DRIP: Decompositional Reasoning for agent Interpretable Planning

Anonymous ACL submission

Abstract

Research on LLM agents has shown remarkable progress, particularly in planning methods that leverage the reasoning capabilities of LLMs. However, challenges such as robustness and efficiency remain in LLM-based planning, with robustness, in particular, posing a significant barrier to real-world applications. In this study, we propose a framework that incorporates human reasoning abilities into planning. Specifically, this framework mimics the human ability to break down complex problems into simpler problems, enabling the decomposition of complex tasks into preconditions and subsequently deriving subtasks. The results of our evaluation experiments demonstrated that this human-like capability can be effectively applied to planning. Furthermore, the proposed framework exhibited superior robustness, offering new perspectives for LLM-based planning methods.

1 Introduction

The evolution of Large Language Models (LLMs) has been remarkable, revolutionizing the field of natural language processing (NLP) and extending their influence to interdisciplinary domains. Among these advancements, the emergence of LLM-powered agent technology (LLM Agents) has garnered significant attention due to its potential for real-world applications. These agents leverage the linguistic and reasoning capabilities of LLMs not only for conversational tasks but also for complex planning and decision-making processes (Liu et al., 2023; Singh et al., 2023; Wang et al., 2023b).

Planning, in the context of LLM agents, refers to the process of devising a sequence of actions required to achieve a specific goal. This process inherently relies on the reasoning and decision-making capabilities of LLMs, which are rooted in their ability to understand, generate, and manipulate natural language. For instance, achieving the goal of brushing one's teeth involves a series of steps such as heading to the sink, locating toothpaste, picking up the toothbrush, etc. If a subtask, such as locating toothpaste, fails, the agent must adapt by either setting a new goal (e.g., purchasing toothpaste) or skipping ahead to the next actionable step.

While LLMs have demonstrated success in planning tasks, challenges remain, particularly in scenarios involving long-horizon goals or complex sequences of actions. As the number of required actions increases, the accuracy of LLM-based planning tends to decline significantly (Valmeekam et al., 2024b). This is because long-horizon tasks expand the search space, and approximate retrieval-based reasoning—typical of current LLMs—struggles to maintain coherence and robustness over extended sequences. This issue highlights the need for a framework that enhances the robustness of LLMs in solving long-horizon tasks within planning scenarios, while also improving their efficiency in utilizing current conditions to create effective plans.

To tackle this challenge, we draw inspiration from human cognition, particularly the ability to break down complex problems into simpler, manageable subproblems. Cognitive psychology, such as that by Simon and Newell (1971); Chipman et al. (2000) suggests that humans naturally decompose difficult tasks into smaller, sequential steps, facilitating reasoning and execution . By mimicking this strategy, LLMs can construct hierarchical plans, enabling more robust and efficient solutions to complex goals.

In this study, we introduce a planning framework that leverages human-inspired decomposition to enhance LLMs'planning capabilities. While most prior methods rely on forward reasoning, our approach is based on backward reasoning, which decomposes goals into subgoals in a top-down manner. As shown in Figure 1, the framework incorpo-

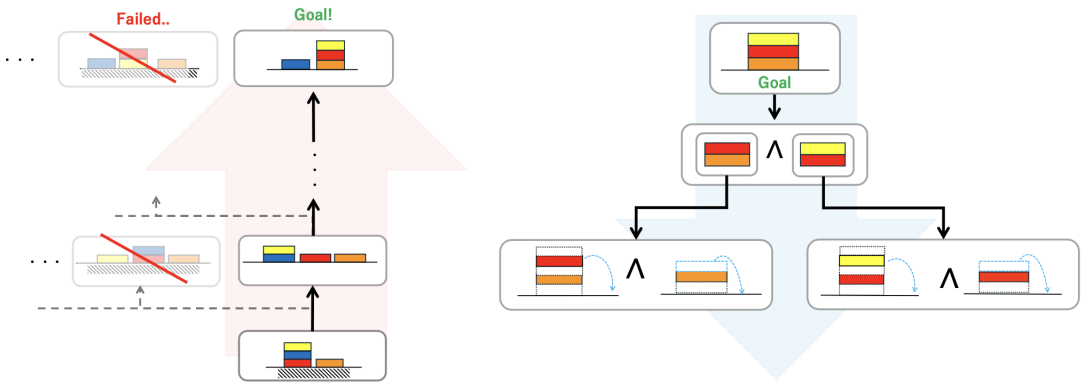

Figure 1: Overview Diagram of the DRIP Concept (Right). The left side illustrates the structure of existing methods using forward reasoning, while the right side represents the proposed method utilizing backward reasoning.

rates Backward Reasoning, a strategy well-suited to the hierarchical nature of goal decomposition, to achieve both efficiency and robustness in planning.

The contributions of this paper are as follows:

- We propose a planning framework that mimics human-like hierarchical goal decomposition, leveraging LLMs'natural language reasoning for task breakdown.
- Improved robustness in classical planning tasks: Through experiments on classical planning tasks, we demonstrate that the proposed framework enhances the robustness of LLM agents compared to existing methods.
- Efficiency through Backward Reasoning: We show that the framework reduces the number of actions required to achieve goals, highlighting its efficiency compared to Forward Reasoning approaches.
- Applicability to stochastic environments: The framework demonstrates effectiveness in dynamic and partially observable environments, such as household tasks, where goals are underspecified and actions may have uncertain outcomes.

By integrating insights from human cognition and leveraging the linguistic strengths of LLMs, this study aims to advance in LLM-based planning, paving the way for more reliable and versatile agent technologies.

