

One STEP at a time: Language Agents as Stepwise Planners

Anonymous ACL submission

Abstract

Language agents have shown promising adaptability in dynamic environments to perform complex tasks. However, despite the versatile knowledge embedded in large language models, these agents still fall short when it comes to tasks that require planning. We introduce STEP, a novel framework designed to efficiently learn from previous experiences to enhance the planning capabilities of language agents in future steps. Concretely, STEP functions through four interconnected components. First, the *Planner* takes on the task, breaks it down into subtasks and provides relevant insights. Then the *Executor* generates action candidates, while the *Evaluator* ensures the actions align with learned rules from previous experiences. Lastly, *Memory* stores experiences to inform future decisions. In the ScienceWorld (Wang et al., 2022) benchmark, our results show that STEP consistently outperforms state-of-the-art models, achieving an overall score of 67.4 and successfully completing 12 out of 18 tasks. These findings highlight STEP’s potential as a framework for enhancing planning capabilities in language agents, paving the way for more sophisticated task-solving in dynamic environments.¹

“You don’t have to see the whole staircase, just take the first step.” – Martin Luther King Jr.

1 Introduction

Autonomous agents that incorporate Large Language Models (LLMs) as integral cognitive systems (Sumers et al., 2023) have demonstrated significant capabilities in addressing a diverse range of interactive tasks e.g., mathematical problems (Cobbe et al., 2021; Hendrycks et al., 2021), programming challenges (Zhuo et al., 2024; Jimenez et al., 2024), and logical reasoning (Tafjord et al., 2021; Saparov and He, 2023). Nonetheless, their

performance tends to diminish in dynamic scenarios, such as Web navigation (Zhou et al., 2024b; Yao et al., 2023a) and Open-ended environments (Wang et al., 2022; Shridhar et al., 2021), which require robust reasoning capabilities of the agents.

A key contributing factor to language agents’ efficiency in long tasks is the notion of memory (Sumers et al., 2023). The recent approaches (Majumder et al., 2023; Zhao et al., 2023a) guided the agent to store reflections on their experience of solving a task (Shinn et al., 2023) in memory, and then to retrieve these to improve future attempts (Yao et al., 2023c). The use of verbal refinements rather than updating model parameters, these techniques are more flexible than conventional Reinforcement Learning (RL) methods. However, this memory module often lacks a retrieval mechanism. Additionally, complex tasks that cannot be solved in a single attempt also require effective planning and goal decomposition mechanisms (we will explain this in more detail later in this paper).

In this paper, we take a close look at the memory utilization and planning capabilities of the language agent. More concretely, we propose **STEP** - a novel framework for Stepwise Planning which consists of a *Planner*, an *Executor*, and an *Evaluator*. Upon receiving a task from the environment, the *Planner* decomposes it into manageable subtasks and retrieves relevant information from the *Memory*. After receiving messages, the *Executor* then generates action candidates, which are subsequently evaluated by the *Evaluator* (Madaan et al., 2023). Once an action is approved and sent back to the environment, the agent receives an observation and determines whether the subtask requires refinement. After completing an episode, the agent generates learning insights (Majumder et al., 2023), which are stored in the *Memory* for future attempts. A key aspect is that the *Planner* not only breaks down tasks but also dynamically distills relevant insights from previous attempts to enhance the current task

¹Our code and data will be available with publications.

trace. This iterative process maximizes the efficiency of the memory system, ensuring continuous learning and adaptation.

We evaluate STEP within ScienceWorld (Wang et al., 2022) - a dynamic, text-based environment designed to simulate complex scientific tasks. Our results demonstrate that STEP consistently outperforms state-of-the-art (SOTA) models, achieving an overall score of 67.4. The model successfully completes 12 out of 18 tasks, ranking first in 11 tasks.

2 Planning with Language Agents

Language agents are designed to interpret and execute tasks through natural language interaction (Sumers et al., 2023). A critical aspect of their functionality is planning, which involves decomposing complex tasks into manageable subtasks, executing actions iteratively, and refining strategies based on feedback. This structured approach is essential for achieving accuracy, leveraging prior experience, and adapting to complex environments. However, language agents encounter significant limitations in logical reasoning, a fundamental requirement for effective planning (Liu et al., 2023b; Wan et al., 2024). These agents are inherently probabilistic systems designed to predict text sequences rather than execute formal logic or computation. Their autoregressive nature, which generates outputs token-by-token based on prior context, further constrains their ability to prioritize logical validity over coherence (Arkoudas, 2023). As a result, they exhibit difficulties in handling tasks that demand precise reasoning and planning, particularly in out-of-distribution or computationally complex situations.

For a broader background please see Appendix A. Here we provide a brief overview of existing works on planning with language agents. Recent research has explored the use of LLMs as planners in various approaches.

LLM+P (Liu et al., 2023a) and LLM-DP (Dagan et al., 2023) utilize LLMs to translate natural language descriptions into the Planning Domain Definition Language (PDDL), enabling classical planners to derive solutions. While symbolic reasoning guarantees finding optimal solutions, it often requires significant human effort for language conversion (Huang et al., 2024), and the assumption of perfect observation of all object states may not hold in dynamic environments. Given the numerous tra-

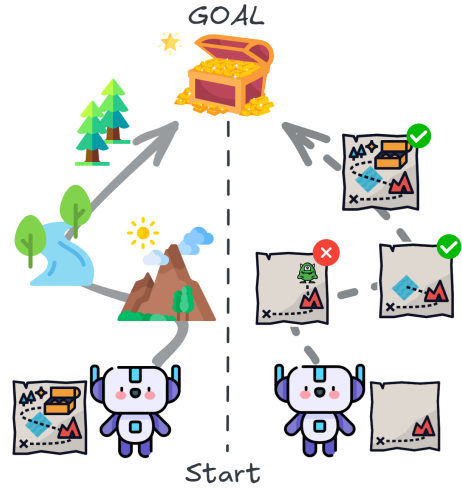


Figure 1: **Planning methods.** (LEFT) plan-from-the-start. (RIGHT) plan-on-the-go.

jectories available to achieve a goal, work on expanding multi-plans in tree structures has been explored. Tree of Thoughts (ToT) (Yao et al., 2023b), which employs conventional BFS/DFS for optimal pathfinding, RAP (Hao et al., 2023) and LLM-MCTS (Zhao et al., 2023b) leverage the Monte Carlo Tree Search (MCTS) algorithm for searching. However, these methods are computationally exhaustive, and using LLMs as a world model to evaluate plans becomes impractical, particularly in complex, open-ended environments.

SOTAs in LLM suggest a different planning method, where instead of tuning the model’s parameters (Schuster et al., 2022; Lin et al., 2023), adjusting the prompts proves advantageous. Two primary schools of thought in planning are “plan-from-the-start” and “plan-on-the-go” (see Figure 1). A representative of the former, Plan-and-Solve (Wang et al., 2023b), builds on Zero-shot Chain of Thought (CoT) (Kojima et al., 2023) by transitioning from “Let’s think step-by-step” to “Let’s first make a plan” and “Let’s carry out the plan”. While this pre-planning strategy offers a straightforward approach of solving task (Shen et al., 2023; Singh et al., 2022), it heavily relies on accurate decomposition problems into simpler subproblems at once (Zhou et al., 2023). Moreover, predefining a plan before interacting with the environment can result in unrealistic subtasks, requiring adjustments during deployment.

