environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017.

Xiang Li, Ning Yan, and Masood Mortazavi. Embodied task planning via graph-informed action generation with large language models. In *International Conference on Machine Learning*, 2026.

Weiyu Liu, Neil Nie, Ruohan Zhang, Jiayuan Mao, and Jiajun Wu. Learning compositional behaviors from demonstration and language. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 1992–2028. PMLR, 2025.

Yunhao Luo, Utkarsh Mishra, Yilun Du, and Danfei Xu. Generative trajectory stitching through diffusion composition. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1267.

Xiaoyu Ma, Lianyu Hu, Wenbing Tang, Zixuan Hu, Zeqin Liao, Zhizhen Wu, and Yang Liu. BrainMem: Brain-inspired evolving memory for embodied agent task planning. *arXiv preprint arXiv:2604.16331*, 2026.

Baochang Ren, Yunzhi Yao, Rui Sun, Shuofei Qiao, Ningyu Zhang, and Huajun Chen. Aligning agentic world models via knowledgeable experience learning. In *Findings of the Association for Computational Linguistics: EMNLP*, 2026.

Gabriel Sarch, Lawrence Jang, Michael J. Tarr, William W. Cohen, Kenneth Marino, and Katerina Fragkiadaki. VLM agents generate their own memories: Distilling experience into embodied programs of thought. In *Advances in Neural Information Processing Systems*, volume 37, pp. 75942–75985, 2024. doi: 10.52202/079017-2418.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10740–10749, 2020.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10161317.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. LLM-Planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2998–3009, 2023.

Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimír Vondruš, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Yuki Wang, Gonzalo Gonzalez-Pumariega, Yash Sharma, and Sanjiban Choudhury. Demo2Code: From summarizing demonstrations to synthesizing code via extended chain-of-thought. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0655.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025.

Bingqing Wei, Zhongyu Xia, Dingai Liu, Xiaoyu Zhou, Zhiwei Lin, and Yongtao Wang. ELITE: Experiential learning and intent-aware transfer for self-improving embodied agents. *arXiv preprint arXiv:2603.24018*, 2026.

Haoming Xu, Zhenlin He, Hengyi Wang, Jiafeng Xu, and Hao Dong. Mimir: A neuro-symbolic memory system with dynamic grounding for embodied agents in interactive environments. *arXiv preprint arXiv:2608.04933*, 2026a.

Peiran Xu, Jiaqi Zheng, and Yadong Mu. RoboAgent: Chaining basic capabilities for embodied task planning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15276–15290, 2026b.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 70576–70631. PMLR, 2025.

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*, 2023.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Jiacong Zhou, Jiaxu Miao, Xianyun Wang, and Jun Yu. Enhancing LLM planning for robotics manipulation through hierarchical procedural knowledge graphs. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4246.

A METHOD DETAILS

This appendix instantiates the method on EB-ALFRED. We use its high-level action interface, dynamically constructed action space, and environment feedback to make each stage concrete. The same demonstrations shown to the baseline planner are retained, but their action sequences are numbered so that the selector can identify inclusive source intervals. Variables enclosed in angle brackets are filled at inference time. Structured outputs are constrained by the JSON schemas stated below.

A.1 FRAGMENT-LEVEL SYMBOLIC ANALYSIS

The symbolic analyzer processes an adapted fragment in action order without assuming a complete initial scene. When an action requires a fact whose initial value is unknown, the analyzer records that fact as an entry precondition. It then applies the action's state changes before analyzing the next action. A contradiction between an established state and a later action requirement is reported at the action where it occurs. The resulting fragment specification contains its external preconditions and net effects; state changes that depend on an object being in a receptacle remain conditional on that relation.

In the EB-ALFRED implementation, the rule set tracks the hand-held object, the object selected by a search action, the current placement target, open and on/off states, object-processing states, and placement relations. The analyzer retains facts established by earlier actions until a subsequent action changes them. The rules are fixed for action types and do not read hidden simulator state or encode task-specific demonstration trajectories.

