arXiv:2511.02424v1 [cs.AI] 4 Nov 2025

ReAcTree: Hierarchical LLM Agent Trees with Control Flow for Long-Horizon Task Planning

Jae-Woo Choi[1], Hyungmin Kim[1,2], Hyobin Ong[1,2], Minsu Jang[1,2],
Dohyung Kim[1,2], Jaehong Kim[1], Youngwoo Yoon[1,2†]
[1]Electronics and Telecommunications Research Institute
[2]University of Science and Technology
Daejeon, Republic of Korea
{jwchoi0717,khm159,ohnghb,minsu,dhkim008,jhkim504,youngwoo}@etri.re.kr

ABSTRACT

Recent advancements in large language models (LLMs) have enabled significant progress in decision-making and task planning for embodied autonomous agents. However, most existing methods still struggle with complex, long-horizon tasks because they rely on a monolithic trajectory that entangles all past decisions and observations, attempting to solve the entire task in a single unified process. To address this limitation, we propose *ReAcTree*, a hierarchical task-planning method that decomposes a complex goal into more manageable subgoals within a dynamically constructed agent tree. Each subgoal is handled by an LLM agent node capable of reasoning, acting, and further expanding the tree, while control flow nodes coordinate the execution strategies of agent nodes. In addition, we integrate two complementary memory systems: each agent node retrieves goal-specific, subgoal-level examples from *episodic memory* and shares environment-specific observations through *working memory*. Experiments on the WAH-NL and ALFRED datasets demonstrate that *ReAcTree* consistently outperforms strong task-planning baselines such as *ReAct* across diverse LLMs. Notably, on WAH-NL, *ReAcTree* achieves a 61% goal success rate with Qwen 2.5 72B, nearly doubling *ReAct*'s 31%.

KEYWORDS

Task planning, large language models, decision-making, hierarchical planning, behavior trees, embodied agents

1 INTRODUCTION

Large language models (LLMs) have recently exhibited remarkable reasoning capabilities by generating intermediate reasoning steps [23, 48]. These advances open up new possibilities for decision-making and task planning in embodied autonomous agents, with the long-term objective of enabling them to fulfill high-level natural language commands autonomously. As early progress toward this goal, several works [3, 19, 25, 41] have demonstrated that LLMs can leverage their pre-trained world knowledge to generate mid-level action sequences without additional training, primarily through prompting rather than handcrafted heuristics or learned policies [15, 52].

The decision-making and planning capabilities of LLMs can be further improved by incorporating observations or feedback from external environments. Recent approaches explore adapting the next action [20, 55, 56], adjusting the entire plan [37], or refining both the plan and the next action [42, 47]. However, these methods frequently use a single, monolithic trajectory that entangles all past decisions and observations from multiple subgoals, which increases the risk of hallucination and logical failures [26]. Another line of research investigates multiple reasoning paths [46, 54, 58] for general reasoning tasks. Such methods typically assume the capability to simulate multiple paths and revert to previous states [18, 57, 60], which is rarely feasible in real-world scenarios due to irreversible actions (e.g., slicing an apple) and partial observability.

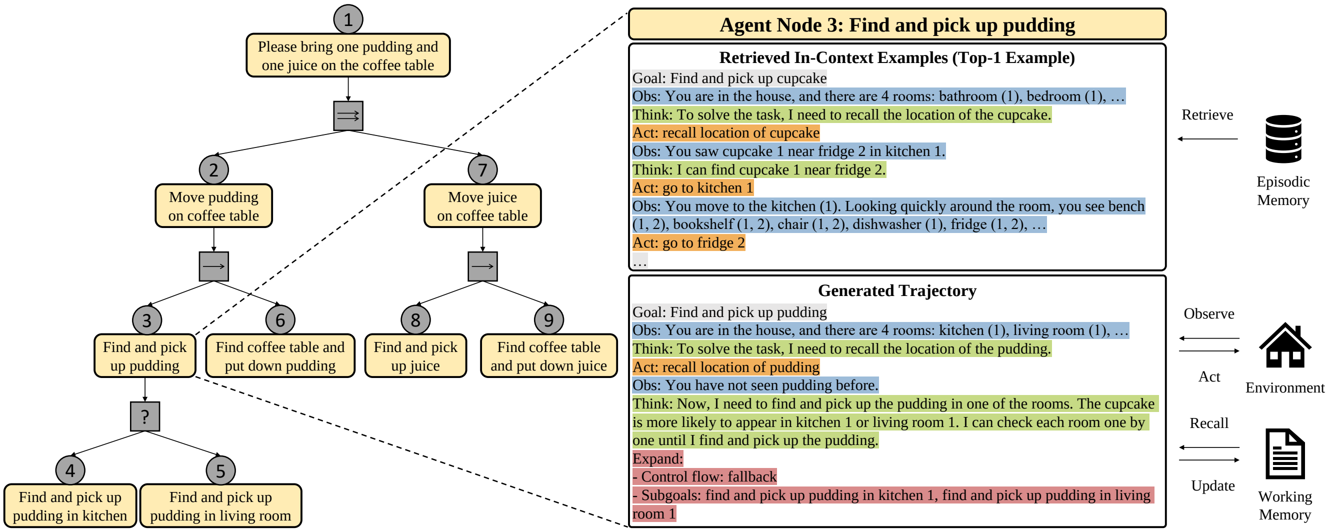
To tackle these challenges, we introduce *ReAcTree*, a novel framework that dynamically constructs an agent tree in the subgoal space rather than an action tree over primitive actions. Each LLM-powered agent node is responsible for a subgoal and extends the *ReAct* paradigm [55]: it can reason, act, or expand the tree by proposing new subgoals with an associated control flow when a task is too complex. Control flow nodes, inspired by Behavior Trees [12], coordinate these agents by sequencing, fallback, or parallel execution. Conceptually, *ReAcTree* extends the *Least-to-Most* prompting strategy [59] from static reasoning to dynamic, agentic planning. By decomposing a problem into semantically isolated subgoals and providing each with its own focused context, *ReAcTree* prevents the propagation of reasoning errors and makes long-horizon tasks more tractable for LLMs. The hierarchical structure of *ReAcTree* is illustrated in Figure 1.

To enhance the in-context learning and coordination capabilities of the proposed agent architecture, we incorporate two complementary memory systems. First, *episodic memory* assists each agent node in effective in-context learning by providing subgoal-level examples from past experiences that are semantically similar to the current subgoal. It can be bootstrapped from a small set of manually collected trajectories and gradually expanded through successful runs. Second, *working memory* functions as a shared blackboard, allowing agent nodes to exchange critical observations (e.g., the locations of movable objects). This collective situational awareness reduces redundant searches and mitigates hallucinations. Together, these memory systems enable *ReAcTree* to learn from the past and coordinate in the present, strengthening the robustness of agentic decision-making under partial observability.

We evaluate *ReAcTree* by extending LoTa-Bench [10], a benchmark providing two household task environments (WAH-NL [10, 33] with VirtualHome [32] and ALFRED [38] with AI2THOR [24]), to a more challenging partially observable setting. This setup reflects real-world constraints and tests agentic decision-making under uncertainty. Across both environments, *ReAcTree* consistently outperforms strong baselines across various LLMs. On WAH-NL,

---

[†]Corresponding author.


1
Please bring one pudding and one juice on the coffee table
⇒
2
Move pudding on coffee table
→
7
Move juice on coffee table
→
3
Find and pick up pudding
6
Find coffee table and put down pudding
8
Find and pick up juice
9
Find coffee table and put down juice
?
4
Find and pick up pudding in kitchen
5
Find and pick up pudding in living room
Retrieve
Episodic Memory
Observe
Act
Environment
Recall
Update
Working Memory

Agent Node 3: Find and pick up pudding

Retrieved In-Context Examples (Top-1 Example)

Goal: Find and pick up cupcake
Obs: You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), …
Think: To solve the task, I need to recall the location of the cupcake.
Act: recall location of cupcake
Obs: You saw cupcake 1 near fridge 2 in kitchen 1.
Think: I can find cupcake 1 near fridge 2.
Act: go to kitchen 1
Obs: You move to the kitchen (1). Looking quickly around the room, you see bench (1, 2), bookshelf (1, 2), chair (1, 2), dishwasher (1), fridge (1, 2), …
Act: go to fridge 2
…

Generated Trajectory

Goal: Find and pick up pudding
Obs: You are in the house, and there are 4 rooms: kitchen (1), living room (1), …
Think: To solve the task, I need to recall the location of the pudding.
Act: recall location of pudding
Obs: You have not seen pudding before.
Think: Now, I need to find and pick up the pudding in one of the rooms. The cupcake is more likely to appear in kitchen 1 or living room 1. I can check each room one by one until I find and pick up the pudding.
Expand:
- Control flow: fallback
- Subgoals: find and pick up pudding in kitchen 1, find and pick up pudding in living room 1

Figure 1: An illustrative example of how *ReAcTree* generates an agent tree for the natural language instruction: *Please bring one pudding and one juice to the coffee table.* The left side shows the hierarchical structure with agent nodes (circles) and control flow nodes (squares); the number inside each circle denotes the execution order, and the attached text box specifies the corresponding subgoal. Control flow nodes are labeled by their types (→ for sequence, ? for fallback, and ⇒ for parallel). The right side highlights the decision-making trajectory of agent node 3, including observation, reasoning, acting, subgoal expansion, and the use of episodic and working memory.

it achieves a 61% goal success rate with Qwen 2.5 72B, nearly doubling *ReAct*'s 31% under the same model size. Notably, even with a much smaller 7B model, it reaches 37%, showing that *ReAcTree* maintains strong performance despite reduced model capacity. The contributions of its core components are further substantiated by ablation studies on memory systems and control flow types, as well as in-depth analyses of computational cost and failure modes.

In summary, this paper makes the following contributions: (1) *ReAcTree*, a hierarchical planning framework for long-horizon tasks that dynamically constructs an LLM agent tree, where each agent node independently solves a subgoal while control flow nodes coordinate their execution into a coherent plan; (2) two memory systems: episodic memory that enhances in-context learning ability of agent nodes, and working memory that facilitates information sharing across agent nodes to support collective situational awareness; and (3) extensive experiments on LoTa-Bench under partially observable settings, demonstrating the effectiveness of *ReAcTree* across strong baselines and providing detailed analysis through ablations, computational cost analysis, and failure case studies. The code is available https://github.com/Choi-JaeWoo/ReAcTree.

2 RELATED WORK

LLM-based Embodied Agents. LLMs such as GPT-3, GPT-4, PaLM, LLaMA, and Qwen [2, 6, 11, 14, 53] have demonstrated impressive few-shot in-context learning capabilities, advancing the state of the art in NLP tasks. Prompting techniques [16, 23, 48, 59] further enhance reasoning by encouraging articulation of intermediate steps. Building on these advancements, research has increasingly explored text-based applications to embodied decision-making. Early studies [3, 19, 41] showed that LLMs could generate mid-level action sequences without additional training, while later studies introduced code-based plan generation [25, 40], the extension of classical agent architectures [21], and the integration of environmental feedback or tools [7, 20, 28, 36, 56]. *ReAct* [55] enhanced planning by prompting explicit intermediate reasoning, while *Reflexion* [37] applied iterative self-refinement [9, 29, 31, 49], offering additional flexibility for long-horizon tasks.

Hierarchical Task Planning with LLMs. To tackle more complex, long-horizon tasks, researchers have introduced hierarchical frameworks that decompose goals into manageable planning levels. *AdaPlanner* [42] and *DEPS* [47] adopt bi-level hierarchies, refining both an overall plan and next-step decisions with environmental feedback. Other methods combine classical task-and-motion planning with learned low-level controllers guided by LLM reasoning [13, 27, 39, 45, 50]. Some works leverage behavior trees [4, 8, 44] to organize high-level actions in specialized domains such as human-robot cooking or robotic assembly. However, they typically rely on predefined structures or domain-specific routines. In contrast, our approach, *ReAcTree*, enables dynamic task decomposition, expanding subgoals in complex environments without being tied to a specific domain.

Tree Search-Based Planning with LLMs. A prominent line of research explores multiple reasoning paths to evaluate diverse hypotheses before committing to a final solution. Such approaches [5, 17, 46, 51, 54, 58] show that systematically branching reasoning steps significantly improves performance in reasoning tasks. Several studies extend this idea to agentic planning. For instance, *LLM-MCTS* [57] and *ToolChain** [60] search action trees using Monte Carlo Tree Search and A* search, respectively. *Tree-Planner* [18]

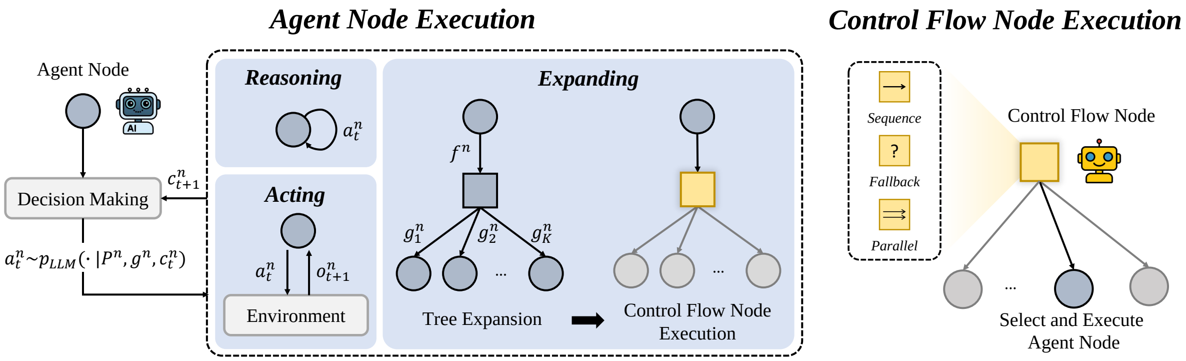


Figure 2: Illustration of agent node execution and control flow node execution in *ReAcTree*.

independently samples planning paths and merges them into an action tree, then makes decisions based on grounded observations at each node. However, these methods typically assume reversible actions (i.e., rolling back to prior states) in simulators, which limits their applicability in real-world settings. In contrast, *ReAcTree* constructs a dynamically expanding LLM agent tree that decomposes complex goals into subgoals, delegates them to agent nodes, and coordinates their execution via behavior tree-inspired control flow, replacing search-based exploration with agent coordination.

