ReCAP: Recursive Context-Aware Reasoning and Planning with Language Models

Zhenyu Zhang
Department of Computer Science
Stanford University
zhenyuz5@stanford.edu

Tianyi Chen
Department of Computer Science
Stanford University
tchen288@stanford.edu

Weiran Xu
Department of Computer Science
Stanford University
weiran@stanford.edu

Abstract

Motivation. Intelligent agents must handle high-level goals and low-level actions over dozens of steps. Flat prompting strategies such as Chain-of-Thought, ReAct, or Reflexion struggle once the dialogue grows beyond the model's context window, causing plans to be forgotten or repeated. Empirically, we find that every open-source model at or below 14 B parameters *fails every task* in the long-horizon ROBOTOUILLE benchmark, exposing a severe *small-model planning gap*. **Method.** We present *ReCAP* (*Re*cursive *C*ontext-*A*ware *P*lanning), a hierarchical prompting framework that: (i) *recursively decomposes* a user goal into finer subtasks until a primitive action appears; (ii) stores each goal, reasoning trace, subtask list, and observation in a *dynamic context tree*; and (iii) triggers *backtracking-driven revision* whenever a leaf node completes or fails, re-prompting the LLM at its parent to update downstream plans. This keeps strategic intent adjacent to the current dialog turn, preventing drift. To lift small models, we add **ReCAP-DPO**: rank-16 LoRA adapters on Qwen-2.5-14B are tuned with *Direct Preference Optimization*. We mine 1,500 preference triples by pairing the first three subtask turns from successful GPT-4o runs (y^+) with the corresponding failed Qwen plans (y^-) under identical prompts; a single classification loss ($\beta = 0.1$) then shifts likelihood mass from y^- to y^+. **Implementation & Results.** The pipeline wraps ROBOTOUILLE in an OpenAI evaluation harness. All agents receive an "onion-cheese sandwich" one-shot demo, same context cap, and a $3\times$ step budget. *Full-suite GPT-4o:* **ReCAP** boosts pass@1 from **38 → 70 %** (sync) and **24 → 53 %** (async). *Cross-scale:* on three representative recipes (IDs 2, 4, 6) ReCAP raises success from {63, 10, 23, 37, 57}% (ReAct) to {90, 33, 53, 60, 87}% across GPT-4o, Qwen-32B, Qwen-72B, LLaMA-4 400B, and DeepSeek-V3 671B. **ReCAP-DPO** turns the 14 B model's 0 % into **20 %** success with only 0.17 % trainable parameters and 28 GB of GPU memory. **Discussion & Conclusion.** Structured recursion and context-aware memory allows even proprietary-size models to maintain coherent long-range plans without expanding context windows, while lightweight DPO+LoRA alignment revives commodity checkpoints. Limitations remain: ReCAP relies entirely on LLM judgments, so low-level execution errors propagate unchecked, and recursive prompting increases latency and API cost. Future work will separate high-level planning from primitive execution, introduce graph-based memory for targeted retrieval, and explore self-generated preference signals to remove oracle

Preprint. Under review.

dependence. Overall, ReCAP and ReCAP-DPO demonstrate a recipe for scalable, compute-efficient embodied agents.

Abstract

We present *ReCAP*, a recursive, context-aware prompting framework that keeps long-horizon plans near an LLM's current context via a dynamic context tree and backtracking-driven refinement. On the embodied ROBOTOUILLE benchmark ReCAP boosts GPT-4o's pass@1 success rates from 38 $\rightarrow$ 70% (synchronous) and 24 $\rightarrow$ 53 % (asynchronous) and consistently outperforms ReAct across five model scales (32 B – 671 B). All models $\leq$ 14 B parameters, however, still fail. To bridge this gap we introduce *ReCAP-DPO*: rank-16 LoRA adapters on Qwen-2.5-14B are tuned with 1,500 GPT-4o-vs-Qwen preference triples using Direct Preference Optimization. The aligned agent leaps from 0 % to 20 % success while adding only 0.17 % trainable parameters, demonstrating that structured recursion plus lightweight preference alignment can deliver compute-efficient, long-horizon reasoning.

1 Introduction

A key characteristic of intelligence is the fluid transition between high-level abstract reasoning and low-level concrete execution—something humans routinely perform in everyday tasks [13]. Imagine preparing a complex dish like an onion cheese sandwich. The process begins with formulating a broad strategic plan—identifying essential ingredients, selecting appropriate tools, and outlining a feasible sequence of steps. This high-level plan is then progressively refined into precise, actionable subtasks such as gathering bread slices, slicing onions, and assembling ingredients in the correct order. However, real-world interactions typically introduce unexpected complexities: the cutting board may already be occupied with another ingredient, a needed utensil might be temporarily unavailable, or intermediate steps previously planned could turn incomplete or incorrect. Addressing such scenarios demands real-time long-horizon adaptive reasoning and planning, requiring an intelligent system not only to maintain its original intention but also to flexibly accommodate unexpected deviations through context-aware plan revision and high-to-low-level transitions [18, 4].