In contrast, “plan-on-the-go” methods perform better in dynamic environments, where subtasks are revealed one by one during the deploying process. This approach allows agents to improve through

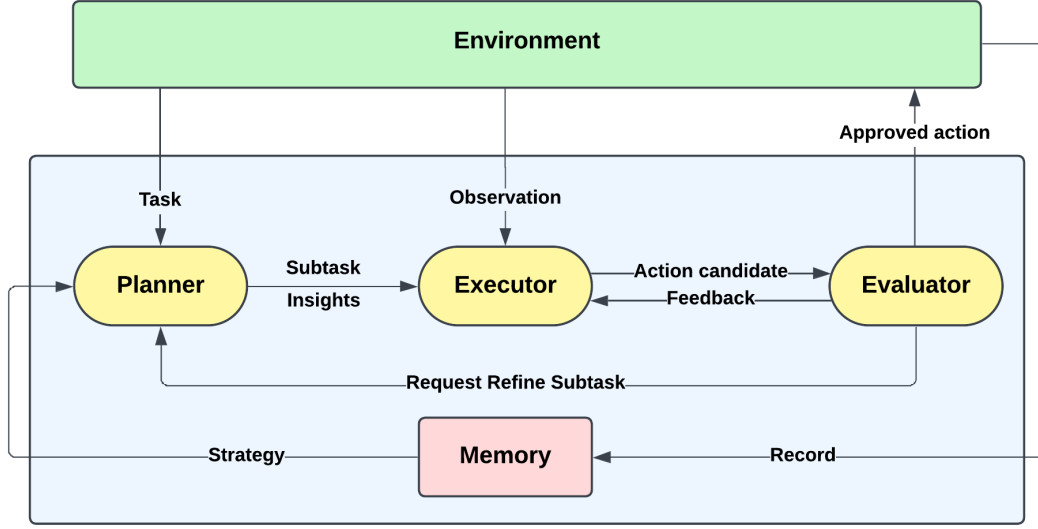


Figure 2: The architecture of **STEP**. (1) *Planner* receives the task and generates achievable subtasks and relevant insights, (2) *Executor* creates action candidates based on the generated subtasks and insights, (3) *Evaluator* assesses these action candidates for their alignment, and (4) *Memory* stores the experience for future use.

feedback-driven retries, without external supervision. ReAct (Yao et al., 2023c) interleaves actions and thoughts for robust planning, while Reflexion (Shinn et al., 2023) adds a verbal reflection after each attempt, showing the self-learning capability of language agents. Building on this, the CLIN (Majumder et al., 2023) model introduces a new type of insight called causal abstractions, which determine whether “X is necessary for Y”. At each step, the agent accesses a lengthy memory log to determine its next action. While experiments show good adaptability, accessing the entire memory at once may cause the agent to incorrectly prioritize subtasks and risk confusion or overlook key insights. This lack of a transfer learning mechanism (Zhao et al., 2023a) limits its ability to fully leverage learning across different tasks or environments. Building on the CLIN model, we introduce STEP which can maintain task order while leveraging distilled in-context insights from memory, ensuring more accuracy in task execution.

The question of whether language agents can serve as effective planners remains a prominent topic in the Generative AI field. Despite the notable advancements in language agents, Kambhampati et al. (2024) argue that LLMs cannot independently perform planning tasks. Their auto-regressive nature limits LLMs from generating fully executable plans or conducting self-verification, which is essential for reasoning and planning. Instead, the au-

thors propose the LLM-Modulo framework, where LLMs contribute by generating candidate plans, while external verifiers refine and validate these plans for accuracy. We agree with this perspective, acknowledging that language agents cannot act as autonomous planners. However, they can serve as stepwise planners, iteratively learning through trial and error. By adapting based on feedback from the external environment, language agents can refine their decision-making process and improve over time, enhancing their planning capabilities within a controlled framework.

3 Stepwise Planning Language Agent

We evaluate our model in simulated text-based environments, represented as a partially observable Markov decision process (POMDP). The agents are required to perform sequential actions to accomplish a specific task M . Our setup allows an agent to attempt a task multiple times, ranging from I to K . Each attempt k consists of multiple trials (i.e. episodes), and each trial T comprises a total of t steps. At each step, after issuing an action, the agent receives feedback on environmental changes through observations o and rewards r , indicating its performance. After each trial, we enable the agent to make a reflection, generating and storing learning insights in memory for future trials. During the deployment process, STEP consists of three components: *Planner*, *Executor*, and *Evaluator*. At

Algorithm 1 Planning with STEP

```
1: Task:  $M$ , Memory:  $S, St$ 
2: for  $k \in 1, \dots, K$  do
3:   Initialize Trial  $T, t, s' = rule(S_{k-1})$ 
4:   while  $t < \text{max. total steps}$  or task not complete do
5:     if done then
6:        $m, s = \text{Planner}(M, St, S_{k-1}, a_{<t}, o_{<t}, q)$ 
7:        $q.append(m)$ 
8:     else
9:        $m, s = \text{Planner}(M, St, m, S_{k-1}, a_{<t}, o_{<t})$ 
10:    end if
11:     $t_0 = t$ 
12:    while  $t < \text{max. sub steps} + t_0$  or not done do
13:       $g_t, a_t = \text{Executor}(m, s, a_{<t}, o_{<t}, f)$ 
14:       $f, done = \text{Evaluator}(m, s', a_{\leq t}, o_{<t})$ 
15:      if not  $f$  then
16:         $r_t, o_t = \text{Simulator}(T_{<t}, a_t)$ 
17:         $T_{<t+1} = T_{<t} + (r_t, a_t, o_t, g_t)$ 
18:      end if
19:    end while
20:  end while
21:  Final reward  $r_k = r_t$ 
22:   $S_k, St = \text{Memory-Generator}(S_{<k}, St, T_k, r_k)$ 
23: end for
```

the end of each trial, STEP updates its *Memory* to facilitate continued learning (See Figure 2).

Planner is the strategic component of the framework, responsible for breaking down the current task into manageable subtasks, leveraging a frozen LLM (a model with fixed parameters) to assist in generating the next steps (Algorithm 1. lines 5-10). At step t , the *Planner* receives the main task M , the suggested strategy St from the past, and the history of the current trial (a sequence of actions and observations: $\{a_1, o_1, a_2, o_2, \dots, a_t, o_t\}$). Based on the status of the previous subtask m , the *Planner* either refines it or derives a new subtask with the list of successful subtasks q to advance the agent toward the main task. Additionally, the *Planner* also retrieves relevant insights s from the learning summary of the previous attempt S_{k-1} , which prevents agent access to irrelevant information and focusing on the subtask itself (Zhao et al., 2023a).

Executor is the language agent’s implementer, tasked with generating appropriate actions to execute the objectives set by the *Planner* (Algorithm 1. line 13). Following the ReAct concept, the *Executor* is required to derive the next rationale g_t and action a_t given the current subtask m and relevant insight s from the *Planner*. Another modification from the CLIN model is that while the *Executor* is still provided with the sequence of actions and observations, it no longer receives the rationales of previous actions. This is because, at different

steps, the *Executor* may have a different subtask m , leading to a different motivation for actions. This setup enhances the module’s purpose in task performance, while the *Planner* is the sole component managing the overall task flow, the *Executor* has access only to its current subtask. Thus, within the *Executor*, subtasks are converted into concrete actions that can interact with the environment.