The goal vocabulary used in this implementation is `in(object, container)`, `on(object, surface)`, `cleaned(object)`, `heated(object)`, `cooled(object)`, `sliced(object)`, `holding(object)`, and `toggled_on(device)`. Object arguments are grounded to names exposed by the current EB-ALFRED action space. The analyzer applies deterministic rules for FIND, PICKUP, PUTDOWN, OPEN, CLOSE, TOGGLEON, TOGGLEOFF, and SLICE. For example, FIND changes the selected object or placement target; PICKUP requires the hand to be empty and the selected object to match its argument; and PUTDOWN requires a held object and a selected destination, after which it establishes an `in` or `on` relation and empties the hand. Device-mediated effects such as cleaning, heating, and cooling are applied only when their corresponding object–device relation and device state are established by the action sequence.

A.2 COMPOSITION CHECK AND FEEDBACK

After fragment ordering, the analyzer propagates an accumulated symbolic state through both the connecting sequences and the selected fragments. For each boundary, it first checks the actions in the connecting sequence and applies their effects. It then compares the resulting state with the next fragment's entry preconditions. Feedback identifies the first action that cannot be executed or the conditions still missing at a fragment boundary. Once the candidate sequence is executable, the predicted final state is checked against all goal predicates. The composer receives this feedback to revise the ordering or connecting actions before the plan is executed in the environment.

Each candidate is represented by a fragment identifier, its adapted action sequence, entry preconditions, and effects. The composer initially returns an ordered list of fragment identifiers rather than reproducing their internal actions. Code expands each identifier to its complete action sequence. If the first symbolic conflict occurs at a fragment boundary, the analyzer reports the state established by the preceding fragment and the entry conditions of the next fragment. The VLM supplies actions only for that boundary, and code repeats symbolic propagation after inserting them. A final check executes the expanded plan symbolically and compares its terminal state with every goal predicate. Thus, source-interval recovery, object substitution, fragment expansion, and symbolic state propagation are deterministic; the VLM is used for semantic selection, ordering, and missing connections.

A.3 GROUNDING AND LOCAL EXECUTION UPDATES

The execution state is updated after each environmental action. A successful action contributes its rule-derived state changes; a failed action contributes only side effects explicitly reported by environmental feedback. Task-level object references remain associated with selected simulator

instances, so replacing an instance changes the grounding of all pending actions that refer to the same task object. A failed search triggers a local search sequence, while an appearance or spatial query is evaluated after the relevant object is found. Once the local issue is resolved, execution returns to the pending plan.

EB-ALFRED execution is plan preserving. Actions proceed without an additional model call while their environment feedback indicates success. A failed FIND, or a PICKUP failure caused by an inaccessible target, activates bounded search. Explicit feedback that names a receptacle is handled by code using FIND–OPEN–FIND recovery when applicable. Otherwise, code first enumerates untried instances of the target class. The VLM is called only when an ambiguous occlusion requires choosing among unsearched receptacles. After the target is found, execution resumes at the interrupted position of the original plan.

For instructions containing appearance or spatial constraints, a separate call derives verification queries before execution. A query is evaluated immediately after the corresponding FIND. If the result is false, the verifier selects an untried instance of the same object class, updates the binding shared by all pending references to that task object, and repeats the search and verification. Search and verification use bounded candidate sets; exhaustion returns control to the remaining-plan adjustment mechanism.

B PROMPT TEMPLATES

The following templates show the prompts used for EB-ALFRED. Each planning call receives the original EB-ALFRED role description, action definitions, and in-context demonstrations unless the template explicitly replaces the action list with the scene-object list. JSON schemas are enforced through the model interface and are shown in compact form.

B.1 INITIAL PLANNING

EB-ALFRED System Context

You are a robot operating in a home. Given a task, accomplish it using the defined actions.

Action descriptions and validity rules: FIND, PICKUP, PUTDOWN, DROP, OPEN, CLOSE, TOGGLEON, TOGGLEOFF, and SLICE, followed by their EB-ALFRED parameterization and preconditions.

Available action IDs and names: `<available_actions>`

Demonstrations: `<numbered_demonstrations>`

Human instruction: `<instruction>`

Goal Predicate Generation

Objects in the scene: `<scene_objects>`. Among them, only `<containers>` are containers and could be used as the second argument of `in(object, container)`.