Recent advances have demonstrated the impressive capabilities of LLMs in complex sequential reasoning and decision-making tasks through frameworks such as ReAct [4] and Reflexion [16]. By leveraging interactive reasoning and self-reflective prompting strategies, these methods enable LLM agents to interleave reasoning with action, enhancing their problem-solving performance. However, in long-horizon settings, sequential execution frameworks often encounter a key limitation: early plans and interactions may no longer fit within the LLM's context window, causing the agent to lose track of its high-level strategy or repeatedly attempt previously failed actions. Even with extended context lengths, LLMs still struggle to accurately retrieve and follow prior plans [11]. While Reflexion introduces iterative self-reflection to revise memory and improve robustness, it relies on multiple rounds of trial and correction, which can be inefficient and unstable in real settings. Without mechanisms for effective context recall and adaptation, these methods remain fragile to unexpected feedback and execution errors [22, 3], often resulting in unproductive action loops and reduced task success rates.

To overcome these limitations, we introduce RECAP (**Re**cursive **C**ontext-**A**ware reasoning and **P**lanning), a novel hierarchical reasoning framework specifically developed to enhance the adaptability of LLM agents in long-horizon, multi-step environments. ReCAP comprises three novel components: **(1) recursive hierarchical decomposition**, systematically dividing complex tasks into relatively simple subtasks and recursively executing them; **(2) dynamic context tree**, tracking evolving task hierarchies, ongoing reasoning trajectories, and external environment feedback; and **(3) a real-time adaptive subtask generation and backtracking mechanism**, enabling continuous plan refinement and error recovery as tasks evolve. At its core, the dynamic context tree acts as a structured memory representation, enabling ReCAP agents to alternate between reasoning levels by descending into detailed subproblems and ascending back to higher-level goals upon completion or detection of failures. These three components enable ReCAP agents to perform **recursive execution**. Such a

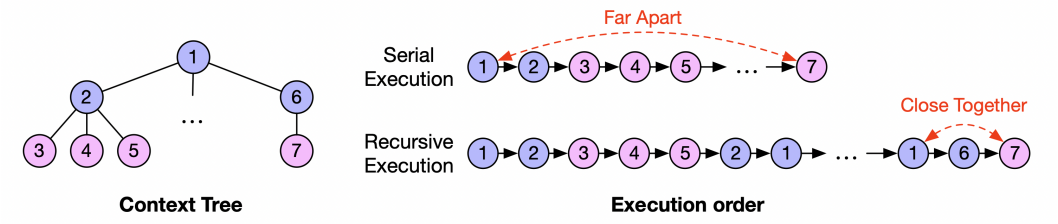

Figure 1: Execution flow and context structure in sequential vs. recursive execution. Sequential execution (e.g., ReAct) pushes early plans further into context, while ReCAP's recursive strategy keeps high-level goals closer to the current execution point, preserving semantic alignment.

structure provides critical context awareness across task progressions, mitigating the weakness of previous sequential reasoning methods.

We benchmark ReCAP in two settings on the high-fidelity ROBOTOUILLE cooking simulator [6]. First, on *all* ten synchronous and ten asynchronous tasks using GPT-4o, ReCAP boosts pass@1 success from 38 % to 70 % in synchronous mode and from 24 % to 53 % in asynchronous mode. Second, to probe model-scale generality we test three representative synchronous recipes (IDs 2, 4, 6) under a 64-message context cap across five LLMs. ReCAP raises success rates for Qwen2.5-32B, Qwen2.5-72B, LLaMA-4 (400B), and DeepSeek-V3 (671B) from {10, 23, 37, 57}% to {**33**, **53**, **60**, **87**}%, respectively, and lifts GPT-4o from 63 % to **90**%. Strikingly, every model at or below 14 B parameters scores 0 %, exposing a "small-model planning gap" in long-horizon embodied tasks.

To close this gap we incorporate *Direct Preference Optimization* (DPO) [14] with *Low-Rank Adaptation* (LoRA) [7]. We pair 1,500 *successful* GPT-4o plans with *failed* Qwen2.5-14B plans for identical inputs and fine-tune only rank-8 LoRA adapters in the smaller model. After three epochs the resulting *ReCAP-DPO* agent attains a 20 % success rate, transforming an unusable 14 B model into one that attains non-zero success rate.

In summary, ReCAP couples structured recursion with context-aware memory to deliver long-horizon reasoning; its merits generalize across model scales; and lightweight DPO can rescue small models that otherwise fail.

2 Related Work

Chain-of-thought (CoT) prompting has emerged as a simple yet powerful technique to elicit step-by-step reasoning in LLMs, improving performance on complex multi-step problems [19]. Building on this idea, the ReAct framework interleaves reasoning traces with actionable steps, enabling an agent to query external tools or environments in the midst of its chain of thought [4]. These strategies bolster the interactivity and interpretability of LLM reasoning; however, because they operate in a strictly sequential fashion, they often struggle to maintain long-horizon consistency. In extended decision-making or tool-use scenarios, a CoT or ReAct agent may lose track of global objectives or earlier context, as there is no mechanism for revisiting earlier decisions or enforcing an overarching plan. To address the limitations of linear thought chains, researchers have proposed hierarchical and search-based reasoning frameworks. Tree-of-Thoughts (ToT) generalizes CoT prompting by allowing the model to branch into multiple possible thought paths and then search or backtrack among them to find a more coherent solution [5]. By exploring a tree of reasoning steps with self-evaluation at each branch, ToT and similar methods introduce lookahead and deliberation beyond a greedy left-to-right generation. This improves performance on tasks requiring planning or exploration, such as puzzle solving and game-play, by considering alternate reasoning pathways. However, the breadth-first or backtracking search over thoughts comes with a much higher computational cost, and its applicability is often confined to short, self-contained problems. In open-ended or real-time agent domains, an exhaustive search may be impractical, and a strict tree search may still fail to incorporate new information obtained as the environment evolves.