3 PRELIMINARIES

Problem Formulation. We consider the agentic planning problem as a sequential decision-making problem aimed at achieving a goal g expressed in natural language. At each time step t, the agent has access to the context $c_t = (o_1, a_1, o_2, a_2, \cdots, a_{t-1}, o_t)$, where o_i and a_i represent the observation and action at time step i, respectively. The objective of the agent is to generate the next appropriate action a_t based on the context c_t, with the aim of eventually achieving the goal g.

ReAct [55]. *ReAct* addresses this problem by interleaving reasoning and action execution using a pre-trained LLM p_{LLM}. The action policy is defined as $a_t \sim p_{LLM}(\cdot|P, g, c_t)$, where $P = (P_{sys}, P_{ic})$ is the initial prompt, composed of a system prompt P_{sys} and in-context examples P_{ic}. The key idea is to use the augmented action space, $\hat{\mathcal{A}}_t = \mathcal{A}_t \cup \mathcal{L}$, where $\mathcal{A}_t$ is the set of executable skills available at time t, and $\mathcal{L}$ is the language space representing reasoning steps or thoughts. If $a_t \in \mathcal{A}_t$, the agent executes the action and obtains a text observation from the environment. If $a_t \in \mathcal{L}$, it is called a thought or reasoning trace, which aids in the logical inference of the LLM. In this case, the agent does not receive a new observation from the environment, i.e., $o_{t+1} = \phi$.

4 REACTREE

ReAcTree is a hierarchical task-planning framework for agentic decision-making in complex environments. It dynamically constructs a tree of *agent nodes* and *control flow nodes*, as illustrated in Figure 2. Agent nodes operate as LLM-based task planners that can *reason*, *act*, and *expand* the tree with new subgoals and appropriate control flow. Control flow nodes, inspired by behavior trees [12], determine how child agent nodes are executed and how their outcomes are propagated upward in the tree. This design offers two key advantages: (1) isolating subgoals reduces hallucination and logical errors in long trajectories by keeping each agent node focused on its local context and goal, which also enables targeted in-context example selection, and (2) control flow nodes ensure robust and interpretable execution logic for reliable planning.

4.1 ReAcTree Algorithm

ReAcTree algorithm dynamically constructs an agent tree of *agent nodes* and *control flow nodes* to accomplish a natural language goal g. It begins with a single agent node for the top-level goal and dynamically grows the tree as agents decide to expand. The following describes the execution of agent nodes and control flow nodes.

Agent Nodes. Each agent node n operates as an LLM-based task planner with a specific natural language subgoal g^n, responsible for decision-making to achieve that goal. At each time step t, it accesses its context $c_t^n = (o_1^n, a_1^n, o_2^n, a_2^n, \ldots, a_{t-1}^n, o_t^n)$, where o_i^n and a_i^n represent the observation and action at time step i. It then samples an action a_t^n from an LLM policy $p_{\text{LLM}}(\cdot)$:

$$a_t^n \sim p_{LLM}(\cdot \mid P^n, g^n, c_t^n), \tag{1}$$

where the initial prompt $P^n = (P_{sys}, P_{ic}^n)$ consists of a system prompt P_{sys} and agent node-specific in-context examples P_{ic}^n.

A key feature of *ReAcTree* is its extended action space, $\hat{\mathcal{A}}_t^n = \mathcal{A}_t^n \cup \mathcal{L} \cup \mathcal{E}$, where $\mathcal{A}_t^n$ represents the set of executable skills at time t (e.g., *pick up apple 1*); $\mathcal{L}$ is the language space for self-reasoning; and $\mathcal{E} = \mathcal{F} \times \mathcal{L}$ is the expand space, with $\mathcal{F}$ and $\mathcal{L}$ denoting the set of control flow types and the language space used to express subgoals, respectively.

If the action $a_t^n \in \mathcal{A}_t^n$ or $a_t^n \in \mathcal{L}$, the agent operates as in the *ReAct* framework, either executing actions or engaging in reasoning. If $a_t^n \in \mathcal{E}$, the agent expands the tree by creating a control flow node as its child and agent nodes for the generated subgoals as grandchildren. Formally, this expansion action is expressed as $a_t^n = (f^n, [g_1^n, \ldots, g_K^n])$, where f^n is the control flow type and each g_i^n is a natural language subgoal. A control flow node n_f with type f^n is then attached as a child of node n, and agent nodes n_i with subgoals g_i^n are added as children of n_f. After expansion, node n executes the control flow node n_f and awaits its result. The agent node terminates when one of the following occurs: generating *done* (returning success), *failure*, or reaching the maximum decision count (both returning failure).

Control Flow Nodes. Each control flow node coordinates the execution of its child agent nodes according to behavior tree principles [12]. *ReAcTree* supports three types of control flow nodes: *sequence* (→), *fallback* (?), and *parallel* (⇒). The *sequence* node executes its children sequentially, returning success only if all of them succeed and failing immediately if any child fails. The *fallback* node also executes children in order but returns success as soon as the first child succeeds and fails only if all children fail. Lastly, the *parallel* node executes all children sequentially without early termination and aggregates outcomes according to a predefined policy; we adopt majority voting, which is useful for subgoals that can be attempted independently (e.g., placing multiple objects in different locations). After execution, a control flow node reports its overall success or failure to its parent node. The complete pseudocode is provided in Appendix A.

4.2 Memory Systems

To support *ReAcTree*'s hierarchical planning and decision-making, we introduce two complementary memory systems: *episodic memory* and *working memory.* Episodic memory enhances the in-context learning capability of agent nodes by storing and retrieving past subgoal-level experiences. In contrast, working memory facilitates information sharing across nodes by recording environment-specific observations, such as the latest location of movable objects, during task execution.

Episodic Memory. In *ReAcTree*, each agent node e is responsible for a particular subgoal g^e, and its entire decision-making trajectory forms a subgoal-level experience. Episodic memory M_{ep} stores experiences from agent nodes involved in task executions that ultimately succeed, even if the individual node did not complete its subgoal.

By storing experiences at the subgoal level, episodic memory maintains shorter, goal-directed trajectories than monolithic planners like *ReAct.* For instance, while *ReAct* stores a single long trajectory for *Bring pudding and juice to the coffee table*, *ReAcTree* instead stores focused trajectories for each subgoal, such as *find and pick up pudding* and *find and pick up juice.* This finer granularity enables more targeted in-context retrieval and improves decision accuracy at each agent node.

Formally, each experience is recorded as a tuple (t^e, v^e, s^e), where: (1) $t^e = (g^e, o_1^e, a_1^e, \ldots, o_T^e, a_T^e)$ denotes the full text trajectory, recording all observations o_t^e and actions a_t^e during the node's lifetime; (2) $v^e = f_{\text{sen}}(g^e)$ is the sentence embedding of the node's subgoal, computed using a pretrained encoder f_{sen} such as Sentence-BERT [34]; and (3) $s^e \in \{\textit{success}, \textit{failure}, \textit{expand}\}$ is the termination state of the agent node.

The collected episodic memory M_{ep} provides in-context examples for agent node inference. Before decision-making begins at inference time, an agent node retrieves in-context examples from M_{ep} by comparing its current subgoal g^n to the stored goals using cosine similarity. Specifically, the agent embeds its goal g^n as $v^n = f_{\text{sen}}(g^n)$ and computes the similarity with each v^e in M_{ep}:

$$\text{sim}(v^n, v^e) = \frac{v^n \cdot v^e}{\| v^n \| \, \| v^e \|}. \tag{2}$$

The agent retrieves the top k most similar experiences (with k determined by a token limit) to assemble as in-context examples. If multiple experiences have the same similarity score, *ReAcTree* samples uniformly across termination states {*success*, *failure*, *expand*} to promote diversity. In practice, episodic memory can be bootstrapped from a few manually collected trajectories and gradually augmented with additional successful executions.

Working Memory. Working memory captures environment-specific information within a single *ReAcTree* run and is shared across all agent nodes to store and recall key observations. In this work, we focus on tracking the locations of movable objects to reduce redundant interactions and mitigate hallucinations by providing accurate, environment-specific knowledge.

Working memory is integrated into agent nodes of *ReAcTree* through two mechanisms. First, the executable skill set $\mathcal{A}_t^n$ is augmented with special actions, *recall location of <movable object>*, enabling the agent to query the stored location of any movable object directly instead of actively exploring the environment. This recall action exclusively queries working memory. Second, working memory is automatically updated whenever an agent observes movable objects during interaction. For instance, if an agent opens a fridge and finds juice, it updates the working memory to reflect that juice is now in the fridge. We implement working memory as a lightweight Python dictionary that maps object classes to lists of observed instances and their associated locations (e.g., IDs, rooms, receptacles). This mechanism extends the concept of tool usage in language models [35], where the agent treats *recall location* as a specialized tool-like action.

5 EXPERIMENTS

5.1 Experimental Setup

Datasets and Simulators. We follow the evaluation protocol of LoTa-Bench [10], which provides two dataset-simulator pairs for language-based task planning: WAH-NL [10, 33] with VirtualHome [32], and ALFRED [38] with AI2THOR [24]. We extend both environments to more realistic, partially observable settings, where the agent receives limited textual observations after each action. All objects additionally include class and instance identifiers for precise grounding. We conduct primary experiments on WAH-NL with VirtualHome, as it features longer-horizon, multi-room tasks with multiple subgoals, and use ALFRED with AI2THOR, which contains relatively short-horizon, single-room, single-goal tasks, mainly for complementary validation.

WAH-NL, an extension of WAH [33], comprises 250 training and 100 test tasks across five categories. We use the training set to build episodic memory and the test set for evaluation. To improve reliability, we manually corrected four test tasks with mismatched instructions and goal conditions (see Appendix B.1 for details). ALFRED includes seven task types. In LoTa-Bench, the *pick and place two objects* task was omitted due to the absence of instance identifiers. We include this task in our evaluation by incorporating simulator-provided instance identifiers. As with WAH-NL, the training set is used to build episodic memory, and evaluation is conducted on the valid-seen and valid-unseen splits.

In both simulators, the agent executes natural language-based primitive actions. After each action, it receives an action-dependent

Table 1: Performance of *ReAcTree, ReAcTree+WM,* and 5 baselines across 7 LLMs. Both GSR and SSR (%) are reported. Bold indicates the best result, and <u>underlined</u> indicates the second best. For brevity, we use Phi-4-RP to denote Phi-4-reasoning-plus.

Method	LLaMA 3.1 8B		LLaMA 3.1 70B		Qwen 2.5 7B		Qwen 2.5 72B		Mistral 7B		Gemma 2 9B		Phi-4-RP 14B	
	GSR	SSR	GSR	SSR	GSR	SSR	GSR	SSR	GSR	SSR	GSR	SSR	GSR	SSR
ZSP	1.00	13.03	0.00	14.42	0.00	8.98	0.00	14.22	0.00	11.65	1.00	13.87	0.00	17.90
Tree-Planner$_{N=25}$	1.00	17.00	2.00	16.72	6.00	22.23	6.00	32.41	1.00	20.43	2.00	17.58	3.00	17.52
Tree-Planner$_{N=50}$	4.00	21.85	4.00	23.43	8.00	28.10	9.00	36.03	6.00	23.63	3.00	23.30	4.00	20.40
ReAct	8.00	34.25	30.00	57.05	10.00	31.82	26.00	51.38	6.00	28.18	9.00	37.20	<u>33.00</u>	48.13
ReAct+WM	16.00	42.65	<u>33.00</u>	<u>63.15</u>	13.00	39.73	31.00	54.05	9.00	31.95	11.00	39.93	<u>33.00</u>	51.28
ReAcTree	<u>21.00</u>	<u>51.98</u>	32.00	60.58	<u>18.00</u>	<u>50.20</u>	<u>48.00</u>	<u>75.13</u>	<u>11.00</u>	<u>37.92</u>	<u>26.00</u>	<u>60.43</u>	**49.00**	<u>67.47</u>
ReAcTree+WM	**30.00**	**60.77**	**58.00**	**79.27**	**37.00**	**59.63**	**61.00**	**79.58**	**15.00**	**49.57**	**38.00**	**67.08**	**49.00**	**69.30**

textual observation generated from the simulator's ground-truth state, revealing only visible objects and receptacles. VirtualHome supports six primitive actions (*go to, pick up, put down, open, close, turn on*), while AI2THOR supports eight (*go to, pick up, put down, open, close, turn on, turn off, slice*). Examples of action-observation pairs are provided in Appendix B.2.

Evaluation Metrics. We evaluate task planning performance using two metrics. The *goal success rate* (GSR) is the percentage of tasks where the agent successfully achieves the overall goal. The *subgoal success rate* (SSR) is the ratio of completed subgoals to the total number of subgoals. We report both GSR and SSR for WAH-NL, and only GSR for ALFRED, which lacks explicit subgoal definitions.