Evaluator serves as a quality control mechanism, assessing the action generated by the *Executor* before they are executed in the environment (Algorithm 1. lines 12-19). While it has been argued that relying solely on internal knowledge for refinement may reduce the performance of language agents (Kambhampati et al., 2024; Gou et al., 2024a; Valmeekam et al., 2023), our *Evaluator* does not select the best action candidate based on its own judgment (Madaan et al., 2023). Instead of evaluating the alignment between the action candidate and the task, it focuses on how well the action aligns with the rules s' (e.g. “X does NOT contribute to Y”) from S_{k-1} . While the dynamic nature of the environment makes it nearly impossible to construct a world model upfront in “plan-from-the-start” approaches, the “plan-on-the-go” strategy takes advantage of this flexibility. After exploring the environment, the *Executor* resembles as simplified world model, which assesses the action candidates. Based on these assessments, the *Evaluator* either sends a feedback f to the *Executor* for refinement or approves it for execution. It also monitors the completion status of subtasks, issuing refinement requests when necessary to avoid exhaustive exploration.

Finally, **Memory** serves as a critical component in ensuring the agent’s continual learning process (Gou et al., 2024b; Shinn et al., 2023; Zhou et al., 2024a) (Algorithm 1. line 22). The *Memory* is structured into two primary components: the list of insights S_k (which is inherited from CLIN) and the suggested strategy St . At the end of each trial k , the agent reflects on its performance T_k based on final reward r_k , generates new strategy St , and updates S_k from previous trials $S_{<k}$. The notion of insights is well-established in recent research and can encompass pairs of actions and observations (Schuster et al., 2022), causal verbal reflections (Shinn et al., 2023), or human feedback (Ouyang et al., 2022). In this work, we adopt the causal abstraction framework proposed by Majumder et al. (2023), which evaluates how action X influences action Y . This relationship can yield useful in-

sights, such as “ X is necessary for Y ,” or identify errors, such as “ X may not contribute to Y .”

Although these insights are valuable for guiding the agent’s execution process, a large information may impair retrieval performance, leading the agent to deviate from the intended task sequence. To mitigate this issue and maintain system efficiency, the *Planner* is provided with a suggested strategy St from the most recent trial, enabling it to accurately determine the next subtask and forward relevant insights to the *Executor*.

4 Experiments

4.1 Experimental Setup

Task Environment. To assess STEP, we chose ScienceWorld (Wang et al., 2022) ², a text-based interactive environment that simulates elementary science tasks across 10 interconnected locations, including settings like a living room, workshop, and art studio. The environment features 200 object types, such as devices, substances, plants, and animals, and allows for 25 high-level actions (see Appendix B for details on action space). When combined with objects, this results in approximately 200,000 possible action-object combinations per step. The environment is dynamically populated with various object arrangements to prevent memorization and promote adaptability. Agents are challenged to perform scientifically reasoned actions, such as testing electrical conductivity or observing biological life stages. Tasks in ScienceWorld vary in complexity, with short tasks requiring fewer than 37 steps to complete, while longer tasks require more than 37 steps.

Evaluation Protocol. Agents are evaluated based on their ability to complete tasks, with positive scores ranging from 0 to 100. Each task is divided into required steps and optional sub-goals that guide the agent toward the final objective. Performance is primarily measured by how well the agent completes these goals, often requiring it to “focus on” the correct object to progress. Critical mistakes, such as focusing on unapproved objects, result in a penalty of -100 and cause the task to reset. In the failure event, the highest positive score achieved before the mistake is recorded to reflect the agent’s progress. Otherwise, the final score of each runtime is used, whether the agent successfully completes the task or reaches the maximum

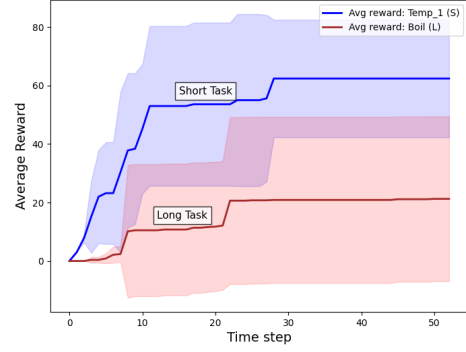


Figure 3: Performance example of STEP in Long and Short Tasks.

step limit per task.

Baselines agents. In addition to the baseline methods evaluated in the ScienceWorld paper, which included three RL agents - DRRN (He et al., 2016), KGA2C (Ammanabrolu and Hausknecht, 2020), and CALM (Schuster et al., 2022) - as well as experiments derived by Lin et al. (2023) involving three generative language agents - SayCan (Ahn et al., 2022), ReAct (Yao et al., 2023c), and Reflexion (Shinn et al., 2023) - we also incorporate the CLIN ADAPT version (Majumder et al., 2023). For further details, please refer to Appendix A.

Configurations. We use gpt-4o-mini as the base language model for running both CLIN and STEP. Tasks include 9 short tasks (denoted as S) and 9 long tasks (denoted as L). For short tasks, the maximum step limit is set to 37, while for long tasks it is 70. Each task run consists of 5 episodes, where the first episode represents the agent having no prior knowledge of the environment, and the final episode occurs after the agent has learned and updated its memory to adapt to the task. The highest score achieved across these episodes is taken as the score for the current runtime.

4.2 Main Results

Figure 3 presents the model’s performance for a short task Temp₁ and a long task Boil. The solid lines represent average rewards while the shaded areas capture the variability in performance. Inherited from CLIN, STEP demonstrates strong performance. For the short task, the model shows rapid improvement, reaching an average reward close to 60 within approximately 30 steps, with a noticeable reduction in variability. In contrast, the long task presents slower improvement, with the model’s average reward gradually increasing and

²<https://sciworld.apps.allenai.org/explore>

Task	Type	RL Agents			Generative Language Agents				STEP
		DRRN	KGA2C	CALM	SayCan	ReAct	Reflexion	CLIN*	
Temp ₁	S	6.6	6.0	1.0	26.4	7.2	5.9	9.0	63.0
Temp ₂	S	5.5	11.0	1.0	8.0	6.1	28.6	83.3	62.7
Pick&Place ₁	S	15.0	18.0	10.0	22.9	26.7	64.9	100.0	100.0
Pick&Place ₂	S	21.7	16.0	10.0	20.9	53.3	16.4	69.3	100.0
Chemistry ₁	S	15.8	17.0	3.0	47.8	51.0	70.4	55.3	61.0
Chemistry ₂	S	26.7	19.0	6.0	39.3	58.9	70.7	100.0	100.0
Lifespan ₁	S	50.0	43.0	6.0	80.0	60.0	100.0	100.0	100.0
Lifespan ₂	S	50.0	32.0	10.0	67.5	67.5	84.4	100.0	100.0
Biology ₁	S	8.0	10.0	0.0	16.0	8.0	8.0	28.0	32.2
Boil	L	3.5	0.0	0.0	33.1	3.5	4.2	4.0	21.5
Freeze	L	0.0	4.0	0.0	3.9	7.8	7.8	32.3	50.9
GrowPlant	L	8.0	6.0	2.0	9.9	9.1	7.3	30.3	71.5
GrowFruit	L	14.3	11.0	4.0	13.9	18.6	13.0	19.3	14.0
Biology ₂	L	21.0	5.0	4.0	20.9	27.7	2.6	59.3	46.5
Force	L	10.0	4.0	0.0	21.9	40.5	50.6	73.3	80.0
Friction	L	10.0	4.0	3.0	32.3	44.0	100.0	56.7	73.3
Genetics ₁	L	16.8	11.0	2.0	67.5	25.7	50.9	69.8	84.2
Genetics ₂	L	17.0	11.0	2.0	59.5	16.8	23.7	39.0	51.8
S		22.1	19.1	5.2	36.5	37.6	49.9	71.7	79.9
L		11.2	6.2	1.9	29.2	21.5	29.2	42.7	54.9
All		16.7	12.7	3.6	32.9	29.6	39.4	57.2	67.4