Reflexion [16] is a self-reflective agent framework that reinforces an LLM-based policy through trial-and-error, using verbal self-critique on failed attempts to refine its behavior incrementally. This strategy requires many reasoning rounds with frequent environment resets, which can become

inefficient on long-horizon tasks that demand extended, coherent execution without interruption. Similarly, AutoGPT [21] pioneered fully autonomous goal decomposition and action execution by an LLM agent, but it often suffers from reliability issues and aimless looping due to the unpredictability of a purely language-driven planning process. Meanwhile, Voyager [18] demonstrates continuous skill acquisition in an open-ended world (Minecraft) via iterative code generation and self-verification, achieving significant improvements in exploration and skill reuse; however, its domain-specific design and reliance on constant environment feedback limit its generalizability. In contrast, ReCAP offers a more robust, structured, and context-aware alternative that avoids inefficient retries through comprehensive planning, thereby enhancing task generalization and ensuring more stable execution even on complex long-horizon tasks.

RL-based Alignment and Direct Preference Optimization. Reinforcement learning from human feedback (RLHF) [2] aligns LLMs with user intent by training a reward model from pairwise preference data and then optimizing the policy with Proximal Policy Optimization (PPO) [15]. Although effective, RLHF entails a two-stage pipeline and extensive hyper-parameter tuning; recent work therefore explores lighter alternatives such as RLAIF, which replaces human labels with AI-generated preferences [9], and self-rewarding language models that iteratively judge and improve their own generations [23]. Direct Preference Optimization (DPO) streamlines the pipeline by showing that the optimal RLHF policy can be obtained via a simple classification loss without explicit reward modeling or on-policy sampling [14]. DPO has matched or surpassed PPO-based RLHF on summarization, dialogue, and sentiment control while being substantially simpler to implement. In practice, parameter-efficient fine-tuning methods such as LoRA [7] further reduce computational cost, allowing billions-parameter models to be aligned with only a few megabytes of trainable weights. Complementary studies on reward-model evaluation and benchmarking—e.g. RewardBench [10] and comprehensive RM surveys [1]—highlight the importance of reliable preference signals for stable policy optimization. Our work situates ReCAP within this alignment landscape: we retain its architectural advantages and demonstrate that a lightweight DPO+LoRA training can close the small-model performance gap exposed by our Robotouille benchmarks.

3 ReCAP: Recursive Context-Aware Reasoning and Planning

3.1 Framework Overview

ReCAP achieves recursive execution by maintaining a dynamic context tree, where each node represents a task and stores the reasoning history and subtasks generated by the LLM and observations from the task environment. At each turn of the dialogue, the LLM is prompted using the node's local context—including environmental observations and prior thoughts—and responds with either (i) a decomposition into subtasks, which extends the tree, or (ii) an action that can be executed directly in the environment. When a subtask is completed, ReCAP backtracks to its parent task to allow the LLM to refine the higher-level plan based on updated observations.

ReCAP begins by creating a root node $p \leftarrow \texttt{Node}(g)$ from the initial goal g and calling $\texttt{ReCAP}(p, o, A, \texttt{LLM})$ where o is the initial observation and A the set of valid actions. This call triggers the recursive loop of reasoning, planning, and acting described in Algorithm 1.

3.2 Dynamic Context Tree

The only external data structure maintained by ReCAP is a dynamic context tree, which explicitly encodes the dependency relationships between recursively generated subtasks. Each node in ReCAP's dynamic context tree is represented as a structured tuple `(desc, subtask_list, children_list, obs_list, think_list)`, where `desc` denotes the natural language description of the current task, `subtask_list` contains the latest list of planned subtasks, `children_list` refers to the list of completed subtasks nodes (represented as child nodes), `obs_list` stores the sequence of environmental observations received during execution, and `think_list` represents the list of intermediate reasoning steps generated by the LLM for the current task. As ReCAP recurses, nodes grow by appending new subtasks, observations, and thoughts, providing a local view that supports both breakdowns into subtasks and backtracking for plan refinement. In Algorithm 1, we denote `Node(desc)` as the creation of a new task node initialized with the task description `desc`, where all other fields—`subtask_list`, `children_list`, `obs_list`, and `think_list`—are initially empty.