Implementation Details. We evaluate *ReAcTree* and its working memory variant, *ReAcTree+WM*, in a few-shot in-context learning setting without fine-tuning. By default, all experiments include episodic memory unless otherwise specified.

To bootstrap episodic memory, we first manually collected a small number of trajectories and used them as in-context examples to run *ReAcTree* with LLaMA-3.1 70B [14] over the training set. Only successful executions were retained in episodic memory. For WAH-NL, we collected one manual trajectory per task type and executed the full training set. For ALFRED, we collected three per type and stored up to 100 successful trajectories per type to reduce computational overhead, given the large training set (21K tasks).

The retrieval size was constrained such that the total length of retrieved trajectories used as in-context examples did not exceed 5K tokens. The decision cap was set to 200 for WAH-NL and 100 for ALFRED. We used the Guidance library [30] for text generation, employing temperature 0.0 for free-form generation and leveraging its constrained generation capabilities to deterministically select acting actions and control flow types. Prompt templates for both WAH-NL and ALFRED are available in Appendix E.1.

Baselines. We compare *ReAcTree* and *ReAcTree+WM* against five baselines: *ZSP* [19], *Tree-Planner* [18] with $N = 25$ or 50 sampled plans, *ReAct* [55], and its working memory variant *ReAct+WM*. All baselines are evaluated on WAH-NL using the same retrieval size and decision cap as *ReAcTree*. For ALFRED, we report results only for *ReAct+WM*, as it serves as the strongest baseline on WAH-NL. Further implementation details and prompt templates are provided in Appendices C and E.2, respectively.

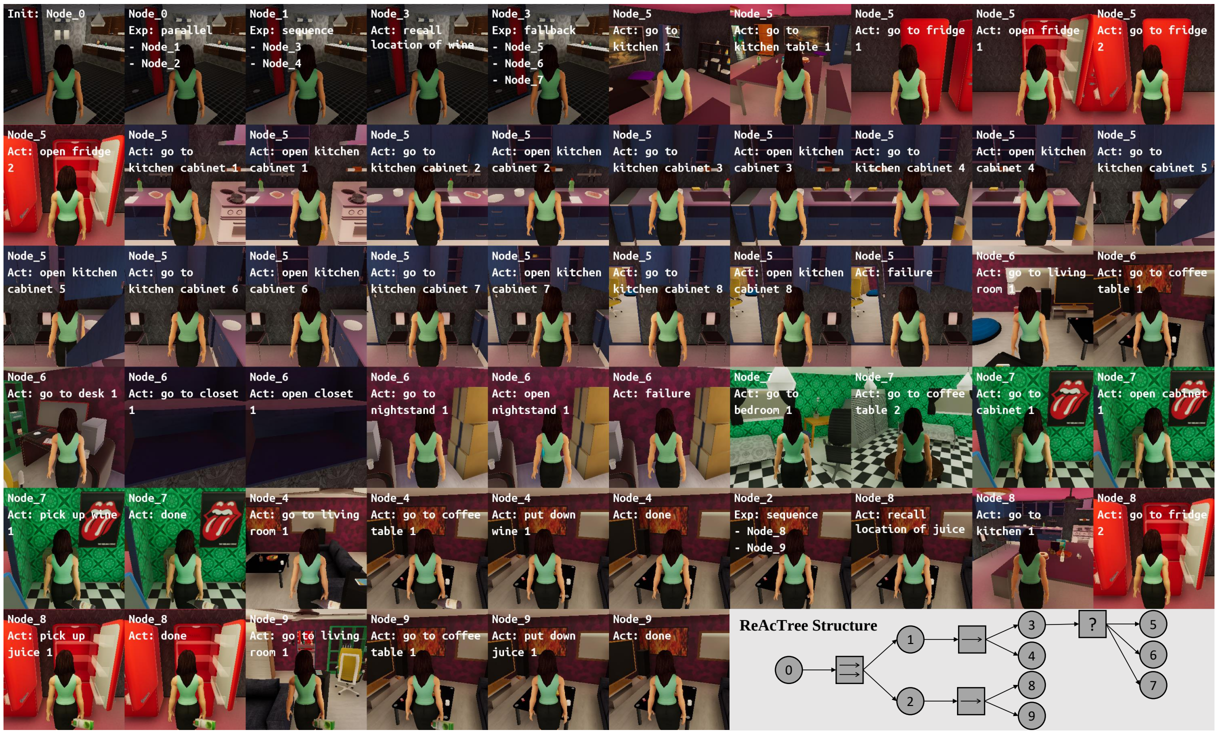
5.2 Main Results

We first present the primary evaluation results on the WAH-NL dataset, which features complex, long-horizon tasks under partial observability. Table 1 compares five baselines (*ZSP*, *Tree-Planner* with $N = 25$ or 50, *ReAct*, *ReAct+WM*) and our methods (*ReAcTree*, *ReAcTree+WM*) across seven LLMs ranging from small (7–14B) to large (70–72B), including LLaMA 3.1 [14], Qwen 2.5 [53], Mistral [22], Gemma 2 [43], and Phi-4-reasoning-plus [1] (see the Appendix D for the full list of language models and their HuggingFace identifiers). Both GSR and SSR are reported.

ZSP generates a complete plan at once and thus struggles to adapt under partial observability, resulting in poor performance. *Tree-Planner* improves this by sampling multiple high-level plans ($N =$ 25 or 50) to construct an action tree and adapting its actions based on observations. However, if none of the sampled plans succeed, the system cannot recover. This limitation becomes critical as the search space grows. For example, a typical kitchen may contain multiple *kitchen tables*, *kitchen counters*, *fridges*, and a *dishwasher*. Without knowing in advance which receptacle contains the target object, plan sampling becomes intractable and frequently fails to generate a valid plan. Even after relaxing action constraints for *pick up* and *put down* by omitting the instance identifier (e.g., *pick up apple* instead of *pick up apple 1*), the method still yields only marginal improvements over *ZSP* in our partially observable setting.

Both *ReAct* and *ReAcTree* significantly improve GSR and SSR over *ZSP* and *Tree-Planner*. While *ReAct* shows limited gains, *ReAcTree* consistently achieves higher scores across all LLMs. For example, with Qwen 2.5 72B, *ReAcTree+WM* achieves a GSR of 61% compared to 31% for *ReAct+WM*. Its SSR also rises from 54.05% to 79.58%. These consistent gains highlight the effectiveness of decomposing complex tasks into manageable subgoals, where agent nodes handle each semantically isolated subgoal within a focused context and control flow nodes coordinate execution strategies, together making long-horizon tasks more tractable under partial observability.

Notably, *ReAcTree* enables smaller LLMs to outperform baselines with much larger models. For example, *ReAcTree+WM* with Qwen 2.5 7B surpasses all baselines in GSR and most in SSR, even against Qwen 2.5 72B and LLaMA 3.1 70B. This improvement stems from two factors: (1) decomposing tasks into simpler subgoals, which keeps the accumulating decision-making trajectory focused on one subgoal at a time, and (2) retrieving subgoal-level examples that


Init: Node_0
Node_0 Exp: parallel - Node_1 - Node_2
Node_1 Exp: sequence - Node_3 - Node_4
Node_3 Act: recall location of wine
Node_3 Exp: fallback - Node_5 - Node_6 - Node_7
Node_5 Act: go to kitchen 1
Node_5 Act: go to kitchen table 1
Node_5 Act: go to fridge 1
Node_5 Act: open fridge 1
Node_5 Act: go to fridge 2
Node_5 Act: open fridge 2
Node_5 Act: go to kitchen cabinet 1
Node_5 Act: open kitchen cabinet 1
Node_5 Act: go to kitchen cabinet 2
Node_5 Act: open kitchen cabinet 2
Node_5 Act: go to kitchen cabinet 3
Node_5 Act: open kitchen cabinet 3
Node_5 Act: go to kitchen cabinet 4
Node_5 Act: open kitchen cabinet 4
Node_5 Act: go to kitchen cabinet 5
Node_5 Act: open kitchen cabinet 5
Node_5 Act: go to kitchen cabinet 6
Node_5 Act: open kitchen cabinet 6
Node_5 Act: go to kitchen cabinet 7
Node_5 Act: open kitchen cabinet 7
Node_5 Act: go to kitchen cabinet 8
Node_5 Act: open kitchen cabinet 8
Node_5 Act: failure
Node_6 Act: go to living room 1
Node_6 Act: go to coffee table 1
Node_6 Act: go to desk 1
Node_6 Act: go to closet 1
Node_6 Act: open closet 1
Node_6 Act: go to nightstand 1
Node_6 Act: open nightstand 1
Node_6 Act: failure
Node_7 Act: go to bedroom 1
Node_7 Act: go to coffee table 2
Node_7 Act: go to cabinet 1
Node_7 Act: open cabinet 1
Node_7 Act: pick up wine 1
Node_7 Act: done
Node_4 Act: go to living room 1
Node_4 Act: go to coffee table 1
Node_4 Act: put down wine 1
Node_4 Act: done
Node_2 Exp: sequence - Node_8 - Node_9
Node_8 Act: recall location of juice
Node_8 Act: go to kitchen 1
Node_8 Act: go to fridge 2
Node_8 Act: pick up juice 1
Node_8 Act: done
Node_9 Act: go to living room 1
Node_9 Act: go to coffee table 1
Node_9 Act: put down juice 1
Node_9 Act: done
ReAcTree Structure
0
1
2
3
4
?
5
6
7
8
9

Node 0	*Ensure that both a wine and a juice are on the coffee table.*	Node 5	*Find and pick up the wine in kitchen 1.*
Node 1	*Move the wine onto the coffee table.*	Node 6	*Find and pick up the wine in living room 1.*
Node 2	*Move the juice onto the coffee table.*	Node 7	*Find and pick up the wine in bedroom 1.*
Node 3	*Find and pick up the wine.*	Node 8	*Find and pick up the juice.*
Node 4	*Find the coffee table and put down the wine.*	Node 9	*Find the coffee table and put down the juice.*

Figure 3: Success case of *ReAcTree* on the WAH-NL dataset using Qwen 2.5 72B. The snapshot of each step and the tree structure of nodes are shown. The subgoal of each node is also listed.

Table 2: Performance of *ReAct+WM* and *ReAcTree+WM* on the ALFRED across 3 LLMs. GSR (%) is reported for both valid-seen and valid-unseen splits. Bold indicates the best result.

Split	Method	LLaMA 3.1		Qwen 2.5		Phi-4-RP
		8B	70B	7B	72B	14B
Valid-Seen	*ReAct+WM*	21.22	33.31	16.83	37.07	31.71
	ReAcTree+WM	**25.85**	**40.00**	**20.98**	**40.85**	**35.12**
Valid-Unseen	*ReAct+WM*	19.61	32.40	16.81	39.10	29.72
	ReAcTree+WM	**26.19**	**37.03**	**25.09**	**39.83**	**36.18**

are more directly comparable than task-level retrieval. Together, these factors enable smaller models to reason more effectively, substantially narrowing the performance gap across model scales.

To demonstrate how *ReAcTree* plans and executes complex tasks, we analyze a representative task: *Make sure there is a wine and a juice on the coffee table.* As shown in Figure 3, *ReAcTree+WM* successfully solves it by decomposing into two subgoals: *move the wine onto the coffee table* and *move the juice onto the coffee table*, and executing them in parallel. It explores multiple rooms using fallback strategies, whereas *ReAct+WM* remains confined to *kitchen 1*, failing to locate the wine in *bedroom 1*. Full trajectories are provided in Appendix G, and the corresponding failure case for *ReAct+WM* is visualized in Appendix F.

We further evaluate *ReAcTree+WM* on the ALFRED dataset. As shown in Table 2, it consistently outperforms *ReAct+WM* across all evaluated models and splits. For instance, on the valid-unseen split, it improves over *ReAct+WM* by 6.58 and 4.63 percentage points with LLaMA 3.1 8B and 70B, respectively, with similar gains observed for Qwen 2.5 and Phi-4-reasoning-plus. These results indicate that *ReAcTree* remains effective even in relatively short-horizon, single-room tasks and generalizes well to unseen environments.

Beyond quantitative gains, *ReAcTree+WM* also handles complex tasks that require more precise procedures. For instance, given the instruction *place a cooked potato slice in the fridge*, both methods slice the potato, but *ReAct+WM* skips the cooking step and places it directly in the fridge, whereas *ReAcTree+WM* correctly heats it using the microwave before storage by decomposing the instruction into four subgoals with a sequence control flow node: *find and pick up knife*, *slice and pick up potato*, *cook and pick up potato*, and *place potato in fridge*. The execution trajectories for both methods are visualized in Appendix H.

Table 3: Memory ablation results on Qwen 2.5 7B and 72B. EM, WM describes the memory configuration: none (✗, ✗), WM only (✗, ✓), EM only (✓, ✗), or EM+WM (✓, ✓). We report GSR (%) and SSR (%) with Δ improvements over the no-memory baseline.