Table 1: Comparison of STEP with baselines in ScienceWorld environment. *We use the same backbone LLM, gpt-4o-mini, for both CLIN and STEP experiments for fairness. Type S and L denote short and long tasks, respectively.

peaking around 20 after 50 steps, showing higher variability throughout. This suggests that the model adapts more efficiently and stably in the short task while facing greater difficulty and inconsistency in the longer task. See Appendix C for a sample from STEP task solving.

Next, we will compare STEP with other state-of-the-art (SOTA) agents, as shown in Figure 6 of Appendix A. In particular, we will conduct further experiments to compare it with CLIN, the predecessor of STEP, to evaluate performance improvements.

4.2.1 STEP outperform SOTA Agents

Table 1 compares the performance of various agents across 18 tasks in the ScienceWorld environment. The results highlight the superior performance of LLM-based methods (Generative Language Agents) over conventional RL agents due to their advanced generalization abilities, though they come with higher deployment costs. Among the RL agents, DRRN achieves the highest overall score of 16.7, which is still significantly lower than the scores of all the generative language agents, with the lowest among them being ReAct at 29.6. Reflexion demonstrates a strong capability by utilizing episodic memory to retain learning insights

across trials, showing more than a 10-point improvement over SayCan and ReAct in short tasks. Building upon Reflexion’s memory-based advantages, CLIN benefits from causal abstraction insights, which significantly boosts its performance to a 57.2 overall score, demonstrating its effectiveness in both task types.

Notably, STEP consistently outperforms all SOTA models across both short and long tasks, achieving 67.4 points overall. In short tasks, STEP achieves an impressive 79.9, representing an 11.4% increase over its predecessor. In long tasks, STEP further demonstrates its ability to handle more complex challenges, with a score of 54.9, marking a 28.6% improvement over CLIN. Analyzing individual tasks, STEP ranks first in 11 out of 18 tasks, clearly showcasing its dominant performance. For example, in the short task Temp₁, STEP significantly boosts the second highest score from SayCan’s 26.4 to an impressive 63.0. In the long task GrowPlant, it doubles the score of 30.3 in CLIN to 71.5. This substantial leap in performance across individual tasks emphasizes STEP’s capacity to generalize well and adapt to a wide range of environments.

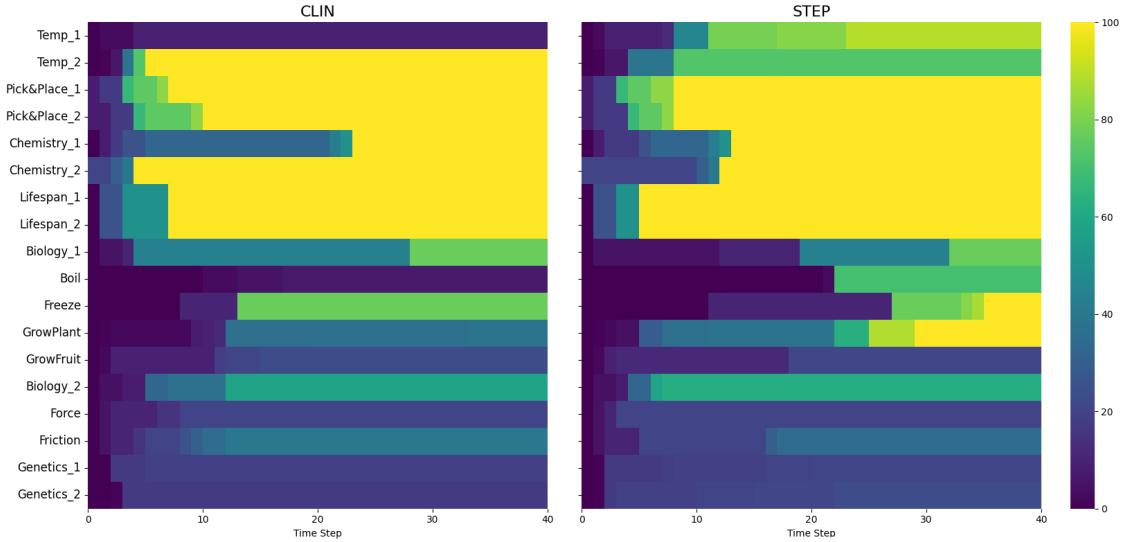


Figure 4: Task performance comparison between STEP and CLIN.

4.2.2 STEP outperforms CLIN in continual learning

Next, we compare the performance between STEP and CLIN across all tasks (see Figure 4), reporting the best traces while excluding false-positive cases where the agent unintentionally “cheats” the task (e.g. in the Friction task, the agent might randomly focus on an inclined plane and complete the task by chance, without adhering to the intended experimental procedure). In short tasks, both CLIN and STEP exhibit strong performance, as indicated by the prominent yellow regions for tasks such as Pick&Place, Chemistry and Lifespan. However, a key difference arises in Temp₁, where CLIN struggles, achieving only 10 points overall. In contrast, STEP shows a gradual progression, steadily achieving subgoals and eventually reaching nearly 90 points. This demonstrates STEP’s capability to break down tasks and make incremental progress, where CLIN appears to fall short.

The performance gap widens further in long tasks, where STEP consistently outperforms CLIN. CLIN shows difficulties in tasks such as Boil, Freeze, and GrowPlant, marked by darker shades in the heatmap, indicating poorer performance. STEP, on the other hand, demonstrates significant improvements in these complex tasks, with more yellow regions, suggesting higher scores and better task completion. A potential explanation for this distinction is that CLIN completes all tasks within 30 steps, as shown by the cutoff in its heatmap, suggesting faster but possibly incomplete learning. Conversely, STEP continues learning be-

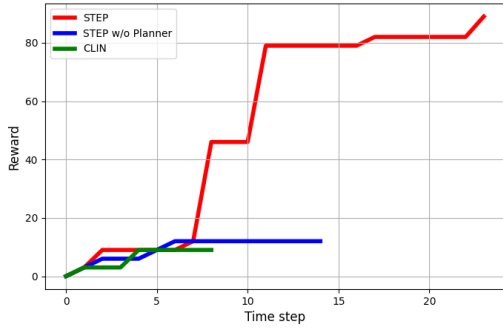
yond that, which allows it to achieve superior overall performance and higher scores, particularly in tasks where CLIN has plateaued. For example, in Freeze, while CLIN initially learns quickly, achieving around 80 points within the first 14 steps, it subsequently halts progress or becomes stuck in a loop. In contrast, after experiencing stagnation up to 15 steps without any improvement, STEP rapidly peaks at the 27th and successfully completes the task at the 36th step. This suggests that STEP has a greater capacity for long-term learning and task improvement, particularly in more complex scenarios, where CLIN’s performance tends to stall.