Algorithm 1 ReCAP: Recursive Context-Aware Reasoning and Planning

Require: Task node p, observation o, valid actions A, dialog model `LLM`
Ensure: Updated observation o after resolving task

$q \leftarrow$ `GeneratePromptFromNode`(p, o) {Initial prompt generation}
$r \leftarrow$ `LLM.invoke`(q) {Initial subtask plan and reasoning}
p.`SetSubtasksAndThink`(r) {Store initial subtasks for this task}
p.`SetObs`(o) {Store initial observation}
$t \leftarrow p$.`GetTaskDescription`(p)
while `HasRemainingSubtasks`(p) **do**
  $t' \leftarrow$ `NextSubtask`(p) {Get the current subtask}
  if `IsPrimitive`(t') **then**
    if `IsValidAction`(t', o, A) **then**
      Execute action t' in environment
      Update observation o based on new environment state
    else
      Mark subtask t' as failed
    end if
  else
    $c \leftarrow$ `Node`(t') {Create new subtask node}
    p.`AddChild`(c) {Add the new node to the `children_list` of the current node}
    $o \leftarrow$ `ReCAP`$(c, o, A,$ `LLM`$)$ {Recursively resolve subgoal}
  end if
  $q' \leftarrow$ `GenerateRefinePrompt`(p, o, A)
  $r' \leftarrow$ `LLM.invoke`(q')
  p.`UpdateSubtasksAndThink`(r') {Refine current thought and remaining subtasks}
  p.`UpdateObs`(o) {Update observation}
end while
return o

3.3 Recursive Task Decomposition and Execution

The core of ReCAP is its recursive hierarchical decomposition mechanism, centered around a task node pointer p that indicates the current task, and an associated subtask list and thought history maintained within each node. Starting from a user-specified high-level task, ReCAP queries an LLM, which generates thoughts and corresponding decomposable subtasks for the current task. The pointer p then moves downward through the first remaining subtask in the newly generated subtask list toward a more specific action. This process then repeats, forming the downward recursive flow of task decomposition.

This downward recursion temporarily pauses once a subtask generated by the LLM corresponds directly to an executable action within the current environment. Such subtasks then become the leaf nodes of the task hierarchy, and their validity is evaluated against the currently executable actions given by the latest environmental constraints. Regardless of whether this leaf-level subtask execution succeeds or fails (e.g., due to action invalidity or unforeseen environmental conditions), ReCAP initiates a backtracking procedure to its parent node to further refine the high-level reasoning and revise the subtask plan accordingly.

3.4 Backtracking and Next Subtask Generation

Backtracking is triggered whenever a child node—whether a leaf or an intermediate subtask—completes execution (i.e., no remaining subtasks), regardless of success or failure. During backtracking, a new prompt is constructed to include the latest environmental observations and the parent node's task-specific information—such as its description (`desc`), reasoning trace (`think_list`), and previously generated subtask list (`subtask_list`). This prompt guides the LLM to update the parent node's high-level reasoning and refine its previous subtask list based on the new observation.

Depending on the updated environment state, the LLM may prune completed steps from the subtask list, refine the remaining steps, or maintain the current plan. ReCAP then proceeds by selecting the

first unfinished subtask from the refined list, instantiates it as a new child node, and continues the recursive process. Figure 2 illustrates the local structure of backtracking and next subtask generation.

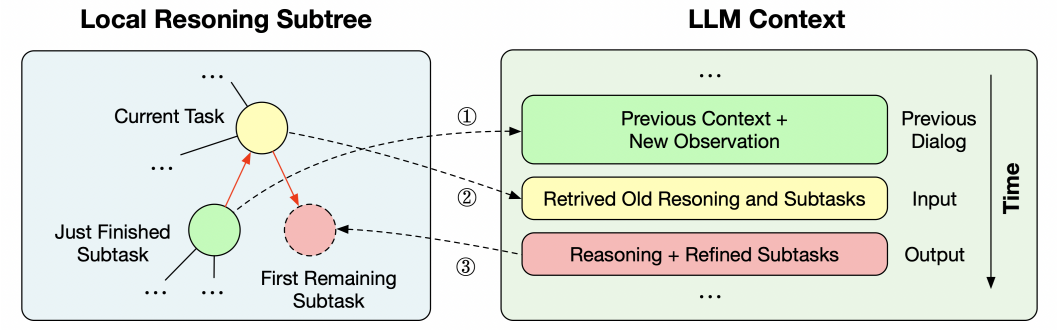

Figure 2: Overview of ReCAP's backtracking and subtask regeneration: ① The completed subtask and new observation are appended to the LLM context. ② Previous reasoning and subtask plans are retrieved and included as user input. ③ The LLM generates refined reasoning and a new subtask list, from which the next subtask is selected to continue running.

The main advantage of our backtracking mechanism is its ability to dynamically retrieve and reintroduce high-level reasoning into the current prompt. This keeps task-relevant information close to the LLM's decision point, maintaining semantic coherence across long-horizon plans. In addition, backtracking enables structured error correction and minor refinements to successful plans, guided by real-time feedback from the environment.

3.5 ReCAP-DPO: Preference-Aligned Task Decomposition for Small Models

The results in Section 5 reveal a *small-model planning gap*: every language model at or below 14 B parameters failed all task instances, whereas larger checkpoints succeeded frequently. Close inspection showed that these failures concentrate in the **high-level planning stage**: small models output subtask lists that (i) violate API syntax or omit required arguments, (ii) place steps in logically impossible order, and (iii) are easily confused by the few-shot demonstration, copying irrelevant phrases or slotting ingredients into the wrong template. We therefore align only the first few dialogue turns, which are those responsible for high-level decomposition and correct formatting, using DPO with LoRA, while leaving the remainder of the ReCAP control loop unchanged.

Preference Dataset. During our GPT-4o evaluations we logged every ReCAP dialogue. For each task instance that *successfully* completed, we extract the first few turns—corresponding to high-level subtask generation—as *positive* responses y^+. We then run the same prompt on an untuned Qwen2.5-14B-Instruct model; its corresponding high-level plans (which almost always lead to downstream failure) serve as *negative* responses y^-. Each preference triple (x, y^+, y^-) shares the same input x (root goal, environment description, and tool list). Aggregating across ten synchronous and ten asynchronous recipes yields **1,500** triples.