Method	EM, WM	Qwen 2.5 7B		Qwen 2.5 72B	
		GSR (Δ)	SSR (Δ)	GSR (Δ)	SSR (Δ)
ReAct	✗, ✗	7.00	22.82	13.00	35.75
	✗, ✓	6.00 (−1.00)	19.53 (− 3.29)	18.00 (+ 5.00)	44.33 (+ 8.58)
	✓, ✗	10.00 (+3.00)	31.82 (+ 9.00)	26.00 (+13.00)	51.38 (+15.63)
	✓, ✓	13.00 (+6.00)	39.73 (+16.91)	31.00 (+18.00)	54.05 (+18.30)
ReAcTree	✗, ✗	2.00	9.32	31.00	56.82
	✗, ✓	1.00 (− 1.00)	7.45 (− 1.87)	47.00 (+16.00)	64.72 (+ 7.90)
	✓, ✗	18.00 (+16.00)	50.20 (+40.88)	48.00 (+17.00)	75.13 (+18.31)
	✓, ✓	37.00 (+35.00)	59.63 (+50.31)	61.00 (+30.00)	79.58 (+22.76)

5.3 Memory Ablation

All subsequent experiments in this section are conducted on the WAH-NL dataset, featuring long-horizon, multi-room tasks. We study the impact of *episodic memory (EM)* and *working memory (WM)* through ablation experiments with Qwen 2.5 models (7B and 72B). When EM is disabled, we use the manually annotated trajectory of a randomly selected training task as a fixed in-context example for all test tasks. For *ReAcTree*, trajectories from all agent nodes within the selected task are concatenated into a single in-context example. Table 3 reports GSR and SSR of *ReAct* and *ReAcTree* under four memory configurations: no memory, WM only, EM only, and EM+WM.

EM and WM show strong synergy. As shown in Table 3, adding either memory component generally improves performance, but combining them consistently yields the highest performance, underscoring their complementary roles. EM provides clean, semantically relevant in-context examples, while WM retains task-relevant observations.

Model scale is a critical factor in EM-disabled settings. The results also highlight a nuanced interaction between memory and model scale, especially when EM is disabled. On the 7B model, both *ReAct* and *ReAcTree* perform worse in the WM-only setting than the no-memory baseline. This counterintuitive result occurs because, without EM, the agent receives a fixed and potentially mismatched in-context example. Adding WM further introduces complexity (the `recall location of <object>` skill) that is not well-grounded by the poor example, leading to confusion for the smaller model with its limited reasoning capacity.

In contrast, the 72B model is robust to this degradation. Its performance improves significantly with WM alone (+16%p GSR for *ReAcTree*), as its greater reasoning capacity allows it to handle the additional complexity gracefully, even with suboptimal in-context examples.

This dynamic also explains why performance between *ReAcTree* and *ReAct* reverses with scale. When EM is disabled, the 7B *ReAcTree* underperforms because it struggles to parse the complex, multi-granular in-context example, whereas *ReAct*'s simpler, flat prompt structure is easier for the 7B model to follow. However, at the 72B scale, this trend reverses dramatically: the larger model has sufficient capacity to leverage the hierarchical structure, allowing *ReAcTree* to outperform *ReAct* by a large margin (e.g., +18%p in the no-memory setting and +29%p in the WM-only setting).

Hierarchical decomposition is fundamentally powerful. This analysis shows that the hierarchical structure of *ReAcTree* provides inherent value. The most compelling evidence is that even without memory support, the 72B *ReAcTree* achieves a GSR of 31.00%, substantially outperforming the 72B *ReAct* at 13.00%. This confirms that the benefits of hierarchical decomposition are fundamental to the architecture, independent of memory components.

5.4 Control Flow Ablation

We conducted a control flow ablation study of *ReAcTree* with three configurations: all control flows (*all*), sequence and fallback (*seq+fb*), and sequence only (*seq*). Following the same experimental protocol, we manually collected one trajectory per task type for each setting and bootstrapped additional successful trajectories with LLaMA 3.1 70B to construct episodic memory.

As shown in Table 4, using *all* consistently yields the best performance, while *seq+fb* performs comparably. In contrast, relying solely on *seq* significantly degrades performance. These results highlight the importance of expressive control flows, which coordinate dependencies, enable parallel execution, and recover from subgoal failures in long-horizon tasks.

Table 4: Control flow ablation results. GSR and SSR (%) are reported. Bold indicates the best performance.

Ctrl Flow	LLaMA 3.1 8B		LLaMA 3.1 70B		Qwen 2.5 7B		Qwen 2.5 72B	
	GSR	SSR	GSR	SSR	GSR	SSR	GSR	SSR
all	**30.00**	**60.77**	**58.00**	**79.27**	37.00	**59.63**	**61.00**	**79.58**
seq+fb	28.00	58.77	**58.00**	78.52	**38.00**	54.83	**61.00**	79.08
seq	18.00	45.65	36.00	61.17	15.00	32.18	46.00	63.22

5.5 Computational Cost Analysis

To analyze the computational cost of *ReAcTree*, we conducted a controlled comparison. We measured three representative configurations using LLaMA 3.1: *ReAct+WM* (70B), *ReAcTree+WM* (70B), and *ReAcTree+WM* (8B). All methods were evaluated on the same 19 tasks that all three configurations successfully completed. For fairness in execution time comparison, all experiments were run on identical hardware with two H100 GPUs and one H200 GPU.

Our analysis considers two perspectives: (1) the trade-off between performance and efficiency, and (2) token-based resource usage as a proxy for GPU memory demand.

First, Table 5 compares the execution time, number of decision steps, and overall performance. At the same 70B model size, *ReAcTree+WM* requires more decision steps and a longer runtime but achieves a significantly higher GSR (+25%p) than *ReAct+WM*. This runtime difference stems from a greater number of decision steps rather than a higher computational cost per decision, representing a trade-off for increased robustness and success rate. Notably, *ReAcTree+WM* (8B) attains performance comparable to *ReAct+WM* (70B) while being significantly faster, highlighting its efficiency in low-resource settings.

Table 5: Average execution time, decision steps, and goal/-subgoal success rates (GSR/SSR) for three configurations on shared successful tasks.

Configuration	Time (s)	Decision Steps	GSR/SSR (%)
ReAct+WM (70B)	109.1	60.1	33 / 62.15
ReAcTree+WM (70B)	198.6	75.2	58 / 79.27
ReAcTree+WM (8B)	69.9	78.0	30 / 60.77

To measure computational resource requirements, we analyzed token usage. Specifically, the maximum number of input tokens serves as an indicator of peak GPU memory demand, while the average number of input/output tokens per decision step reflects the per-step computational load. The results are summarized in Table 6. The average token usage per decision is similar across all models, confirming that the tested configurations of *ReAcTree* and *ReAct* incur a comparable computational cost per step. However, *ReAct+WM* (70B) shows the highest peak input token count (8316) and the largest variance, suggesting greater GPU memory overhead and less predictable resource usage. In contrast, *ReAcTree* maintains well-bounded and stable token usage due to its modular structure, where each agent node processes a localized subgoal.

Table 6: Peak and average token usage per decision across three configurations.

Configuration	Max Input	Avg. Input	Avg. Output
ReAct+WM (70B)	8316	5359.45 (± 904.83)	16.07 (± 1.95)
ReAcTree+WM (70B)	6977	5362.66 (± 109.06)	17.83 (± 1.50)
ReAcTree+WM (8B)	7173	5390.12 (± 125.72)	17.68 (± 1.73)

5.6 Failure Case Analysis

We analyzed 39 failure cases of *ReAcTree+WM* with Qwen 2.5 72B in WAH-NL, categorizing them into four types: *Ambiguous* (10), *Execution* (12), *Search* (13), and *Expand* (4), as visualized in Figure 4.

Ambiguous cases arose from vague instructions (e.g., *give me 2 drinks*), creating uncertainty about object types or locations. *Execution* failures, largely due to LLM hallucination, included object confusion (*Confusion*, 8), irrelevant action repetition (*Repeat*, 2), and overlooked targets (*Skipping*, 2), reflecting challenges in reasoning consistency and object recognition. *Search* failures, the most frequent, resulted from exploration limitations under partial observability, including incomplete searches (*PlanFail*, 6), local search loops (*SameRm*, 3), and unnecessary revisits (*Revisit*, 2). Finally, *Expand* failures involved incorrect subgoal decomposition, with missing (*Missing*, 2) or incorrect (*Wrong*, 2) subgoals.

Search failures were dominant, underscoring the need for stronger fallback and exploration strategies. Addressing *Ambiguous* cases requires clarifying dialogues, while *Execution* errors highlight the importance of hallucination control. Less frequent *Expand* failures point to the need for subgoal refinement in complex scenarios.

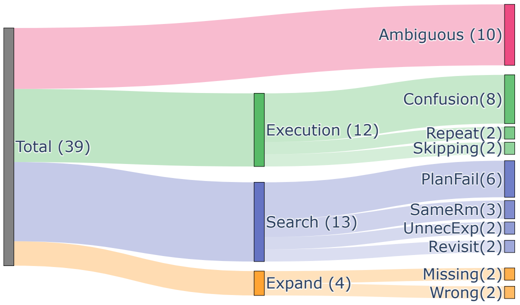


Figure 4: Categories and subcategories of failure cases for *ReAcTree+WM.*

6 CONCLUSION

We introduced *ReAcTree*, a hierarchical task planner that extends *ReAct* by dynamically constructing an agent tree in the subgoal space. Each agent node can reason, act, or expand the tree with new subgoals, while control flow nodes coordinate their execution through sequence, fallback, or parallel strategies. This design prevents the error accumulation typical of single-pass trajectories and makes long-horizon tasks more tractable. To strengthen in-context learning and coordination, we introduced episodic memory, which stores subgoal-level experiences for retrieval, and working memory, which shares environment-specific observations across nodes. Experiments on WAH-NL and ALFRED under partial observability show that *ReAcTree* consistently outperforms strong baselines across diverse LLMs, demonstrating improved reliability, generalization to unseen environments, and even enabling smaller models to rival larger ones.

Despite these benefits, *ReAcTree* still faces LLM-specific challenges, including hallucinations and limited capability to recognize failures. It also lacks mechanisms to revise incorrectly expanded subgoals and to handle instruction ambiguity. Future work will focus on mitigating hallucination, refining subgoal correction, and enabling clarification dialogues, ultimately advancing the robustness and deployability of *ReAcTree* in complex, real-world environments.

REFERENCES

[1] Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, et al. 2025. Phi-4-reasoning technical report. *arXiv preprint arXiv:2504.21318* (2025).

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*. https://openreview.net/forum?id=bdHkMjBJG_w

[4] Jicong Ao, Fan Wu, Yansong Wu, Abdalla Swiki, and Sami Haddadin. 2025. LLM-as-BT-Planner: Leveraging LLMs for Behavior Tree Generation in Robot Task Planning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1233–1239.

[5] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17682–17690.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[7] Boyuan Chen, Fei Xia, Brian Ichter, Kanishka Rao, Keerthana Gopalakrishnan, Michael S Ryoo, Austin Stone, and Daniel Kappler. 2023. Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11509–11522.

[8] Xinglin Chen, Yishuai Cai, Yunxin Mao, Minglong Li, Wenjing Yang, Weixia Xu, and Ji Wang. 2024. Integrating Intent Understanding and Optimal Behavior Planning for Behavior Tree Generation from Human Instructions. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, Kate Larson (Ed.). International Joint Conferences on Artificial Intelligence Organization, 6832–6840. https://doi.org/10.24963/ijcai.2024/755 Main Track.

[9] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching Large Language Models to Self-Debug. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=KuPixIqPiq

[10] Jae-Woo Choi, Youngwoo Yoon, Hyobin Ong, Jaehong Kim, and Minsu Jang. 2024. LoTa-Bench: Benchmarking Language-oriented Task Planners for Embodied Agents. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=ADSxCpCu9s

[11] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[12] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. 2018. Learning of behavior trees for autonomous agents. *IEEE Transactions on Games* 11, 2 (2018), 183–189.

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

[14] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[15] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in neural information processing systems* 32 (2019).

[16] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*. PMLR, 10764–10799.

[17] Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 8154–8173.

[18] Mengkang Hu, Yao Mu, Xinmiao Chelsey Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. 2024. Tree-Planner: Efficient Close-loop Task Planning with Large Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=Glcsog6zOe

[19] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*. PMLR, 9118–9147.

[20] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *6th Annual Conference on Robot Learning*. https://openreview.net/forum?id=3R3Pz5i0tye

[21] Alexandre Yukio Ichida, Felipe Meneguzzi, and Rafael C Cardoso. 2024. Bdi agents in natural language environments. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems.

[22] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[23] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[24] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474* (2017).

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

[26] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. https://doi.org/10.1162/tacl_a_00638

[27] Weiyu Liu, Geng Chen, Joy Hsu, Jiayuan Mao, and Jiajun Wu. 2024. Learning Planning Abstractions from Language. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=3UWuFoksGb

[28] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2024. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems* 36 (2024).

[29] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2024).

[30] Microsoft. 2023. Guidance. https://github.com/guidance-ai/guidance.

[31] Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2024. REFINER: Reasoning Feedback on Intermediate Representations. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1100–1126.

[32] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8494–8502.

[33] Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B. Tenenbaum, Sanja Fidler, and Antonio Torralba. 2021. Watch-And-Help: A Challenge for Social Perception and Human-AI Collaboration. In *International Conference on Learning Representations*.

[34] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

[35] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2024).

[36] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2024).

[37] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2024).

[38] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10740–10749.

[39] Yash Shukla, Wenchang Gao, Vasanth Sarathy, Alvaro Velasquez, Robert Wright, and Jivko Sinapov. 2024. LgTS: Dynamic Task Sampling using LLM-generated Sub-Goals for Reinforcement Learning Agents. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. 1736–1744.

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

[41] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied

agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2998–3009.

[42] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. AdaPlanner: adaptive planning from feedback with language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 58202–58245.

[43] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118* (2024).