Task	STEP w/o Planner	
	Δ avg score (\downarrow)	%succ. rate (\downarrow)
S	10.9	11.1
L	11.0	22.2
All	10.9	16.7

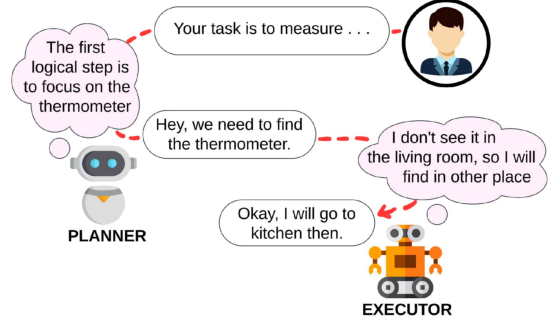
Table 2: Average performance reduction without the Planner on short (S), long (L), and combined (All) tasks.

4.2.3 The importance of Planner

Ablation experiment. To evaluate the necessity of the *Planner* component, we conducted ablation experiments on STEP by removing the *Planner* and retaining only the *Executor*, *Evaluator*, and *Memory* modules (see Figure 2). In this configuration, the *Executor* directly receives tasks from the environment and retrieves information from *Memory* without any distillation or task refinement. Moreover, the absence of the *Planner* eliminates the



(a)



(b)

Figure 5: (a) Comparison of STEP, STEP w/o Planner and CLIN in Temp₁. (b) Planner and Executor interaction.

refinement of subtasks that are typically performed by the *Evaluator*.

As presented in Table 2, the results demonstrate a significant drop in performance. Specifically, the removal of the *Planner* leads to a reduction in the average score by 10.9 points in short tasks and 11.0 points in long tasks. Furthermore, the success rate decreases by 11.1% for short tasks and 22.2% for long tasks, culminating in an overall decline of 10.9 points in average score and 16.7% in success rate across all tasks. Notably, the average score of the STEP w/o Planner model is approximately the same as that of CLIN, with both achieving around 57 points overall. This result was expected, as the ablation model lacks the critical *Planner* component, making it architecturally similar to CLIN. These findings also suggest that the addition of the *Evaluator* (which CLIN does not include) does not significantly enhance performance in the absence of the *Planner*.

A qualitative example. Figure 5a illustrates the optimal strategies employed by STEP, STEP w/o Planner, and CLIN during Task Temp₁, where the agent must measure the temperature of an unknown substance B. Although the location of substance B is known (in the living room), the agent must first locate a hidden thermometer. Without a planner, both CLIN and STEP w/o Planner instinctively head to the living room to interact with substance B, resulting in a -100 point penalty for skipping the necessary step of finding the thermometer. As reflected in the graph, this mistake leads to early termination for both models—CLIN is cut off at step 8 and STEP w/o Planner at step 14—with their rewards stagnating below 15 points due to incorrect task execution.

In contrast, STEP with *Planner* adheres to the

correct task order. The *Planner* isolates the subtask of finding the thermometer, ensuring the *Executor* focuses solely on the immediate task without being distracted by the known location of substance B (see Figure 5b). This isolation prevents the *Executor* from prematurely interacting with substance B and helps STEP avoid penalties. As reflected in the graph, STEP continues to progress smoothly, with its reward rapidly increasing and reaching a maximum of 90.

Additionally, we observed that without the *Planner*, the performance of the other models does not improve, even when *Memory* correctly suggests the location of the thermometer (in the kitchen). The situation worsens when the models receive hints related to substance B, such as “going to the living room is necessary to find substance B” or “focusing on substance B may contribute to the task.” Without the high-level task distillation and guidance provided by the *Planner*, these models are unable to effectively leverage the usefulness of information. This further underscores the *Planner*’s critical role in task decomposition and knowledge distillation, ensuring the agent follows the task order.

5 Conclusion

We present STEP, a simple yet effective framework that leverages LLM as a stepwise planner. STEP enhances memory utilization and task sequence recognition, achieving state-of-the-art performance in the ScienceWorld benchmark. By showcasing the potential of STEP, our research contributes to the ongoing exploration and development of language agents as planners.

6 Limitations

Poor subtask generation. The agent can easily deviate from the intended task if the initial subtask goes off track. The *Executor* relies entirely on the *Planner*’s guidance, while the *Planner* depends on the *Executor*’s exploration to understand the environment. This interdependency can lead to significant problems if the *Planner* generates an irrelevant subtask. In task Biology, the agent is required to “*focus on the 3 life stages of the wolf*”. Instead of guiding the agent to find a wolf outside the house, the *Planner* incorrectly decides that painting a representation of a wolf is a valid solution: “*Since I am in an art studio, I can use the paint to create representations of the wolf’s life stages.*” The *Planner* then directs the *Executor* to gather materials, mix colors, and attempt to draw the wolf.

Once the *Planner* locks onto an incorrect high-level subtask, it becomes increasingly difficult for the agent to revise its approach. As a result, it falls into a repetitive loop of failure, with the *Executor* never leaving the house to discover the real wolf. The only potential way to break this loop is through a refinement mechanism; however, in this case, the mechanism fails to detect the irrelevance of the subtask and continues allowing the agent to explore the painting activity. Instead of exploring horizontally, the agent dives vertically into the wrong path.

Poor strategy generation. After each trial, a strategy is generated to inform future agents of the latest successful traces. Whether provided with a list of completed subtasks or a full action trace, the model struggles to eliminate unnecessary elements. For instance, in the Force task, where the agent must “*determine which of the two inclined planes has the steepest angle*”, the agent first needs to locate the inclined planes in the workshop. After extensive exploration—navigating to the art studio, hallway, outside, and then back to the hallway—the agent finally finds the workshop and continues with the task. However, at the end of the episode, the agent fails to eliminate redundant looping traces, such as repeatedly moving to the hallway without gaining useful information, and instead groups these actions into the vague category of “*Exploring the environment.*”

Balancing strategy abstraction presents significant challenges. Strategies that are too detailed closely resemble action traces, causing future agents to simply replicate past actions without con-

sidering variations in context. In contrast, overly abstract strategies result in unachievable goals, as they lack the necessary steps for effective task completion. To mitigate this, our approach attempts to retain actions that gain rewards from the environment, then prompt LLMs to identify which remaining actions are essential, before abstracting the new action trace. However, this method is still far from optimal and highlights a gap for future research.

References

- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *Preprint*, arXiv:2204.01691.
- Prithviraj Ammanabrolu and Matthew Hausknecht. 2020. Graph constrained reinforcement learning for natural language action spaces. *Preprint*, arXiv:2001.08837.
- Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, Beijing, China. Association for Computational Linguistics.
- Konstantine Arkoudas. 2023. Gpt-4 can’t reason. *Preprint*, arXiv:2308.03762.
- Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2023. Deep reinforcement learning from human preferences. *Preprint*, arXiv:1706.03741.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.
- Gautier Dagan, Frank Keller, and Alex Lascarides. 2023. Dynamic planning with a llm. *Preprint*, arXiv:2308.06391.