DPO Objective. DPO shows that the optimal RLHF policy can be obtained with a simple binary-comparison loss that bypasses reward-model training and on-policy sampling [14]. Given model π_θ and inverse-temperature β we minimize

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y^+,y^-)}\Big[\log \sigma\big(\beta\big[\log \pi_\theta(y^+ \,|\, x) - \log \pi_\theta(y^- \,|\, x)\big]\big)\Big], \tag{1}$$

where $\sigma(\cdot)$ is the logistic function. Intuitively, the gradient pushes the model to assign higher likelihood to y^+ than to y^- for the *same* input, directly aligning the planner with successful decomposition patterns.

Parameter-Efficient Fine-Tuning. We load Qwen2.5-14B-Instruct in 8-bit. Rank-**16** LoRA adapters (`r=16`, `lora_alpha=32`, dropout 0.05) are inserted only in `q_proj`, `k_proj`, `v_proj`, and `o_proj`. All backbone weights remain frozen; the adapters add $\approx 0.17\%$ trainable parameters.

We train for **3 epochs** with learning-rate 1×10^{-4}, per-device batch size 4, and gradient-accumulation 8 (effective batch 32). 8-bit quantisation plus LoRA keeps peak memory below 28 GB, fitting comfortably on a single A100-40 GB.

4 Evaluation

We evaluate ReCAP on ROBOTOUILLE [6], a high-fidelity cooking simulator expressly designed to stress long-horizon reasoning. All experiments use a strict **pass@1** protocol: each agent receives exactly one uninterrupted reasoning–execution trajectory per test instance—no retries, beam search, or ensembling. This setting isolates raw decision quality and avoids auxiliary gains from techniques such as self-consistency, inner monologue [8], or multi-trial majority voting. Agents are prompted one-shot; we reuse the same "onion-cheese sandwich" demonstration for every method and adapt it to match each reasoning format. Unless otherwise noted, GPT-4o runs with the maximum context window.

4.1 Robotouille

ROBOTOUILLE offers two modes. In *synchronous* mode every action completes immediately; in *asynchronous* mode long-running actions (e.g., baking, filling water) progress in parallel, forcing the agent to interleave subgoals. Minimal optimal trajectories span 10–57 steps synchronously and 21–82 steps asynchronously, far longer than prior text-only kitchens and greatly exceeding typical LLM context lengths. Seemingly simple operations can require multiple atomic commands (e.g., three successive `cut` actions to slice a vegetable). As subtasks accumulate, early plans drift out of context, and the environment frequently introduces blockages (e.g., occupied cutting boards) that demand real-time replanning.

We follow prior work and evaluate on ten synchronous and ten asynchronous recipes, each with ten official test seeds. For reproducibility we cap the dialogue history at 64 messages and set the environment's `max_step_multiplier` to 3.

4.2 Baselines

We compare ReCAP with four prompting baselines. **Standard** removes thoughts and actions entirely, asking the model to emit the full action sequence in a single answer. **Chain-of-Thought (CoT)** [20] appends free-form reasoning to Standard. **ReAct** [4] interleaves chain-of-thought, action execution, and observation. **Act-only (Act)** removes the reasoning lines from ReAct, mimicking WebGPT's API-call style [12]. All methods receive the same environment description, the same step limits, and the same few-shot demo.

4.3 Performance Across Model Scales

To gauge whether ReCAP's benefits persist across parameter counts, we replicate the evaluation on three representative synchronous recipes (IDs 2, 4, and 6) while limiting each run to 64 context messages. We test five models—GPT-4o, DeepSeek-V3 (671B), LLaMA-4 (400B), Qwen2.5-72B, and Qwen2.5-32B—as well as the smaller Qwen2.5-14B that will later be fine-tuned. For each model we run both ReAct and ReCAP under identical hyper-parameters and log pass@1 success. Aggregate statistics are reported in Section 5.2.

4.4 Effect of DPO Alignment

Finally, we assess whether preference alignment rescues the failure mode of the 14B model. We fine-tune Qwen2.5-14B-Instruct with the DPO+LoRA procedure described in Section 3.5, then rerun ReCAP on the same synchronous recipes (IDs 2, 4, and 6). We compare the pass@1 success of the *ReCAP-DPO* agent against the un-aligned Qwen2.5-14B under ReCAP. Outcome metrics appear in Section 5.3.

5 Results

5.1 Main Results

Table 1 summarizes the performance of ReCAP and baseline methods on Robotouille (synchronous and asynchronous). These tasks vary significantly in their action sequence lengths and planning demands, allowing us to evaluate how different approaches scale with reasoning horizon. This ordering provides a natural progression in long-horizon complexity, which strongly correlates with the observed performance differences.