[44] Huaxiaoyue Wang, Kushal Kedia, Juntao Ren, Rahma Abdullah, Atiksh Bhardwaj, Angela Chao, Kelly Y Chen, Nathaniel Chin, Prithwish Dan, Xinyi Fan, Gonzalo Gonzalez-Pumariega, Aditya Kompella, Maximus Adrian Pace, Yash Sharma, Xiangwan Sun, Neha Sunkara, and Sanjiban Choudhury. 2024. MOSAIC: Modular Foundation Models for Assistive and Interactive Cooking. In *8th Annual Conference on Robot Learning*. https://openreview.net/forum?id=dUo6j3YURS

[45] Peng-Yuan Wang, Jing-Cheng Pang, Chen-Yang Wang, Xuhui Liu, Tian-Shuo Liu, Si-Hang Yang, Hong Qian, and Yang Yu. 2025. InCLET: Large Language Model In-context Learning can Improve Embodied Instruction-following. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*. 2134–2142.

[46] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=1PL1NIMMrw

[47] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. 2023. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 34153–34189.

[48] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[49] Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating Sequences by Learning to Self-Correct. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=hH36JeQZDaO

[50] Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. 2023. Learning adaptive planning representations with natural language guidance. *arXiv preprint arXiv:2312.08566* (2023).

[51] Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. 2024. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems* 36 (2024).

[52] Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li F Fei-Fei. 2019. Regression planning networks. *Advances in neural information processing systems* 32 (2019).

[53] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024).

[54] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* 36 (2024).

[55] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

[56] Andy Zeng, Maria Attarian, brian ichter, Krzysztof Marcin Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael S Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. 2023. Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=G2Q2Mh3avow

[57] Zirui Zhao, Wee Sun Lee, and David Hsu. 2024. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems* 36 (2024).

[58] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models. In *Forty-first International Conference on Machine Learning*. https://openreview.net/forum?id=njwv9BsGHF

[59] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *The Eleventh International Conference on Learning Representations*.

[60] Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2024. ToolChain*: Efficient Action Space Navigation in Large Language Models with A* Search. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=B6pQxqUcT8

A REACTREE PSEUDOCODE

This section presents the pseudocode for the two core components of *ReAcTree*: agent node execution (Algorithm 1) and control flow node execution (Algorithm 2). The planning process begins with a single agent node n_0 assigned to the top-level goal g, which is executed through a call to ExecAgentNode(n_0, $D_{init} = 0$), where D_{init} denotes the initial decision count.

At each step, an agent node samples an action from the LLM, selecting from reasoning, acting, and expanding. When expanding, the agent node creates a control flow node as its child. This control flow node then instantiates agent nodes for the generated subgoals as its own children and executes them according to its behavior type, reporting success or failure back to its parent. Algorithms 1 and 2 summarize these procedures.

Algorithm 1 ExecAgentNode(n, D_{init}): Execution of an agent node

```
Input: Agent node $n$ with subgoal $g^n$, initial decision count $D_{\text{init}}$, maximum decision count $D_{\max}$
$D \leftarrow D_{\text{init}}$, $t \leftarrow 1$, $terminate \leftarrow$ False
$P^n_{\text{ic}} \leftarrow$ RetrieveFromEM($g^n$)        ▷ Retrieve in-context examples from episodic memory
$P^n \leftarrow (P_{\text{sys}}, P^n_{\text{ic}})$        ▷ Initialize LLM prompt
$c^n_t \leftarrow (o^n_t)$        ▷ Initialize context with current observation
while not terminate do
    $a^n_t \sim p_{\text{LLM}}(\cdot \mid P^n, g^n, c^n_t)$        ▷ Sample next action from LLM
    $D \leftarrow D + 1$
    $t \leftarrow t + 1$
    if $a^n_t = done$ then
        return Success, D
    else if $a^n_t = failure$ or $D \geq D_{\max}$ then
        return Failure, D
    else if $a^n_t \in \mathcal{A}^n_t$ then        ▷ Acting: execute an action and receive a textual observation
        $o^n_{t+1} \leftarrow$ Interact($a^n_t$)
        $c^n_{t+1} \leftarrow$ Append($c^n_t, a^n_t, o^n_{t+1}$)
    else if $a^n_t \in \mathcal{L}$ then        ▷ Reasoning: produce an internal thought step
        $o^n_{t+1} \leftarrow \emptyset$
        $c^n_{t+1} \leftarrow$ Append($c^n_t, a^n_t, o^n_{t+1}$)
    else if $a^n_t \in \mathcal{E}$ then        ▷ Expanding: generate control flow and subgoals
        $(f^n, [g^n_1, \ldots, g^n_K]) \leftarrow$ ParseExpand($a^n_t$)
        $n_f \leftarrow$ InitCtrlFlowNode($f^n$)
        AddChild($n$, $n_f$)
        for $i = 1$ to $K$ do
            $n_i \leftarrow$ InitAgentNode($g^n_i$)
            AddChild($n_f$, $n_i$)
        end for
        return ExecCtrlFlowNode($n_f$, D)        ▷ Execute control flow node (Alg. 2)
    end if
end while
```

Algorithm 2 ExecCtrlFlowNode(n_f, D_{init}): Execution of a control flow node

```
Input: Control flow node $n_f$ with type $f^n$ and child agent nodes $\{n_i\}$, initial decision count $D_{\text{init}}$
$D \leftarrow D_{\text{init}}$
if $f^n$ = sequence then
    for each child node $n_i$ in order do
        status, $D \leftarrow$ ExecAgentNode($n_i$, $D$)          ▷ Execute agent node (Alg. 1)
        if status = Failure then
            return Failure, D
        end if
    end for
    return Success, D
else if $f^n$ = fallback then
    for each child node $n_i$ in order do
        status, $D \leftarrow$ ExecAgentNode($n_i$, $D$)          ▷ Execute agent node (Alg. 1)
        if status = Success then
            return Success, D
        end if
    end for
    return Failure, D
else if $f^n$ = parallel then
    statuses ← [ ]
    for each child node $n_i$ in order do
        $status_i$, $D \leftarrow$ ExecAgentNode($n_i$, $D$)          ▷ Execute agent node (Alg. 1)
        statuses.append($status_i$)
    end for
    status ← AggregatePolicy(statuses)
    return status, D
end if
```

B IMPLEMENTATION DETAILS FOR EXPERIMENTAL SETUP

B.1 WAH-NL Test Set Corrections

We identified four WAH-NL test tasks where the natural language instructions were inconsistent with the goal conditions, such as referencing incorrect object classes or omitting required items. Therefore, to ensure reliable evaluation, we revised these instructions to better align with their intended goals. Table 7 presents the goal conditions, original instructions, and revised versions.

Table 7: Summary of corrections four WAH-NL test tasks were misaligned with their goal conditions.

Task ID	Goal Condition	Original Instruction	Revised Instruction
4	{‘on_juice_coffeetable’:1, ‘on_wine_coffeetable’:1}	Make sure there is a wine and a juice on the kitchen table.	Make sure there is a wine and a juice on the coffee table.
8	{‘on_wine_coffeetable’:1, ‘on_apple_coffeetable’:1}	Can you please put the apple and wine that are on the coffee table into the fridge please.	Can you please put the apple and the wine on the coffee table?
31	{‘inside_waterglass_dishwasher’:1, ‘inside_wineglass_dishwasher’:1, ‘inside_cutleryfork_dishwasher’:1, ‘turnOn_dishwasher’:1}	Please, put the cutlery fork and the wine glass in the dishwasher and then turn it on.	Please put the cutlery fork, the wine glass, and the water glass in the dishwasher and then turn it on.
94	{‘on_plate_kitchentable’:1, ‘on_waterglass_kitchentable’:1, ‘on_wineglass_kitchentable’:1, ‘on_cutleryfork_kitchentable’:1}	Please, put 1 wine glass, 1 water glass, and 1 plate on the table	Please put 1 wine glass, 1 water glass, 1 plate, and 1 cutlery fork on the table

B.2 Partially Observable Settings

We extended LoTa-Bench [10] by implementing a rule-based observation generator for both VirtualHome and AI2THOR to support the partially observable setting. After each action, the agent receives a textual description of visible objects and receptacles in its current room, excluding items hidden inside closed containers. These observations are derived from the simulator's ground-truth state and include class and instance identifiers to enable precise object grounding.

VirtualHome. In VirtualHome, we apply the partially observable protocol, which supports six natural language-based primitive actions (*go to*, *pick up*, *put down*, *open*, *close*, and *turn on*). To capture the variability in observations, we define eight action-dependent observation rules, each specifying how textual feedback is generated based on the semantics of the executed action. Table 8 presents representative action-observation pairs illustrating these rules.

AI2THOR. We apply the same protocol to AI2THOR, which supports eight primitive actions: *go to*, *pick up*, *put down*, *slice*, *open*, *close*, *turn on*, and *turn off*. Table 9 presents representative action-observation pairs in AI2THOR, illustrating how textual feedback differs depending on the executed action.

Table 8: Representative action-observation pairs in VirtualHome under the partially observable setting.

Action Context	Action	Text Observation
initialization	-	You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the middle of a bathroom (1). Looking quickly around the room, you see bathroom cabinet (1), bathroom counter (1), faucet (1), sink (1), toilet (1), towel rack (1), washing machine (1).
go to room	go to kitchen 1	You move to the kitchen (1). Looking quickly around the room, you see chair (1, 2, 3, 4), dishwasher (1), faucet (1), fridge (1), kitchen cabinet (1, 2, 3, 4, 5, 6, 7, 8), kitchen counter (1, 3), kitchen table (1), microwave oven (1), nightstand (1, 2), sink (1), sofa (1), stove (1), toaster (1).
go to object or receptacle	go to kitchen table 1	You arrive at the kitchen table (1). You see apple (2), book (1), bowl (2, 3), cereal (1), chair (1, 2, 3, 4), chocolate syrup (1), condiment bottle (1, 2), condiment shaker (1, 2), cupcake (1, 2), cutlery fork (2, 3, 4), cutlery knife (3, 5, 6), juice (1), kitchen table (1), pancake (1), pear (1), plate (2, 3, 4, 5), pudding (2), water glass (1, 2, 4)
pick up object	pick up juice 1	You pick up juice. You hold juice (1).
put down object	put down apple 1	You put down apple on coffee table.
open receptacle	open fridge 1	You open fridge. You see bench (1), cupcake (1), fridge (1, 2), juice (1).
close receptacle	close dishwasher 1	You close dishwasher.
turn on receptacle	turn on dishwasher 1	You turn on dishwasher.

Table 9: Representative action-observation pairs in AI2THOR under the partially observable setting.

Action Context	Action	Text Observation
initialization	-	You are in the house, and you arrive at the bathroom, Looking quickly around the room, you see TowelHolder (1), Toilet (1), ToiletPaperHanger (1), Sink (1), SinkBasin (1), Shelf (1, 2, 3), HandTowelHolder (1), GarbageCan (1), BathtubBasin (1).
go to object or receptacle	go to BathtubBasin (1)	You arrive at BathtubBasin (1). You see SoapBar (2), Cloth (1), Bathtub (1), BathtubBasin (1).
pick up object	pick up SoapBar (2)	You pick up SoapBar (2). You see SoapBar (2), Cloth (1), Bathtub (1), BathtubBasin (1). I am holding SoapBar (2) now.
put down object	put down SoapBar (2)	You put SoapBar (2) on Shelf (1). You see SoapBar (1, 2, 3), Shelf (1), Candle (2).
slice object	slice Potato (2)	You slice Potato (2). You see Potato (8, 9, 10, 11, 12), Knife (1), Fridge (1). Potato (5, 6, 7, 8, 9, 10, 11, 12) are sliced. I am holding Knife (1) now.
open receptacle	open Microwave (1)	You open Microwave (1). You see StoveKnob (1, 2, 3, 4), StoveBurner (3), Potato (1), Pan (1, 2), Microwave (1), Bowl (3).
close receptacle	close Microwave (1)	You close Microwave (1). You see StoveKnob (1, 2, 3, 4), StoveBurner (3), Pan (1, 2), Microwave (1).
turn on receptacle	turn on Microwave (1)	You turn on Microwave (1). You see Microwave (1), Drawer (5, 6), Cabinet (7, 8, 10).
turn off receptacle	turn off Microwave (1)	You turn off Microwave (1). You see Microwave (1), Drawer (5, 6), Cabinet (7, 8, 10).

C IMPLEMENTATION DETAILS OF BASELINES

All baseline methods are evaluated under the same few-shot in-context learning setup as *ReAcTree*, without fine-tuning. They also share identical retrieval size and decision cap: 5K tokens and 200 steps for WAH-NL, and 5K tokens and 100 steps for ALFRED. Details of episodic memory construction, generation strategies, and other implementation aspects are described for each baseline below.

***ZSP* [19].** For WAH-NL, *ZSP* constructs episodic memory using a fully observable, rule-based planner to extract 183 ground-truth trajectories from the training set. At test time, it retrieves in-context examples most similar to the current task goal using Sentence-BERT [34], following the shared 5K-token retrieval setup. The LLM performs free-form generation with temperature 0.0 to produce an acting action, which is then mapped to an executable skill via Sentence-BERT-based matching. *ZSP* generates its full plan recursively without access to intermediate observations, relying solely on the initial environment description. To compensate, we provide global environment information including all rooms, objects, and receptacles at the beginning of each task. Since object perception is unavailable at each step, we relax action constraints by omitting instance identifiers for *pick up* and *put down* actions, for example using *pick up apple* instead of *pick up apple 1*.

***Tree-Planner* [18].** For WAH-NL, *Tree-Planner* follows a two-step procedure consisting of plan sampling and grounded deciding. During plan sampling, it uses the same episodic memory as *ZSP* (183 rule-based trajectories), retrieves up to 5K tokens of in-context examples, and generates N candidate plans ($N = 25$ or 50) with temperature 0.8 and top-$p = 0.95$, following the original paper. Similar to *ZSP*, it also receives global environment information and generates plans without instance-level identifiers for *pick up* and *put down* actions, resulting in an action tree. In the grounded deciding phase, the model receives step-wise observations and selects an action from the pre-generated tree. This decision process uses a fixed set of in-context examples and leverages the Guidance library [30] to perform constrained generation.