672	Jiazhan Feng, Ruochen Xu, Junheng Hao, Hiteshi	Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yu-	728
673	Sharma, Yelong Shen, Dongyan Zhao, and Weizhu	taka Matsuo, and Yusuke Iwasawa. 2023. Large	729
674	Chen. 2023. Language models can be logical solvers.	language models are zero-shot reasoners. <i>Preprint</i> ,	730
675	<i>Preprint</i> , arXiv:2311.06158.	arXiv:2205.11916.	731
676	Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull	Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brah-	732
677	Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and	man, Shiyu Huang, Chandra Bhagavatula, Prithviraj	733
678	Honglak Lee. 2024. Autoguide: Automated genera-	Ammanabrolu, Yejin Choi, and Xiang Ren. 2023.	734
679	tion and selection of state-aware guidelines for large	Swiftsage: A generative agent with fast and slow	735
680	language model agents. <i>Preprint</i> , arXiv:2403.08978.	thinking for complex interactive tasks. <i>Preprint</i> ,	736
681	Majid Ghasemi, Amir Hossein Moosavi, Ibrahim	arXiv:2305.17390.	737
682	Sorkhoh, Anjali Agrawal, Fadi Alzhouri, and Dariush	Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu,	738
683	Ebrahimi. 2024. An introduction to reinforcement	Shiqi Zhang, Joydeep Biswas, and Peter Stone.	739
684	learning: Fundamental concepts and practical appli-	2023a. Llm+p: Empowering large language mod-	740
685	cations. <i>Preprint</i> , arXiv:2408.07712.	els with optimal planning proficiency. <i>Preprint</i> ,	741
686	Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong	arXiv:2304.11477.	742
687	Shen, Yujiu Yang, Nan Duan, and Weizhu Chen.	Hanmeng Liu, Ruoxi Ning, Zhiyang Teng, Jian Liu, Qiji	743
688	2024a. Critic: Large language models can self-	Zhou, and Yue Zhang. 2023b. Evaluating the logi-	744
689	correct with tool-interactive critiquing. <i>Preprint</i> ,	cal reasoning ability of chatgpt and gpt-4. <i>Preprint</i> ,	745
690	arXiv:2305.11738.	arXiv:2304.03439.	746
691	Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen,	Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler	747
692	Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu	Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon,	748
693	Chen. 2024b. Tora: A tool-integrated reasoning	Nouha Dziri, Shrimai Prabhumoye, Yiming Yang,	749
694	agent for mathematical problem solving. <i>Preprint</i> ,	Shashank Gupta, Bodhisattwa Prasad Majumder,	750
695	arXiv:2309.17452.	Katherine Hermann, Sean Welleck, Amir Yazdan-	751
696	Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong,	bakhsh, and Peter Clark. 2023. Self-refine: It-	752
697	Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023.	erative refinement with self-feedback. <i>Preprint</i> ,	753
698	Reasoning with language model is planning with	arXiv:2303.17651.	754
699	world model. <i>Preprint</i> , arXiv:2305.14992.	Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra,	755
700	Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Li-	Peter Jansen, Oyvind Tafjord, Niket Tandon,	756
701	hong Li, Li Deng, and Mari Ostendorf. 2016. Deep	Li Zhang, Chris Callison-Burch, and Peter Clark.	757
702	reinforcement learning with a natural language action	2023. Clin: A continually learning language agent	758
703	space. <i>Preprint</i> , arXiv:1511.04636.	for rapid task adaptation and generalization. <i>arXiv</i>	759
704	Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul	<i>preprint arXiv:2310.10134</i> .	760
705	Arora, Steven Basart, Eric Tang, Dawn Song, and	Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Car-	761
706	Jacob Steinhardt. 2021. Measuring mathematical	roll L. Wainwright, Pamela Mishkin, Chong Zhang,	762
707	problem solving with the math dataset. <i>Preprint</i> ,	Sandhini Agarwal, Katarina Slama, Alex Ray, John	763
708	arXiv:2103.03874.	Schulman, Jacob Hilton, Fraser Kelton, Luke Miller,	764
709	Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei	Maddie Simens, Amanda Askell, Peter Welinder,	765
710	Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruim-	Paul Christiano, Jan Leike, and Ryan Lowe. 2022.	766
711	ing Tang, and Enhong Chen. 2024. Understanding	Training language models to follow instructions with	767
712	the planning of llm agents: A survey. <i>Preprint</i> ,	human feedback. <i>Preprint</i> , arXiv:2203.02155.	768
713	arXiv:2402.02716.	Abulhair Saparov and He He. 2023. Language models	769
714	Carlos E. Jimenez, John Yang, Alexander Wettig,	are greedy reasoners: A systematic formal analysis	770
715	Shunyu Yao, Kexin Pei, Ofir Press, and Karthik	of chain-of-thought. <i>Preprint</i> , arXiv:2210.01240.	771
716	Narasimhan. 2024. Swe-bench: Can language mod-	Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani,	772
717	els resolve real-world github issues? <i>Preprint</i> ,	Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metz-	773
718	arXiv:2310.06770.	zler. 2022. Confident adaptive language modeling.	774
719	Subbarao Kambhampati, Karthik Valmeekam, Lin	<i>Preprint</i> , arXiv:2207.07061.	775
720	Guan, Mudit Verma, Kaya Stechly, Siddhant Bham-	Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li,	776
721	bri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't	Weiming Lu, and Yueting Zhuang. 2023. Hugging-	777
722	plan, but can help planning in llm-modulo frame-	gpt: Solving ai tasks with chatgpt and its friends in	778
723	works. <i>Preprint</i> , arXiv:2402.01817.	hugging face. <i>Preprint</i> , arXiv:2303.17580.	779
724	Jikun Kang, Romain Laroche, Xingdi Yuan, Adam	Noah Shinn, Federico Cassano, Edward Berman, Ash-	780
725	Trischler, Xue Liu, and Jie Fu. 2024. Think before	win Gopinath, Karthik Narasimhan, and Shunyu Yao.	781
726	you act: Decision transformers with working mem-	2023. Reflexion: Language agents with verbal rein-	782
727	ory. <i>Preprint</i> , arXiv:2305.16338.	forcement learning. <i>Preprint</i> , arXiv:2303.11366.	783