Table 1: Average performance (%) across different tasks and methods using GPT-4o

Task	Step Range	ReCAP	ReAct	CoT	Act	Standard
Robotouille (Async)	21-82	**53.0**	24.0	5.0	8.0	2.0
Robotouille (Sync)	10-57	**70.0**	38.0	14.0	31.0	12.0

ReCAP performs better than ReAct on long-horizon tasks by a large margin. On synchronous Robotouille, it achieves a 32% gain, and on asynchronous Robotouille, a 29% improvement, representing its strength in long-horizon environments where early goals are prone to being overwritten in flat prompting setups. In these cooking tasks, high-level objectives often span multiple atomic actions and require multi-phase coordination. While ReAct performs all reasoning sequentially and suffers from context overflow, ReCAP maintains a dynamic task hierarchy through its context tree and supports explicit backtracking, enabling the agent to preserve global goals and revise local decisions when needed. These features allow ReCAP to better handle the challenges of concurrent subgoals, delayed dependencies, and evolving observations, all of which are common in Robotouille.

We further perform a detailed failure case analysis to understand the nature of errors across task regimes. On **easy to medium tasks** (synchronous #1–5, asynchronous #1–3), ReCAP achieves near-perfect success, with failures mostly due to minor errors like missing the final cut in a sandwich or misplacing the completed item. In contrast, ReAct exhibits more fundamental mistakes even on simple recipes. On **long-chain or multi-dish tasks** (synchronous #8–10, asynchronous #4,5,8–10), ReCAP may occasionally make imperfect subtask choices, but it never enters deadlock. It consistently detects failure signals and backtracks to generate a revised plan. ReAct, on the other hand, frequently enters infinite loops when encountering blocked stations. For instance, if `lettuce1` is occupying `board2`, ReAct will repeatedly attempt to cut `onion1` by stacking and unstacking it on the blocking item without resolving the underlying issue. ReCAP avoids such failure by leveraging its dynamic context tree to identify the blockage and generate a corrected plan—e.g., moving `lettuce1` to an empty table before proceeding with the cut—thus maintaining task progress and preventing deadlocks. Figure 3 illustrates a more detailed dialogue trace in which ReAct becomes trapped in an infinite unstack/stack loop, whereas ReCAP detects the looping behavior via its internal "Think" step, backtracks to move the obstruction, and then successfully resumes execution.

5.2 Performance on Different Models

To assess how our ReCAP generalizes across model scales compared to the standard ReAct setup, we evaluated both architectures on three representative synchronous Robotouille tasks (IDs 2, 4, and 6). To control API costs, we limited our experiments to these three tasks and imposed a hard cap of 64 context messages per run (any excess was truncated). We also set `max_step_multiplier` to 3. We benchmarked four open-source and proprietary LLMs—Qwen2.5-32B, Qwen2.5-72B, LLaMA-4 (400B), and DeepSeek-V3 (671B)—alongside GPT-4o, covering a broad range of parameter counts. Table 2 reports per-task and average success rates (%) for each model under both ReAct and ReCAP. Across every model tested, ReCAP yields a higher average success rate than ReAct, demonstrating that our recursive, context-aware reasoning strategy consistently improves long-horizon performance independent of model size.

5.3 ReCAP-DPO Results

After DPO fine-tuning on 1,500 high-level planning examples for 3 epochs, Qwen2.5-14B-Instruct achieved a 20% on Robotouille success rate under the ReCAP framework, up from 0% before

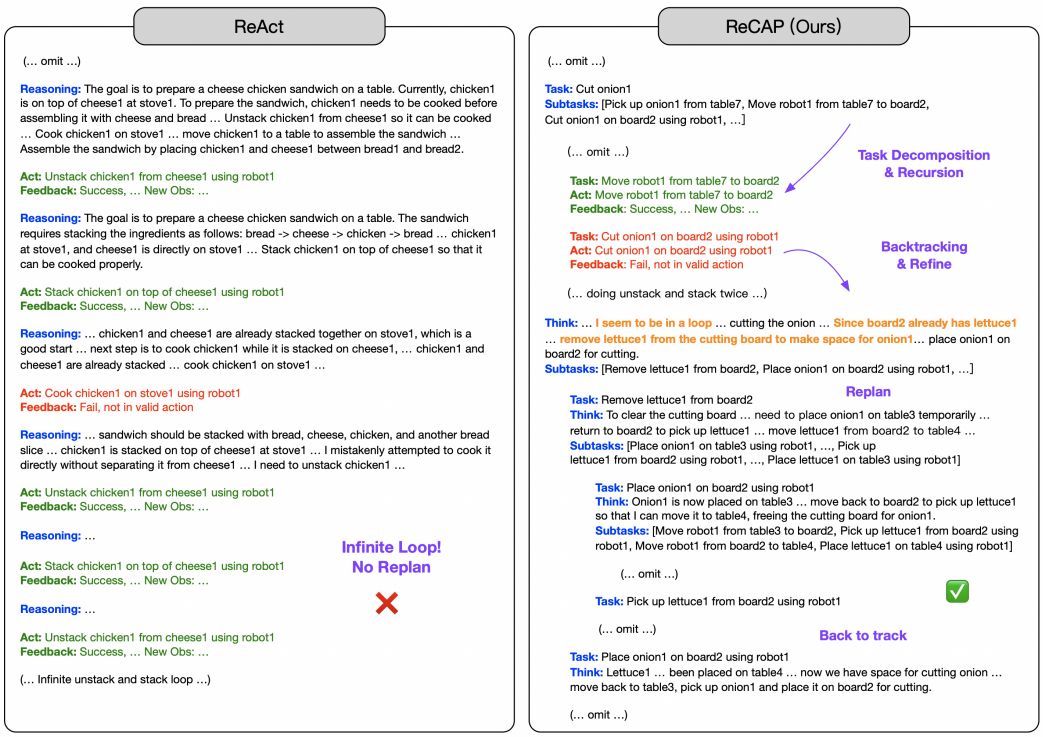

Figure 3: Detailed comparison of ReAct vs. ReCAP on a blocked station. **Left:** ReAct repeatedly alternates between stacking and unstacking the same item, entering an infinite loop. **Right:** ReCAP detects the deadlock, backtracks to clear the board by moving the blocking lettuce, and then proceeds with the correct sequence of actions.