***ReAct* [55] & *ReAct+WM*.** Both methods follow the same implementation settings as *ReAcTree* in terms of episodic memory construction (manual collection followed by LLM-based bootstrapping), retrieval size and decision cap (5K and 200 for WAH-NL; 5K and 100 for ALFRED), and generation settings using the Guidance library [30] with temperature 0.0 for reasoning and constrained action selection.

D LANGUAGE MODELS

Table 10 lists the specific language models used in our experiments. Model names follow their identifiers in the HuggingFace model hub.

Table 10: Language models used in our experiments.

Class	Model name	Model size
LLaMA 3.1	meta-llama/Llama-3.1-8B	8B
	meta-llama/Llama-3.1-70B	70B
Qwen 2.5	Qwen/Qwen2.5-7B	7B
	Qwen/Qwen2.5-72B	72B
Mistral	mistralai/Mistral-7B-v0.3	7B
Gemma 2	google/gemma-2-9b	9B
Phi-4-reasonin-plus	microsoft/Phi-4-reasoning-plus	14B

E PROMPTS TEMPLATES

E.1 *ReAcTree* & *ReAcTree+WM*

We provide the prompt template of *ReAcTree+WM* for WAH-NL and ALFRED in Listings 1 and 2, respectively. The only difference between *ReAcTree+WM* and *ReAcTree* lies in the omission of the "recall location of" action, while otherwise sharing the same structure. The "..." symbol denotes the position where in-context examples retrieved from episodic memory are inserted.

Listing 1: Prompt template of *ReAcTree+WM* for WAH-NL

```
You are an advanced robot with ability to think, act, and expand behavior tree nodes in decision-making process. You can
     perform one of the following tasks:
1. Think: Use reasoning to satisfy the current goal condition.
2. Act: Execute a specific action to accomplish the current goal condition. You should use one of actions of this list:
     [go to, pick up, put down, open, close, turn on, recall location of, done, failure]
3. Expand: Decompose the current goal condition into more detailed subgoals. When expanding, generate appropriate control
     flow and subgoals. Control flow can be "sequence" (achieve subgoals sequentially; if any subgoal fails, the sequence
     is interrupted), "fallback" (attempt subgoals in order until one succeeds; if a subgoal is successful, the remaining
     subgoals are not attempted), or "parallel" (achieve subgoals in parallel; this enables tasks to continue
     independently, even if one subgoal fails).
```

```
Source domain:
...

Target_domain:
Your task is to: Put one cupcake and one apple on the coffee table
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a bedroom (1). Looking quickly around the room, you see bed (1), bookshelf (1), cabinet (1), chair (5),
     desk (1), nightstand (3).
```

Listing 2: Prompt template of *ReAcTree+WM* for ALFRED

```
You are an advanced robot with ability to think, act, and expand behavior tree nodes in decision-making process. You can
     perform one of the following tasks:
1. Think: Use reasoning to satisfy the current goal condition.
2. Act: Execute a specific action to accomplish the current goal condition. You should use one of actions of this list:
     [go to, pick up, put down, slice, open, close, turn on, turn off, done, failure]
3. Expand: Decompose the current goal condition into more detailed subgoals. When expanding, generate appropriate control
     flow and subgoals. Control flow can be "sequence" (achieve subgoals sequentially. If any subgoal fails, the sequence
     is interrupted) or "fallback" (Attempt subgoals in order until one succeeds. If a subgoal is successful, the
     remaining subgoals are not attempted).

Source domain:
...

Target domain:
Your task is to: Hold the clock and turn on the lamp.
You are in the house, and you arrive at the bedroom, Looking quickly around the room, you see Shelf (1, 2, 3, 4, 5, 6),
     Safe (1), LaundryHamper (1), GarbageCan (1), Drawer (1, 2, 3, 4, 5, 6), Desk (1, 2), Bed (1).
```

E.2 Baselines

We provide the prompt templates used for the baseline methods *ZSP*, *Tree-Planner*, and *ReAct+WM* across both WAH-NL and ALFRED.

ZSP. *ZSP* uses the prompt template in Listing 3. The "..." portion is replaced by in-context examples relevant to the current task goal.

Tree-Planner. During its *plan sampling* phase, *Tree-Planner* employs the same template as *ZSP* (Listing 3). For *grounded deciding*, it uses a separate template in Listing 4.

ReAct & ReAct + WM. We provide the prompt templates of *ReAct+WM* for WAH-NL and ALFRED in Listing 5 and 6, respectively. The only difference from *ReAct* is the inclusion of the action "*recall location of*" action from the prompt, while otherwise sharing the same structure. The "..." portion in the template indicates where in-context examples retrieved from episodic memory are inserted.

Listing 3: Prompt template of *ZSP* for WAH-NL

```
You are an advanced robot with ability to generate action plans. You can perform one of the following actions of this
     list: [go to, pick up, put down, open, close, turn on, done]

Source domain:
...

Target_domain:
```

```
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). In the kitchen 1,
     there are chair (1, 2, 3, 4), dishwasher (1), faucet (1), fridge (1), kitchen cabinet (1, 2, 3, 4, 5, 6, 7, 8),
     kitchen counter (1, 3), kitchen table (1), microwave oven (1), nightstand (1, 2), sink (1), sofa (1), stove (1),
     toaster (1). In the bathroom 1, there are bathroom cabinet (1), bathroom counter (1), faucet (2), sink (2), toilet
     (1), washing machine (1). In the bedroom 1, there are bed (1), bookshelf (1), cabinet (1), chair (5), desk (1),
     nightstand (3). In the living room 1, there are bookshelf (2, 3), chair (6), closet (1), coffee table (1), computer
     (1), desk (2), nightstand (4), sofa (2, 3), tv (1). The objects in the house are apple (1, 2, 3), bananas (1), bar
     soap (1), book (1, 2), bowl (1, 2, 3, 4, 5, 6, 7, 8, 9), box (1, 2, 3, 4, 5), bucket (1), candle (1), candy bar (1),
     cell phone (1, 2), cereal (1), chair (1, 2, 3, 4, 5, 6), chips (1, 2), chocolate syrup (1), coffee pot (1),
     condiment bottle (1, 2, 3, 4), condiment shaker (1, 2, 3, 4), cooking pot (1, 2), crackers (1, 2), creamy buns (1),
     cupcake (1, 2, 3), cutlery fork (1, 2, 3, 4), cutlery knife (1, 2, 3, 4, 5, 6), cutlets (1), dishwashing liquid (1),
     face cream (1, 2, 3), folder (1, 2, 3, 4), frying pan (1), hair product (1, 2, 3, 4), hanger (1, 2, 3, 4, 5, 6, 7),
     juice (1, 2), keyboard (1), lime (1), milk (1), mouse (1), mug (1, 2, 3), notes (1), oven tray (1), pancake (1),
     pear (1), pile of clothes (1, 2), pillow (1, 2, 3, 4, 5, 6), plate (1, 2, 3, 4, 5, 6, 7), plum (1, 2), pudding (1,
     2), radio (1), rug (1, 2, 3, 4), slice of bread (1, 2), toilet paper (1), toothbrush (1), toothpaste (1), wall phone
     (1), wall picture frame (1, 2, 3, 4, 5, 6, 7, 8), washing sponge (1), water glass (1, 2, 3, 4, 5), wine (1). You are
     in the middle of a bedroom (1). Looking quickly around the room, you see bed (1), bookshelf (1), cabinet (1), chair
     (5), desk (1), nightstand (3).
Your task is to: Put one cupcake and one apple on the coffee table
```

Listing 4: Prompt template of *Tree-Planner* grounded deciding for WAH-NL

```
You need to act as a home robot. At each moment, I will provide you with observations of your current environment, as
     well as the high-level task I want you to do, and previous mid-level sub-tasks that have been executed. Then, you
     need to select the best sub-task from the options I provide to complete the designated home task based on the
     observation and your past experience. When one choosed sub-task causes an error in the environment, you will be
     provided with the error information and the corresponding sub-task, and you need to re-choose a corrective sub-task
     at the current time step. For example, the actions that have been executed in the environment are:
go to kitchen 1
go to kitchen table 1
The choosed action is: pick up cutlery fork 1
The prompt (error information) would be: The action: "pick up cutlery fork 1" caused an error: Action is not executable,
     since the agent is not close to cutlery fork 1 when executing "pick up cutlery fork 1" Among the following actions,
     wich action would you take.
go to dishwasher 1
go to fridge 1
A corrective choice of action would be: go to dishwasher 1

Currently, you are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are
     in the middle of a bedroom (1). Looking quickly around the room, you see bed (1), bookshelf (1), cabinet (1), chair
     (5), desk (1), nightstand (3). You are close to cabinet (1), cell phone (2), desk (1), plate (7). You hold nothing
     in your hands.
Your task is to: Put one cupcake and one apple on the coffee table
```

Listing 5: Prompt template of *ReAct+WM* for WAH-NL

```
You are an advanced robot with ability to think and act. You can perform one of the following tasks:
1. Think: Use reasoning to satisfy the current goal condition.
2. Act: Execute a specific action to accomplish the current goal condition. You should use one of actions of this list:
     [go to, pick up, put down, open, close, turn on, recall location of, done, failure]
Source domain:
...

Target_domain:
Your task is to: Put one cupcake and one apple on the coffee table
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a bedroom (1). Looking quickly around the room, you see bed (1), bookshelf (1), cabinet (1), chair (5),
     desk (1), nightstand (3).
```

Listing 6: Prompt template of ReAct+WM for ALFRED

```
You are an advanced robot with ability to think and act. You can perform one of the following tasks:
1. Think: Use reasoning to satisfy the current goal condition.
2. Act: Execute a specific action to accomplish the current goal condition. You should use one of actions of this list:
     [go to, pick up, put down, slice, open, close, turn on, turn off, done, failure]

Source domain:
...

Target domain:
Your task is to: Hold the clock and turn on the lamp.
You are in the house, and you arrive at the bedroom, Looking quickly around the room, you see Shelf (1, 2, 3, 4, 5, 6),
     Safe (1), LaundryHamper (1), GarbageCan (1), Drawer (1, 2, 3, 4, 5, 6), Desk (1, 2), Bed (1).
```

F FAILURE CASE OF *REACT+WM*

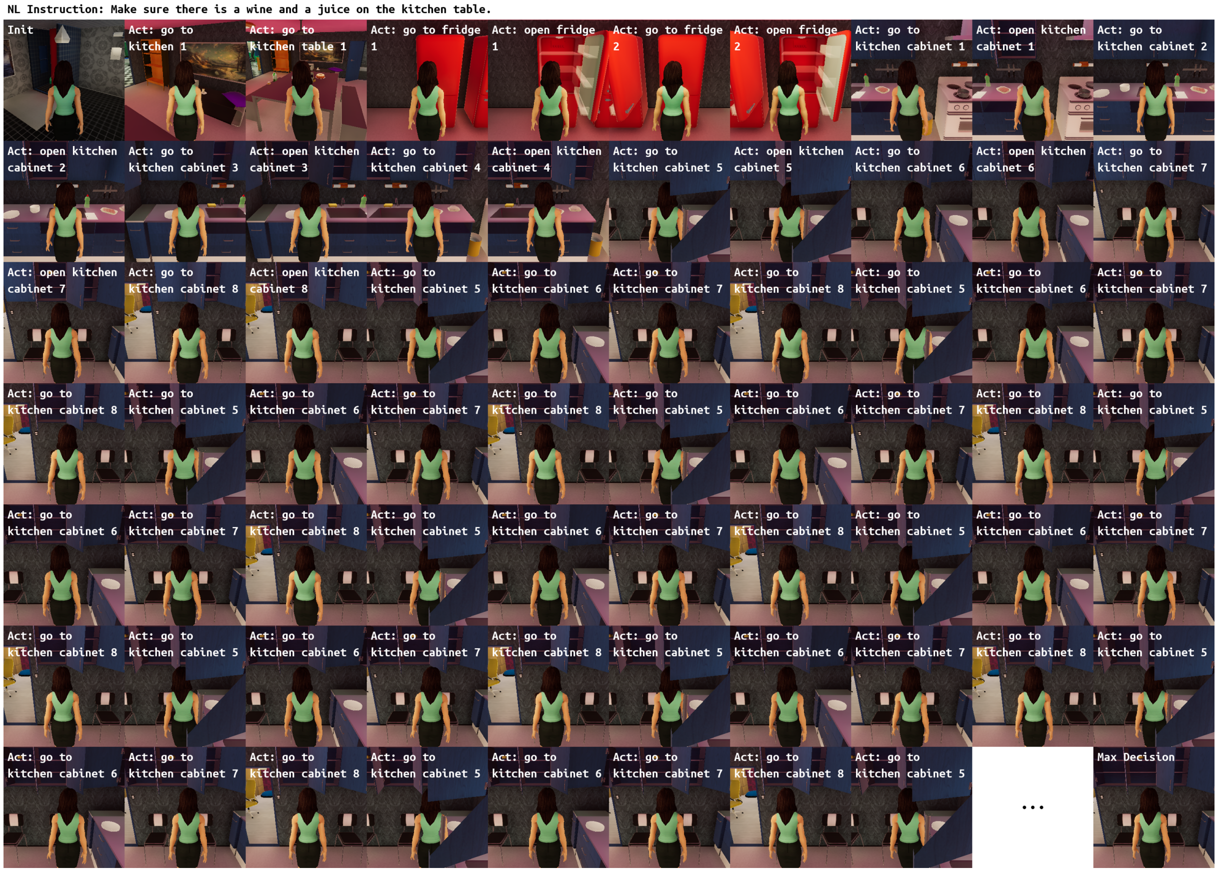

NL Instruction: Make sure there is a wine and a juice on the kitchen table.
Init
Act: go to kitchen 1
Act: go to kitchen table 1
Act: go to fridge 1
Act: open fridge 1
Act: go to fridge 2
Act: open fridge 2
Act: go to kitchen cabinet 1
Act: open kitchen cabinet 1
Act: go to kitchen cabinet 2
Act: open kitchen cabinet 2
Act: go to kitchen cabinet 3
Act: open kitchen cabinet 3
Act: go to kitchen cabinet 4
Act: open kitchen cabinet 4
Act: go to kitchen cabinet 5
Act: open kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: open kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: open kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: open kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
Act: go to kitchen cabinet 6
Act: go to kitchen cabinet 7
Act: go to kitchen cabinet 8
Act: go to kitchen cabinet 5
...
Max Decision

Figure 5: Failure case of *ReAct+WM* on the WAH-NL dataset using Qwen 2.5 72B.