Output all goal predicates that must hold when the task is complete. A piece mentioned in the instruction requires slicing the object first. Every predicate argument must be an exact object name from the provided scene objects. If no scene object exactly corresponds to an object in the instruction, use the most likely corresponding object from the provided scene objects.

Use only: `in(object, container)`, `on(object, surface)`, `cleaned(object)`, `heated(object)`, `cooled(object)`, `sliced(object)`, `holding(object)`, and `toggled_on(device)`.

Output: `{"goal_predicates": ["predicate(arguments)", ...]}`

Fragment Selection and Object Mapping

Goal predicates: `<goal_predicates>`

For each goal predicate, select exactly one contiguous action subsequence from the provided examples that demonstrates the required operation, regardless of the object names used in the example. Output the selected subsequences in the same order as the goal predicates. The number of selected subse-

quences must equal the number of goal predicates. For each subsequence, output its `example_id`, inclusive `start`/`end` step numbers, and `source_actions`. For each selected subsequence, output `object_mappings` as `source_object`/`target_object` pairs needed to adapt its demonstrated result to the corresponding goal predicate. Use the object names in that goal predicate for the replaced task objects.

Output: `{"selected_subsequences": [...], "object_mappings": [...]}`

Effect-Guided Fragment Revision

Goal predicates to revise: `<failed_goal_predicates>`
Previous selections, object mappings, and actual effects: `<failed_fragment_records>`

For each reported goal predicate, the adapted subsequence has the listed actual effects and does not achieve that predicate. Select a different contiguous example interval and output the object replacements needed to adapt its demonstrated result to the corresponding goal predicate. Do not select the same example interval as the previous attempt for that goal predicate.

Output: `{"selected_subsequences": [...], "object_mappings": [...]}`

Fragment Ordering

Human instruction: `<instruction>`
Goal predicates: `<goal_predicates>`
Fragments: `<fragment_id, action_sequence, preconditions, effects>`

Each fragment represents one complete action sequence. Arrange all fragments into an execution order using their preconditions and effects. Use every fragment exactly once. Output `composition`; it must contain only fragment references.

Output: `{"composition": [{"fragment": "F2"}, {"fragment": "F1"}, ...]}`

Boundary Connection

Human instruction: `<instruction>`
Ordered fragments: `<fragment IDs and action sequences>`

A connection is missing between `<previous_fragment>` and `<next_fragment>`. The state after `<previous_fragment>` is `<state_after>`, but the state required before `<next_fragment>` is `<required_state>`. Output `connecting_actions` to insert between these fragments. Use only action names from the available actions.

Output: `{"connecting_actions": [{"action_name": "..."}, ...]}`

Complete-Plan Revision after Symbolic Validation

Human instruction: `<instruction>`
Goal predicates: `<goal_predicates>`
Current composition: `<composition>`
Symbolic validation feedback: `<first failed action and missing preconditions, or missing terminal predicates>`

Revise the fragment order or the connecting actions according to the symbolic validation feedback. Keep each fragment reference intact and use only action names from the available actions for connections. Output the complete revised `composition`.

B.2 CLOSED-LOOP EXECUTION

Prompt for Search Recovery

Choose one receptacle to search for the target object.
Target object: `<target>`
Environment feedback: `<feedback>`
Receptacles not yet searched: `<candidate_receptacles>`
Output: `{"receptacle": "<candidate>"}` or `{"receptacle": null}`.

Visual or Spatial Check Generation

Human instruction: `<instruction>`
Plan: `<initial_plan>`
Identify the objects whose appearance or spatial relation is specified in the instruction. For each object, output its name from the plan as `object`, its description from the instruction as `description`, and a visual query to check it after FIND. Refer to the found object as "this object" in the query. The query must be answerable with true or false.
Output: `{"visual_checks": [{"object": "...", "description": "...", "query": "..."}, ...]}`

Visual Verification and Instance Replacement

Verification call: Answer the visual query using the current image. Output true or false.
Visual query: `<query>`

Replacement call after a false result:
Human instruction: `<instruction>`
Object description: `<description>`
Remaining objects: `<untried_instances>`
Select the remaining object that best matches the object described in the instruction. Output its name.