784	Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté,	Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu	838
785	Yonatan Bisk, Adam Trischler, and Matthew	Lin, Yong-Jin Liu, and Gao Huang. 2023a. Ex-	839
786	Hausknecht. 2021. Alfworld: Aligning text and	pel: Llm agents are experiential learners. <i>Preprint</i> ,	840
787	embodied environments for interactive learning.	arXiv:2308.10144.	841
788	<i>Preprint</i> , arXiv:2010.03768.		
789	Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit	Zirui Zhao, Wee Sun Lee, and David Hsu. 2023b.	842
790	Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox,	Large language models as commonsense knowl-	843
791	Jesse Thomason, and Animesh Garg. 2022. Prog-	edge for large-scale task planning. <i>Preprint</i> ,	844
792	prompt: Generating situated robot task plans using	arXiv:2305.14078.	845
793	large language models. <i>Preprint</i> , arXiv:2209.11302.		
794	Theodore R Sumers, Shunyu Yao, Karthik Narasimhan,	Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman,	846
795	and Thomas L Griffiths. 2023. Cognitive ar-	Haohan Wang, and Yu-Xiong Wang. 2024a. Lan-	847
796	chitectures for language agents. <i>arXiv preprint</i>	guage agent tree search unifies reasoning act-	848
797	arXiv:2309.02427.	ing and planning in language models. <i>Preprint</i> ,	849
		arXiv:2310.04406.	850
798	Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter	Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei,	851
799	Clark. 2021. Proofwriter: Generating implications,	Nathan Scales, Xuezhi Wang, Dale Schuurmans,	852
800	proofs, and abductive statements over natural lan-	Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi.	853
801	guage. <i>Preprint</i> , arXiv:2012.13048.	2023. Least-to-most prompting enables complex	854
		reasoning in large language models. <i>Preprint</i> ,	855
		arXiv:2205.10625.	856
802	Karthik Valmeekam, Matthew Marquez, and Subbarao	Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou,	857
803	Kambhampati. 2023. Can large language models	Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue	858
804	really improve by self-critiquing their own plans?	Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Gra-	859
805	<i>Preprint</i> , arXiv:2310.08118.	ham Neubig. 2024b. Webarena: A realistic web en-	860
		vironment for building autonomous agents. <i>Preprint</i> ,	861
806	Yuxuan Wan, Wenxuan Wang, Yiliu Yang, Youliang	arXiv:2307.13854.	862
807	Yuan, Jen tse Huang, Pinjia He, Wenxiang Jiao, and		
808	Michael R. Lyu. 2024. Logicasker: Evaluating and	Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu,	863
809	improving the logical reasoning ability of large lan-	Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani	864
810	guage models. <i>Preprint</i> , arXiv:2401.00757.	Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon	865
811	Guangzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Man-	Brunner, Chen Gong, Thong Hoang, Armel Randy	866
812	dlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and	Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kad-	867
813	Anima Anandkumar. 2023a. Voyager: An open-	dour, Ming Xu, Zhihan Zhang, Prateek Yadav, Na-	868
814	ended embodied agent with large language models.	man Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu,	869
815	<i>Preprint</i> , arXiv:2305.16291.	Qian Liu, Zijian Wang, David Lo, Binyuan Hui,	870
		Niklas Muennighoff, Daniel Fried, Xiaoning Du,	871
816	Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu,	Harm de Vries, and Leandro Von Werra. 2024. Big-	872
817	Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim.	codebench: Benchmarking code generation with	873
818	2023b. Plan-and-solve prompting: Improving zero-	diverse function calls and complex instructions.	874
819	shot chain-of-thought reasoning by large language	<i>Preprint</i> , arXiv:2406.15877.	875
820	models. <i>Preprint</i> , arXiv:2305.04091.		
821	Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and	A General Background	876
822	Prithviraj Ammanabrolu. 2022. Scienceworld: Is	A.1 Reinforcement Learning (RL) Agents	877
823	your agent smarter than a 5th grader? <i>Preprint</i> ,	An RL agent is an autonomous system that learns	878
824	arXiv:2203.07540.	by interacting with its environment, making deci-	879
825	Shunyu Yao, Howard Chen, John Yang, and Karthik	sions, and receiving feedback in the form of re-	880
826	Narasimhan. 2023a. Webshop: Towards scalable	wards to maximize long-term success (Ghasemi	881
827	real-world web interaction with grounded language	et al., 2024). We provide a brief review of three	882
828	agents. <i>Preprint</i> , arXiv:2207.01206.	works used in our experiment section as baselines:	883
829	Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafra-	DRRN (Deep Reinforcement Relevance Net-	884
830	Thomas L. Griffiths, Yuan Cao, and Karthik	work) (He et al., 2016) is designed to handle state	885
831	Narasimhan. 2023b. Tree of thoughts: Deliber-	and action spaces, represented in natural language	886
832	ate problem solving with large language models.	format. Instead of using raw text, DRRN uses sepa-	887
833	<i>Preprint</i> , arXiv:2305.10601.	rate neural networks to encode both the game states	888
834	Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak	and possible actions into embedding vectors. Sub-	889
835	Shafraan, Karthik Narasimhan, and Yuan Cao. 2023c.	sequently, it combines them to approximate the	890
836	React: Synergizing reasoning and acting in language	optimal action which maximizes the reward. This	891
837	models. <i>Preprint</i> , arXiv:2210.03629.		

approach allows the agent to better understand and navigate environments with high-dimensional, unstructured data, showing that it captures meaning instead of memorizing text strings.

CALM (Contextual Action Language Model) (Schuster et al., 2022) is trained on human game-play to learn linguistic patterns and common actions, allowing it to generate a set of action candidates for each state. These candidates are then passed to a DRRN agent for re-ranking based on game rewards. By integrating human-like action prediction through a language model with reinforcement learning for assessment, CALM enhances the agent’s ability to navigate and interact in complex text-based environments, including unfamiliar situations.

KG-A2C (Knowledge Graph Advantage Actor Critic) (Ammanabrolu and Hausknecht, 2020) enhances the agent’s decision-making by constructing a dynamic knowledge graph during exploration. By leveraging the OpenIE technique (Angeli et al., 2015), it maps relationships between objects, locations, and actions, and dynamically updates this graph. KG-A2C helps the agent prune irrelevant actions and focus on the most relevant ones, improving navigation and decision-making in complex environments.

A.2 Language Agents

A language agent is an AI system designed for interaction with the external world and understanding natural language, extending the abilities of large language models (LLMs) by incorporating memory mechanisms and related action capabilities (Sumers et al., 2023).

A.2.1 Memory

Memory plays a critical role in language agents, as it enables them to store, retrieve, and process information over time. Sumers et al. (2023) categorized memory into short-term and long-term modules including episodic memory, semantic memory and procedural memory. Short-term memory (working memory) holds active information, such as observations from the environment or retrieved knowledge from long-term memory (Kang et al., 2024), necessary for immediate processing. As its name implies, working memory serves as a central hub, providing queries to the LLM, translating responses into actionable steps, and linking long-term memory with the current state for cohesive processing.

Episodic memory records the experiences and

events the agent has encountered. Like traditional reinforcement learning, where agents adjust their policy based solely on rewards, episodic memory improves learning by enabling more refined understanding and strategic planning, leveraging past successes and failures (Ouyang et al., 2022; Shinn et al., 2023; Gou et al., 2024b). Semantic memory holds factual information about the world and the agent’s identity, capabilities, and task context, acting like a system prompt. For instance, SayCan (Ahn et al., 2022) understands it controls a “physical robot” to execute real-world tasks described in natural language, while CRITIC (Gou et al., 2024a) knows its role is to handle free-form question answering or mathematical problems, with the ability to use tools like code interpreters and calculators to verify and refine solutions. Procedural memory comprises the LLM’s internal weights and predefined code. Unlike the flexible nature of software-based episodic or semantic memory, procedural memory functions like hardware, requiring precise design to initialize the agent, where any error could lead to bugs or unintended behavior. While some language agents, such as LOGIPT (Feng et al., 2023) and ToRA (Gou et al., 2024b), exhibit procedural learning, this usually occurs during fine-tuning phase, not in deployment.