G *REACTREE+WM* FULL TRAJECTORY OF SUCCESS CASE

Listing 7: Full Trajectory of ReAcTree+WM

```
### Agent Node 0
Your task is to: Make sure there is a wine and a juice on the coffee table.
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a bathroom (1). Looking quickly around the room, you see bathroom cabinet (1), bathroom counter (1),
     faucet (1), sink (1), toilet (1), towel rack (1), washing machine (1).
Think: To solve the task, I need to accomplish 2 subgoals independently. Move the wine on the coffee table, and move the
     juice on the coffee table.
Expand: {'control_flow': 'parallel', 'conditions': 'move the wine on the coffee table, and move the juice on the coffee
     table'}

### Agent 1
Your primary goal is to: Make sure there is a wine and a juice on the coffee table.
To achieve this, you should perform your sibling tasks in parallel. At this level, your sibling tasks are: move the wine
     on the coffee table, and and move the juice on the coffee table.
Your task is to: move the wine on the coffee table
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a bathroom (1). Looking quickly around the room, you see bathroom cabinet (1), bathroom counter (1),
     faucet (1), sink (1), toilet (1), towel rack (1), washing machine (1).
Think: To solve the task, I need to accomplish 2 subgoals sequentially. Find and pick up the wine, and find the coffee
     table and put down the wine.
Expand: {'control_flow': 'sequence', 'conditions': 'find and pick up the wine, find the coffee table and put down the
     wine'}

### Agent Node 3
Your primary goal is to: move the wine on the coffee table
To achieve this, you should perform your sibling tasks in sequence. At this level, your sibling tasks are: find and pick
     up the wine, and find the coffee table and put down the wine.
Your task is to: find and pick up the wine
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a bathroom (1). Looking quickly around the room, you see bathroom cabinet (1), bathroom counter (1),
     faucet (1), sink (1), toilet (1), towel rack (1), washing machine (1).
Think: To solve the task, I need to recall the location of the wine.
Act: recall location of wine
You have not seen wine before.
Think: Now, I need to find and pick up the wine in one of the rooms. The wine is more likely to appear in kitchen 1,
     living room 1, or bedroom 1. I can check each room one by one until I find and pick up the wine.
Expand: {'control_flow': 'fallback', 'conditions': 'find and pick up the wine in kitchen 1, find and pick up the wine in
     living room 1, find and pick up the wine in bedroom 1'}

### Agent Node 5
Your primary goal is to: find and pick up the wine
To achieve this, you should perform your sibling tasks using a fallback strategy. At this level, your sibling tasks are:
     find and pick up the wine in kitchen 1, find and pick up the wine in living room 1, and find and pick up the wine in
     bedroom 1.
Your task is to: find and pick up the wine in kitchen 1
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a bathroom (1). Looking quickly around the room, you see bathroom cabinet (1), bathroom counter (1),
     faucet (1), sink (1), toilet (1), towel rack (1), washing machine (1).
Think: I am in bathroom 1. Now, I need to go to kitchen 1.
Act: go to kitchen 1
You move to the kitchen (1). Looking quickly around the room, you see bench (1, 2), bookshelf (1, 2), chair (1, 2),
     dishwasher (1), faucet (2), fridge (1, 2), garbage can (1), kitchen cabinet (1, 2, 3, 4, 5, 6, 7, 8), kitchen
     counter (1, 2), kitchen table (1), microwave oven (1), sink (2), stove (1), toaster (1).
Think: Now, I need to find the wine. The wine is more likely to appear on kitchen table (1), or in fridge (1, 2), kitchen
     cabinet (1, 2, 3, 4, 5, 6, 7, 8). I can check each place one by one, starting from kitchen table 1, until I find the
     wine.
Act: go to kitchen table 1
```

```
You arrive at the kitchen table (1). You see bench (2), book (1), bowl (6), cereal (1), kitchen table (1), milk (1),
     pudding (1, 2)
Act: go to fridge 1
You arrive at the fridge (1). The fridge (1) is closed. You see bench (1), cupcake (1), fridge (1)
Act: open fridge 1
You open fridge. You see bench (1), cupcake (1), fridge (1)
Act: go to fridge 2
You arrive at the fridge (2). The fridge (2) is closed. You see bench (1), cupcake (1), fridge (1, 2), juice (1)
Act: open fridge 2
You open fridge. You see bench (1), cupcake (1), fridge (1, 2), juice (1)
Act: go to kitchen cabinet 1
You arrive at the kitchen cabinet (1). The kitchen cabinet (1) is closed. You see chicken (1), cooking pot (1), cutlery
     knife (1, 2), cutlets (1), dishwashing liquid (1), faucet (2), frying pan (1), garbage can (1), kitchen cabinet (1,
     4), kitchen counter (1), sink (2), stove (1)
Act: open kitchen cabinet 1
You open kitchen cabinet. You see chicken (1), cooking pot (1), cutlery knife (1, 2), cutlets (1), dishwashing liquid
     (1), faucet (2), frying pan (1), garbage can (1), kitchen cabinet (1, 4), kitchen counter (1), sink (2), stove (1)
Act: go to kitchen cabinet 2
You arrive at the kitchen cabinet (2). The kitchen cabinet (2) is closed. You see bowl (1, 2, 3, 4, 5), chair (2),
     chicken (1), cutlery knife (1, 2), cutlets (1), dishwashing liquid (1), faucet (2), kitchen cabinet (1, 2, 3, 4, 5,
     6), kitchen counter (1), plate (1, 2, 3, 4), sink (2), washing sponge (1)
Act: open kitchen cabinet 2
You open kitchen cabinet. You see bowl (1, 2, 3, 4, 5), chair (2), chicken (1), cutlery knife (1, 2), cutlets (1),
     dishwashing liquid (1), faucet (2), kitchen cabinet (1, 2, 3, 4, 5, 6), kitchen counter (1), plate (1, 2, 3, 4),
     sink (2), washing sponge (1)
Act: go to kitchen cabinet 3
You arrive at the kitchen cabinet (3). The kitchen cabinet (3) is closed. You see bowl (1, 2, 3, 4, 5), chair (1, 2),
     dishwashing liquid (1), faucet (2), kitchen cabinet (2, 3, 4, 5, 6, 7), kitchen counter (1, 2), plate (1, 2, 3, 4),
     sink (2), washing sponge (1)
Act: open kitchen cabinet 3
You open kitchen cabinet. You see bowl (1, 2, 3, 4, 5), chair (1, 2), dishwashing liquid (1), faucet (2), kitchen cabinet
     (2, 3, 4, 5, 6, 7), kitchen counter (1, 2), plate (1, 2, 3, 4), sink (2), washing sponge (1)
Act: go to kitchen cabinet 4
You arrive at the kitchen cabinet (4). The kitchen cabinet (4) is closed. You see bowl (1, 2, 3, 4, 5), chicken (1),
     cooking pot (1), cutlery knife (1, 2), cutlets (1), dishwashing liquid (1), faucet (2), garbage can (1), kitchen
     cabinet (1, 2, 4), kitchen counter (1), sink (2), washing sponge (1)
Act: open kitchen cabinet 4
You open kitchen cabinet. You see bowl (1, 2, 3, 4, 5), chicken (1), cooking pot (1), cutlery knife (1, 2), cutlets (1),
     dishwashing liquid (1), faucet (2), garbage can (1), kitchen cabinet (1, 2, 4), kitchen counter (1), sink (2),
     washing sponge (1)
Act: go to kitchen cabinet 5
You arrive at the kitchen cabinet (5). The kitchen cabinet (5) is closed. You see bowl (1, 2, 3, 4, 5), chair (1, 2),
     cutlets (1), dishwashing liquid (1), faucet (2), kitchen cabinet (2, 3, 4, 5, 6, 7), kitchen counter (1, 2), plate
     (1, 2, 3, 4), sink (2), washing sponge (1)
Act: open kitchen cabinet 5
You open kitchen cabinet. You see bowl (1, 2, 3, 4, 5), chair (1, 2), cutlets (1), dishwashing liquid (1), faucet (2),
     kitchen cabinet (2, 3, 4, 5, 6, 7), kitchen counter (1, 2), plate (1, 2, 3, 4), sink (2), washing sponge (1)
Act: go to kitchen cabinet 6
You arrive at the kitchen cabinet (6). The kitchen cabinet (6) is closed. You see bowl (1, 2, 3, 4, 5), chair (1, 2),
     dishwashing liquid (1), faucet (2), kitchen cabinet (2, 3, 4, 5, 6, 7, 8), kitchen counter (1, 2), plate (1, 2, 3,
     4), sink (2), washing sponge (1)
Act: open kitchen cabinet 6
You open kitchen cabinet. You see bowl (1, 2, 3, 4, 5), chair (1, 2), dishwashing liquid (1), faucet (2), kitchen cabinet
     (2, 3, 4, 5, 6, 7, 8), kitchen counter (1, 2), plate (1, 2, 3, 4), sink (2), washing sponge (1)
Act: go to kitchen cabinet 7
You arrive at the kitchen cabinet (7). The kitchen cabinet (7) is closed. You see bowl (1, 2, 3, 4, 5), chair (1, 2),
     kitchen cabinet (2, 3, 4, 5, 6, 7, 8), kitchen counter (1), light switch (2), plate (1, 2, 3, 4), sink (2)
Act: open kitchen cabinet 7
You open kitchen cabinet. You see bowl (1, 2, 3, 4, 5), chair (1, 2), kitchen cabinet (2, 3, 4, 5, 6, 7, 8), kitchen
     counter (1), light switch (2), plate (1, 2, 3, 4), pudding (3), sink (2)
Act: go to kitchen cabinet 8
```