Table 2: Average success rate comparison between ReAct and ReCAP across model scales

Method	**GPT-4o** –	**Qwen2.5** 14B	**Qwen2.5** 32B	**Qwen2.5** 72B	**LLaMA-4** 400B	**DeepSeek-V3** 671B
ReAct	63	0	10	23	37	57
ReCAP	**90**	0	**33**	**53**	**60**	**87**

fine-tuning. This improvement shows that DPO-based alignment can help a smaller model generate more correct high-level subtasks within ReCAP's recursive reasoning process. Although fine-tuned performance remains below larger models, achieving a nonzero success rate indicates that even modest amounts of preference data can substantially boost small-model planning.

5.4 Ablation Studies

To further investigate the effectiveness of our structure, we conducted extended ablation studies on the Robotouille task `synchronous/6_lettuce_tomato_cheeseburger`, which requires 23 (theoretical optimal) to 40 (average) rounds of agent-environment interaction to complete. We also evaluate the statistical significance of differences in success rates between the structural variants and the original version. Table 3 reports the success rates and p-values for various ReCAP structural variants, including alterations to the maximum reasoning depth (LEVEL 2/3/4/5), omission of reasoning traces during backtracking (NAME ONLY), and modifying the output format to generate only decomposition/action outputs without the "think" reasoning (NO THINK), as well as passing all think history instead of just the most recent (THINK MANY). All structural variants were evaluated with a context length of 128, where context length refers to the number of messages stored in the LLM history during the conversation. For the NO THINK and NAME ONLY variants, we adapted the

one-shot prompt to match their structure. Table 4 presents the success rates and p-values for context length variants using the original structure.

Table 3: Statistical comparison of ReCAP structural variants: success rates and p-values.

ReCAP Variant	Success Rate (%)	p-value
ReCAP-original	80	-
ReCAP-think_many	70	0.32
ReCAP-no_think	60	0.14
ReCAP-name_only	55	**0.0098**
ReCAP-level_5	70	0.32
ReCAP-level_4	60	0.14
ReCAP-level_3	10	**0.00015**
ReCAP-level_2	0	**0.000012**

Table 4: Statistical comparison of ReCAP context length variants: success rates and p-values.

ReCAP Variant	Success Rate (%)	p-value
ReCAP-original-128	80	-
ReCAP-original-64	55	0.091
ReCAP-original-32	70	0.47
ReCAP-original-16	55	0.091

For the long-horizon task `synchronous/6_lettuce_tomato_cheeseburger`, the success rate degrades significantly when reasoning traces are removed or when the maximum reasoning depth is restricted (LEVEL 2/3). This suggests that the explicit reasoning traces produced by ReCAP help the LLM perform better by allowing it to recall previous subtasks and lines of reasoning. With restricted reasoning depth, the LLM is limited in its ability to recursively decompose higher-level tasks into atomic, directly executable actions, forcing it to generate actions from insufficiently decomposed subtasks and thus reducing accuracy.

On the other hand, the THINK MANY and NO THINK variants achieve success rates comparable to the original, indicating that ReCAP is robust even when the LLM is provided with either excessive reasoning history or only decomposition/action outputs without the intermediate "think" reasoning. This robustness is also observed in the context length variants: no significant performance degradation occurs when limiting the number of messages stored in the LLM history during the conversation, implying that ReCAP remains effective under context-sensitive scenarios.

5.5 Cost Estimate

We conducted cost estimation on Robotouille for ReCAP, and cost comparison between ReCAP and ReAct on ALFWorld [17], a dataset similar to Robotouille but with simpler dependencies and shorter horizons for cost saving.

For the Robotouille task `synchronous/6_lettuce_tomato_cheeseburger`, the average number of LLM calls is 74.95 with a standard deviation of 27.87, and the average cumulative cost for one complete run is 7.77 USD with a standard deviation of 3.45 USD.

For ALFWorld, the total cost of running all 134 tasks in the test set is 37.89 USD using ReAct, and 118.40 USD using ReCAP—approximately three times the cost of ReAct. We identified that the extra cost mainly comes from the additional reasoning traces in the input and the extra steps required for intermediate task decomposition.

6 Discussion

Limitations. Although ReCAP, augmented with DPO, greatly improves long-horizon success, the framework still hinges on the underlying language model for every decision. Without external grounding or formal verification, incorrect reasoning by the LLM can propagate unchecked through

the dynamic context tree. Our DPO alignment targets only the first few planning turns; low-level execution errors remain uncorrected and occasionally derail otherwise sound plans. In addition, collecting preference pairs requires access to a stronger oracle model (GPT-4o here), limiting scalability for domains where such a model is unavailable. Finally, recursive prompting lengthens interaction trajectories, raising latency and API cost—an issue magnified in the asynchronous ROBOTOUILLE mode.