To interact with the memory system, language agents use human-like internal actions - retrieval, learning, and reasoning. Retrieval actions move data from long-term to short-term memory, providing the agent with relevant information to handle tasks. This can be done by directly querying the LLM to retrieve insights based on the current context (Zhao et al., 2023a) or using a key-value method where the key relates to the common in state and the value provides guidance (Fu et al., 2024). Learning actions process observations and feedback from working memory, encoding them into long-term memory to enable continuous learning. For example, VOYAGER (Wang et al., 2023a), a Minecraft agent, retrieves skills from episodic memory, generates executable code, and stores new skills gained from interaction with the environment back into long-term memory. Reasoning actions are more complex, involving reading from and writing to working memory. The LLM integrates the task, retrieved insight, probable solutions, and feedback from humans (Christiano et al., 2023; Ouyang et al., 2022), self-reflection (Madaan et al., 2023; Shinn et al., 2023; Majumder et al., 2023) or external tools (Gou et al., 2024a). If the solution does

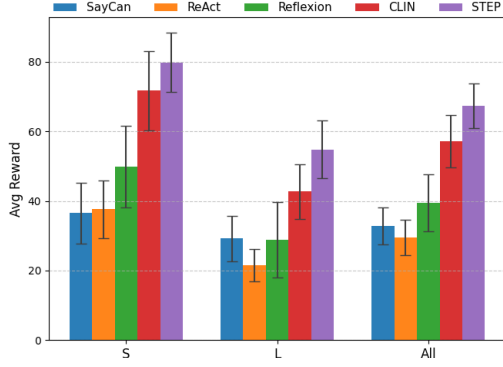


Figure 6: Comparison of STEP with SOTA.

not meet the specified constraints, the LLM refines it iteratively.

A.2.2 State-of-the-art Language Agents

With the rise of LLMs, there has been a growing body of work around embodied language agents.

SayCan (Ahn et al., 2022), developed by Robotics at Google, integrates LLMs with physical robots to execute real-world tasks based on natural language instructions. Viewing a physical robot as an embodied agent, SayCan itself functions as a language agent, bridging the gap between natural language processing and real-world interaction. While CALM uses DRRN to rerank action candidates, SayCan employs the temporal-difference (TD) RL method to train its value function. This allows SayCan to evaluate the feasibility of suggested actions by accounting for both the current state of the embodied agent and the environment, leading to a more grounded decision-making process. By combining LLMs’ semantic knowledge with a contextual understanding of real-world constraints, SayCan efficiently adapts to dynamic environments.

ReAct (Yao et al., 2023c) is a simple yet robust technique that integrates reasoning and action generation within language models. Unlike conventional approaches that separate reasoning and action phases, ReAct interleaves these processes, enabling language models to reason before executing actions and adapt dynamically to evolving circumstances. Through the use of reasoning traces, ReAct formulates plans, manages exceptions, and employs actions to acquire additional information from external sources. This integrated approach helps maintain task progress while reducing error propagation and hallucinations. ReAct can apply to a wide range of tasks, including question-answering, fact verification, and other interactive

decision-making scenarios.

Reflexion (Shinn et al., 2023) is a framework designed to enhance language agents by enabling self-reflection on past actions and outcomes. Rather than relying on traditional reinforcement learning methods that update model parameters, Reflexion enhances agents through verbal feedback, using linguistic reflection to guide future decisions. While prior models retain feedback only for a single reasoning step (Madaan et al., 2023; Gou et al., 2024a; Yao et al., 2023c), Reflexion stores feedback in an episodic memory buffer, allowing agents to utilize past experiences for more effective decision-making in subsequent tasks. This episodic memory facilitates iterative learning, helping agents avoid repeating errors and continuously refine their strategies. Through iterative self-reflection, Reflexion allows agents to adapt rapidly without the need for extensive retraining, making it a flexible approach for improving performance in complex real-world applications.

CLIN (Continual Learning from Interactions) (Majumder et al., 2023) is a language-based agent framework designed to enhance agents’ adaptability by enabling continual learning through dynamic memory updates. Leveraging from Reflexion, CLIN continually refines its memory with *causal abstraction insights* derived from interactions with the environment. This persistent episodic memory enables the agent to reflect on previous trials and incorporate those insights into future decision-making, allowing for rapid adaptation to new environments and tasks without requiring parameter updates. Moreover, with the use of meta-memory, CLIN generalizes its knowledge across multiple tasks and environments, allowing the agent to transfer insights to solve entirely new tasks in different settings. In this paper, we refer to the **CLIN ADAPT** version, which focuses on rapid adaptation in the same task, same environment.

B ScienceWorld Tasks Action Space

The ScienceWorld action space is reported in Table 3.

C Example of STEP trajectory for a task

We present a sample of task trajectory for STEP in Figure 7.

Action	Description
open/close OBJ	open/close a container
de/activate OBJ	activate/deactivate a device
connect OBJ to OBJ	connect electrical components
disconnect OBJ	disconnect electrical components
use OBJ [on OBJ]	use a device/item
look around	describe the current room
look at OBJ	describe an object in detail
look in OBJ	describe a container's contents
read OBJ	read a note or book
move OBJ to OBJ	move an object to a container
pick up OBJ	move an object to the inventory
put down OBJ	drop an inventory item
pour OBJ into OBJ	pour a liquid into a container
dunk OBJ into OBJ	dunk a container into a liquid
mix OBJ	chemically mix a container
go to LOC	move to a new location
teleport to LOC *	teleport to a specific room
eat OBJ	eat a food
flush OBJ	flush a toilet
focus on OBJ	signal intent on a task object
wait [DURATION]	take no action for some duration
task	describe current task
inventory	list agent's inventory

Table 3: The 25 actions in the action space of SCIENCE-WORLD. Actions can take up to two parameters, referencing objects the action should interact with. * signifies that the *teleport* action is only available to agents in a simplified mode.

Task Description: Your task is to measure the melting point of lead, which is located around the kitchen. First, focus on the thermometer. Next, focus on the lead. If the melting point of lead is above 50.0 degrees celsius, focus on the red box. If the melting point of lead is below 50.0 degrees celsius, focus on the green box. The boxes are located around the kitchen.

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

> go hallway
You move to the hallway.

> go greenhouse
You move to the greenhouse.

> look around
This room is called the greenhouse. In it, you see:
    the agent
    a substance called air
    a bee hive. The bee hive door is closed.
    a jug (containing nothing)
    a shovel
    a sink, which is turned off. In the sink is: nothing.
You also see:
    A door to the hallway (that is open)
    A door to the outside (that is open)

> go outside
You move to the outside.

> go kitchen
You move to the kitchen.

> focus on thermometer
You focus on the thermometer.

> pick up cup containing lead
You move the plastic cup to the inventory.

> focus on lead
You focus on the lead.

> focus on thermometer
You focus on the thermometer.

> look at thermometer
a thermometer, currently reading a temperature of 10 degrees celsius

> look around
This room is called the kitchen. In it, you see:
    the agent
    a substance called air
    a chair. On the chair is: nothing.
    a table. On the table is: a glass cup (containing nothing).
    a thermometer, currently reading a temperature of 10 degrees celsius
You also see:
    A door to the bathroom (that is open)
    A door to the hallway (that is open)
    A door to the outside (that is open)

> focus on lead
You focus on the lead.

> focus on red box
You focus on the red box. (Task Completed!)
```

Figure 7: Example of task Temp₂ trajectory with STEP.