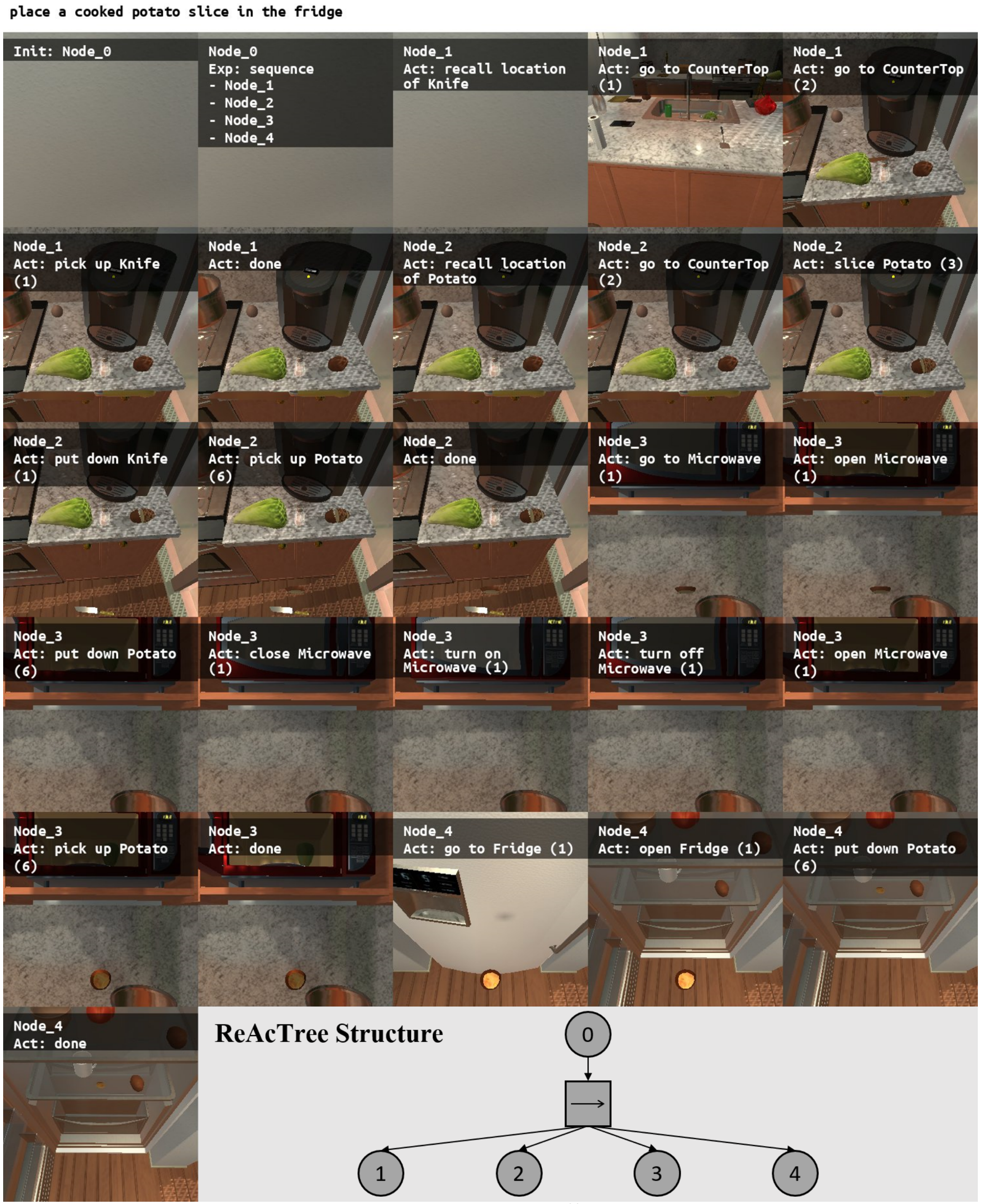
```
You arrive at the kitchen cabinet (8). The kitchen cabinet (8) is closed. You see bench (2), chair (1, 2), kitchen cabinet (5, 6, 7, 8), light switch (2), pudding (3)
Act: open kitchen cabinet 8
You open kitchen cabinet. You see bench (2), chair (1, 2), kitchen cabinet (5, 6, 7, 8), light switch (2), pudding (3)
Think: I cannot find the wine in kitchen 1.
Act: failure

### Agent Node 6
Your primary goal is to: find and pick up the wine
To achieve this, you should perform your sibling tasks using a fallback strategy. At this level, your sibling tasks are: find and pick up the wine in kitchen 1, find and pick up the wine in living room 1, and find and pick up the wine in bedroom 1.
Your task is to: find and pick up the wine in living room 1
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the middle of a kitchen (1). Looking quickly around the room, you see bench (1, 2), bookshelf (1, 2), chair (1, 2), dishwasher (1), faucet (2), fridge (1, 2), garbage can (1), kitchen cabinet (1, 2, 3, 4, 5, 6, 7, 8), kitchen counter (1, 2), kitchen table (1), microwave oven (1), sink (2), stove (1), toaster (1).
Think: I am in kitchen 1. Now, I need to go to living room 1.
Act: go to living room 1
You move to the living room (1). Looking quickly around the room, you see bookshelf (3), chair (3), closet (1), coffee table (1), computer (1), desk (1), garbage can (2), nightstand (1), sofa (1), tv (1).
Think: Now, I need to find the wine. The wine is more likely to appear on coffee table (1), desk (1), or in closet (1), nightstand (1). I can check each place one by one, starting from coffee table 1, until I find the wine.
Act: go to coffee table 1
You arrive at the coffee table (1). You see alcohol (2), bananas (1), bowl (7), cell phone (1, 2), coffee table (1), lime (1), mug (1), plum (1, 2), sofa (1)
Act: go to desk 1
You arrive at the desk (1). You see chair (3), computer (1), desk (1), nightstand (1), water glass (2)
Act: go to closet 1
You arrive at the closet (1). The closet (1) is closed. You see closet (1), light switch (3)
Act: open closet 1
You open closet. You see closet (1), light switch (3)
Act: go to nightstand 1
You arrive at the nightstand (1). The nightstand (1) is closed. You see box (2, 5, 7), chair (3), computer (1), desk (1), nightstand (1), water glass (2)
Act: open nightstand 1
You open nightstand. You see box (2, 5, 7), chair (3), computer (1), desk (1), nightstand (1), water glass (2)
Think: I cannot find the wine in living room 1.
Act: failure

### Agent Node 7
Your primary goal is to: find and pick up the wine
To achieve this, you should perform your sibling tasks using a fallback strategy. At this level, your sibling tasks are: find and pick up the wine in kitchen 1, find and pick up the wine in living room 1, and find and pick up the wine in bedroom 1.
Your task is to: find and pick up the wine in bedroom 1
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the middle of a living room (1). Looking quickly around the room, you see bookshelf (3), chair (3), closet (1), coffee table (1), computer (1), desk (1), garbage can (2), nightstand (1), sofa (1), tv (1).
Think: I am in living room 1. Now, I need to go to bedroom 1.
Act: go to bedroom 1
You move to the bedroom (1). Looking quickly around the room, you see bed (1), bookshelf (4), cabinet (1), chair (4), closet (2), coffee table (2), computer (2), desk (2, 3), garbage can (3).
Think: Now, I need to find the wine. The wine is more likely to appear on coffee table (2), or in cabinet (1), closet (2). I can check each place one by one, starting from coffee table 2, until I find the wine.
Act: go to coffee table 2
You arrive at the coffee table (2). You see coffee table (2), magazine (1)
Act: go to cabinet 1
You arrive at the cabinet (1). The cabinet (1) is closed. You see cabinet (1), computer (2), desk (2), light switch (4), sundae (1)
Act: open cabinet 1
You open cabinet. You see cabinet (1), computer (2), desk (2), light switch (4), sundae (1), wine (1)
```

```
Think: I find wine 1. Now I need to pick it up.
Act: pick up wine 1
You pick up wine. You hold wine (1).
Think: Now I accomplish the goal.
Act: done

### Agent Node 4
Your primary goal is to: move the wine on the coffee table
To achieve this, you should perform your sibling tasks in sequence. At this level, your sibling tasks are: find and pick
     up the wine, and find the coffee table and put down the wine.
Your task is to: find the coffee table and put down the wine
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a bedroom (1). Looking quickly around the room, you see bed (1), bookshelf (4), cabinet (1), chair (4),
     closet (2), coffee table (2), computer (2), desk (2, 3), garbage can (3). You hold wine (1).
Think: I hold wine 1. Now, I need to find the coffee table. The coffee table is more likely to appear in living room 1.
Act: go to living room 1
You move to the living room (1). Looking quickly around the room, you see bookshelf (3), chair (3), closet (1), coffee
     table (1), computer (1), desk (1), garbage can (2), nightstand (1), sofa (1), tv (1). You hold wine (1).
Think: I can see coffee table 1.
Act: go to coffee table 1
You arrive at the coffee table (1). You see alcohol (2), bananas (1), bowl (7), cell phone (1, 2), coffee table (1), lime
     (1), mug (1), plum (1, 2), sofa (1), wine (1) You hold wine (1).
Think: Now, I need to put down wine 1.
Act: put down wine 1
You put down wine on coffee table
Think: Now I accomplish the goal.
Act: done

### Agent Node 2
Your primary goal is to: Make sure there is a wine and a juice on the coffee table.
To achieve this, you should perform your sibling tasks in parallel. At this level, your sibling tasks are: move the wine
     on the coffee table, and and move the juice on the coffee table.
Your task is to: and move the juice on the coffee table
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a living room (1). Looking quickly around the room, you see bookshelf (3), chair (3), closet (1), coffee
     table (1), computer (1), desk (1), garbage can (2), nightstand (1), sofa (1), tv (1).
Think: To solve the task, I need to accomplish 2 subgoals sequentially. Find and pick up the juice, and find the coffee
     table and put down the juice.
Expand: {'control_flow': 'sequence', 'conditions': 'find and pick up the juice, find the coffee table and put down the
     juice'}

### Agent Node 8
Your primary goal is to: and move the juice on the coffee table
To achieve this, you should perform your sibling tasks in sequence. At this level, your sibling tasks are: find and pick
     up the juice, and find the coffee table and put down the juice.
Your task is to: find and pick up the juice
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the
     middle of a living room (1). Looking quickly around the room, you see bookshelf (3), chair (3), closet (1), coffee
     table (1), computer (1), desk (1), garbage can (2), nightstand (1), sofa (1), tv (1).
Think: To solve the task, I need to recall the location of the juice
Act: recall location of juice
You saw juice 1 near fridge 2 in kitchen 1.
Think: I can find juice 1 near fridge 2 in kitchen 1. I can see fridge 2.
Act: go to kitchen 1
You move to the kitchen (1). Looking quickly around the room, you see bench (1, 2), bookshelf (1, 2), chair (1, 2),
     dishwasher (1), faucet (2), fridge (1, 2), garbage can (1), kitchen cabinet (1, 2, 3, 4, 5, 6, 7, 8), kitchen
     counter (1, 2), kitchen table (1), microwave oven (1), sink (2), stove (1), toaster (1).
Think: I can see fridge 2.
Act: go to fridge 2
You arrive at the fridge (2). The fridge (2) is open. You see bench (1), cupcake (1), fridge (1, 2), juice (1)
Think: I find juice 1. Now I need to pick it up.
Act: pick up juice 1
```

```
You pick up juice. You hold juice (1).
Think: Now I accomplish the goal.
Act: done

### Agent Node 9
Your primary goal is to: and move the juice on the coffee table
To achieve this, you should perform your sibling tasks in sequence. At this level, your sibling tasks are: find and pick up the juice, and find the coffee table and put down the juice.
Your task is to: find the coffee table and put down the juice
You are in the house, and there are 4 rooms: bathroom (1), bedroom (1), kitchen (1), living room (1). You are in the middle of a kitchen (1). Looking quickly around the room, you see bench (1, 2), bookshelf (1, 2), chair (1, 2), dishwasher (1), faucet (2), fridge (1, 2), garbage can (1), kitchen cabinet (1, 2, 3, 4, 5, 6, 7, 8), kitchen counter (1, 2), kitchen table (1), microwave oven (1), sink (2), stove (1), toaster (1). You hold juice (1).
Think: I hold juice 1. Now, I need to find the coffee table. The coffee table is more likely to appear in living room 1.
Act: go to living room 1
You move to the living room (1). Looking quickly around the room, you see bookshelf (3), chair (3), closet (1), coffee table (1), computer (1), desk (1), garbage can (2), nightstand (1), sofa (1), tv (1). You hold juice (1).
Think: I can see coffee table 1.
Act: go to coffee table 1
You arrive at the coffee table (1). You see alcohol (2), bananas (1), bowl (7), cell phone (1, 2), coffee table (1), juice (1), lime (1), mug (1), plum (1, 2), sofa (1), wine (1) You hold juice (1).
Think: Now, I need to put down juice 1.
Act: put down juice 1
You put down juice on coffee table
Think: Now I accomplish the goal.
Act: done
```

H QUALITATIVE RESULTS FOR ALFRED

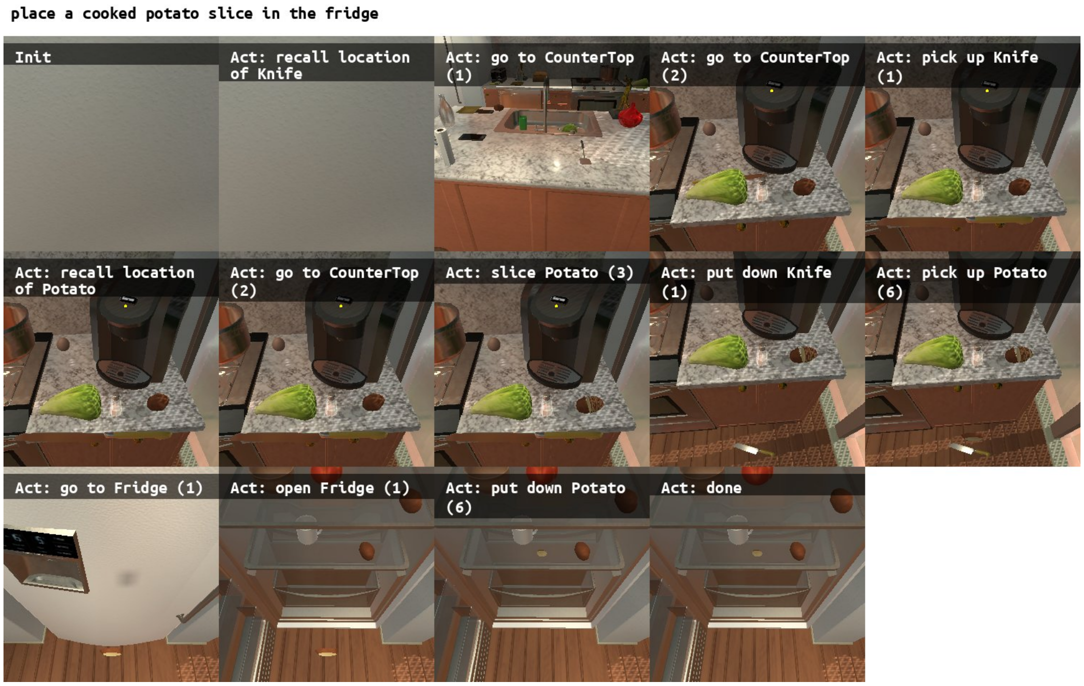
H.1 Success Case of *ReAcTree+WM* on ALFRED



Node 0	*Place a cooked potato slice in the fridge*	Node 3	*cook and pick up potato*
Node 1	*find and pick up knife*	Node 4	*place potato in fridge*
Node 2	*slice and pick up potato*		

Figure 6: Success case of *ReAcTree+WM* on ALFRED using LLaMA 3.1 70B.

H.2 Failure Case of *ReAct+WM* on ALFRED



place a cooked potato slice in the fridge
Init
Act: recall location of Knife
Act: go to CounterTop (1)
Act: go to CounterTop (2)
Act: pick up Knife (1)
Act: recall location of Potato
Act: go to CounterTop (2)
Act: slice Potato (3)
Act: put down Knife (1)
Act: pick up Potato (6)
Act: go to Fridge (1)
Act: open Fridge (1)
Act: put down Potato (6)
Act: done

Figure 7: Failure case of *ReAct+WM* on ALFRED using LLaMA 3.1 70B.