Broader Impact. A compute-efficient, preference-aligned planner lowers the barrier to deploying autonomous agents on modest hardware, potentially democratizing embodied AI research. Conversely, the same technology could amplify misuse: an aligned small model may execute long-horizon instructions that facilitate disallowed behavior if the preference data are biased or incomplete. Careful auditing of alignment data and explicit safety constraints will be essential before real-world deployment.

Project Challenges. Three challenges dominated the project. (i) **Data curation.** Instrumenting ROBOTOUILLE to capture *only* the high-level subtask turns, and filtering out dialogues whose `chosen` string accidentally contained user roles, required substantial logging utilities. (ii) **Resource constraints.** Even with 8-bit loading, Qwen2.5-14B plus rank-16 LoRA adapters consumed close to 28 GB; we iterated on smaller ranks before finding a stable configuration. (iii) **Cost.** GPT-4o queries for 1,500 preference triples were expensive; we mitigated this by limiting collection to the most informative synchronous tasks and pruning redundant demonstrations. These difficulties motivated our shift from PPO to the simpler, data-efficient DPO objective, which is more tractable under the available budget and hardware.

7 Conclusion

We introduce ReCAP, a recursive, context-aware reasoning and planning framework that enables LLM agents to tackle complex, long-horizon tasks through hierarchical decomposition and adaptive execution. By maintaining a dynamic context tree and supporting subgoal-level backtracking, ReCAP offers a structured alternative to flat prompting, significantly improving robustness and decision quality in feedback-rich environments. Our experiments across embodied and symbolic tasks demonstrate consistent gains without requiring any model training, highlighting ReCAP's ability to generalize through architecture alone. This opens new directions for non-linear context representations, modular planning systems, and memory-efficient reasoning, laying the groundwork for more scalable and generalizable LLM-based agents.

Building on these foundations, we leverage DPO and LoRA to further adapt ReCAP for challenging settings that require alignment and efficient fine-tuning. Future work includes modularizing the system by separating high-level planning from low-level execution, enabling specialized models to collaborate more effectively. LoRA's parameter efficiency allows for selective adaptation of different modules, improving flexibility and reducing compute costs. Additionally, optimizing the recursive context tree—for example, by structuring memory as an executable graph or enabling targeted retrieval—may further enhance reasoning under context constraints. Integrating preference-based objectives like DPO with advanced memory or routing strategies presents a promising path for improving both the scalability and decision quality of ReCAP in complex environments.

8 Contributions

- Zhenyu Zhang: Responsible for the design and implementation of the agent and algorithm, including the context tree, recursive subtask decomposition, backtracking, and error correction. Also designed and analyzed ablation studies. Compared to the original proposal, my role shifted from focusing on evaluation to leading the algorithm's design and implementation, while evaluation was handed over to two other team members. This adjustment was made to better leverage our respective strengths and improve our overall efficiency.
- Tianyi Chen: Set up the ROBOTOUILLE simulation environment and integrated both the base Qwen2.5-14B model and the GPT-4o oracle into the pipeline, development of the DPO fine-tuning loop, and implemented the data-collection scripts that automatically log successful GPT-4o runs, pair them with the corresponding failed Qwen outputs, and construct the

positive/negative triples required for DPO training. Aside from switching from PPO to DPO and adding the sample-mining code, the responsibilities remain unchanged from the original proposal.

- Weiran Xu: Wrote detailed one-shot example prompts and conducted research on multi-GPU training, focusing on implementing all baselines and running them on different models. Also monitored recent papers for new techniques that we might incorporate, such as improved reward modeling or enhanced prompting strategies for the oracle. Coordinated the writing of the final report. My responsibilities have remained largely unchanged from the proposal, slightly shift more to running experiments to better support the whole workflow.
- Sponsors: Alex Pentland, Jiaxin Pei. Sponsored OpenAI API and Together AI API for testing various models.

References

[1] Rui Chen, Mingyu Zhang, and Xiang Li. A comprehensive survey of reward models: Taxonomy, progress, and challenges. *arXiv preprint arXiv:2504.12328*, 2024.

[2] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017.

[3] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a llm, 2023.

[4] Yao et al. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[5] Yao et al. Tree of thoughts: Deliberate problem solving with large language models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[6] Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for llm agents, 2025.

[7] Edward Hu, Yelong Shen, Phil Wallis, Zeyuan Allen, Zhiqing Saunders, Lu Li, and Deep Ganguli. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2022.

[8] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022.

[9] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, and Colton et al. Bishop. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback. In *arXiv preprint arXiv:2309.00267*, 2023.

[10] Yaoming Li, Danni Cheng, Tianyun Zhi, and et al. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024.

[11] Nelson F Liu, Kevin Lin Yu, Nora Kassner, Fan Du, Daniel Khashabi, Ashish Sabharwal, and Noah A Smith. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023. Computer Science > Computation and Language, arXiv:2307.03172.

[12] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022.

[13] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[14] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *arXiv preprint arXiv:2305.18290*, 2023.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

[16] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

[17] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021.

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

[19] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.

[20] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[21] Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions, 2023.

[22] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[23] Weicheng Yuan, Yuntao Bai, Julian Michael, and et al. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.