2 Related Work

2.1 LLM Reasoning with decompose

The ability to simplify complex tasks by breaking them down into smaller, manageable subtasks is a hallmark of human cognition (Chipman et al., 2000). This concept, deeply rooted in cognitive psychology and logic (Simon and Newell, 1971), has inspired recent advancements in multi-step reasoning using LLMs (Xue et al., 2024; Junbing et al., 2023; Zhou et al., 2023). These studies commonly employ decomposition strategies, where a complex question is divided into simpler sub-questions, solved iteratively, and integrated to achieve the final solution. This approach often aligns with backward reasoning, a process of reasoning from the goal state to the initial state.

Empirical results from these studies have demonstrated significant improvements in the accuracy of solving challenging reasoning tasks. For instance, Xue et al. (2024) reported not only enhanced accuracy but also increased efficiency in reasoning tasks through decomposition. These findings suggest that decomposition-based reasoning is a promising approach for addressing the limitations of LLMs in handling complex problems. Building on this foundation, our study extends the application of backward reasoning from question-answering tasks to planning tasks.

2.2 Regression Planning

Backward reasoning, or regression planning, has long been studied in classical AI planning literature. It has played a central role in traditional planning algorithms, dating back to early works such as Waldinger (1977). Regression planning involves reasoning backward from the goal state to identify the sequence of actions required to achieve it. However, traditional regression planning methods often rely on symbolic planners, which necessitate predefined causal relationships between actions (Xu et al., 2019; Silver et al., 2022). This reliance on symbolic representations poses signifi-

cant challenges for real-world applications, where the dynamics of the environment are often too complex or uncertain to be fully captured by static, predefined rules.

In contrast, LLMs offer a unique advantage in their ability to dynamically generate and adapt rules based on their extensive pre-trained knowledge. This generative capability enables LLMs to overcome the rigidity of symbolic approaches, making them more suitable for real-time applications. Our study leverages this strength of LLMs to implement a regression planning framework that dynamically decomposes goals into sub-goals, addressing the limitations of traditional symbolic methods.

2.3 Planning for LLM Agents

Planning methods for LLM agents have been extensively studied, with various approaches proposed to enhance their reasoning and decision-making capabilities. According to the taxonomy by Huang et al. (2024), our study falls under the category of task decomposition, a strategy that has been widely adopted in LLM-based planning.

One prominent approach is Chain-of-Thought (CoT) prompting (Wei et al., 2023; Kojima et al., 2023), which encourages LLMs to explicitly consider intermediate reasoning steps. This method effectively breaks down problems into subtasks, facilitating step-by-step reasoning. The Plan-and-Solve framework (Wang et al., 2023a) further refines this approach by decomposing tasks into sequential subtasks, reducing reasoning leaps. While effective for static problems, its applicability to dynamic environments remains limited. Another notable approach is ReAct (Yao et al., 2023), which alternates between reasoning and planning, enabling decision-making in dynamic environments. This method significantly enhances planning capabilities but still relies on forward reasoning.

Forward reasoning, while widely used, faces inherent challenges in handling complex tasks due to the exponential growth of the search space (Yu et al., 2023). Even advanced reasoning models designed to enhance forward reasoning have struggled to achieve robust performance in long-horizon planning tasks (Valmeekam et al., 2024b). These limitations highlight the need for alternative approaches that can efficiently navigate the complexities of planning.

Backward reasoning has recently been explored in the context of LLM agent planning. For example, Ren et al. (2024) proposed a method that redefines the goal state as the initial state and "flips" the initial state to the goal state, simulating backward reasoning using LLMs. While promising, this approach encounters limitations in scenarios with multiple goal states or ambiguous goal representations. For instance, in environments like Blockworld, a goal such as "*The red block is on top of the blue block*" may allow for multiple valid configurations, leading to inconsistencies in the generated plans.

To address these challenges, our study proposes a stricter adherence to backward reasoning by explicitly decomposing the goal into intermediate subgoals. This approach ensures that each sub-goal is well-defined and contributes directly to achieving the final objective. By leveraging the extensive knowledge embedded in LLMs, our framework can handle ambiguous or underspecified goal representations, enhancing its applicability to diverse and dynamic problem-solving contexts.

3 Planning Framework:DRIP

Building upon cognitive psychology and logical reasoning, this study introduces DRIP — a framework that integrates hierarchical decomposition with dynamic planning for LLM agents. Inspired by the theory that humans solve problems by breaking them into subgoals (Simon and Newell, 1971; Chipman et al., 2000), DRIP operationalizes this mechanism through structured backward reasoning. This decomposition process aligns closely with the principles of backward reasoning, enabling the systematic breakdown of high-level goals into actionable subtasks. A high-level overview of the algorithm is presented in Algorithm 1, followed by detailed descriptions of each phase in the subsequent subsections.

3.1 Decompose

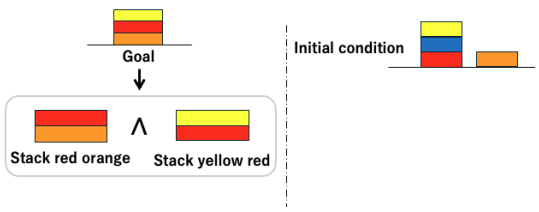

Figure 2: Decompose the goal into actions

In DRIP, "decomposition" refers to the process where the LLM recursively breaks down a goal into simpler subtasks by identifying the necessary preconditions. This forms a reasoning tree $T =$

Algorithm 1 DRIP Planning Algorithm

Require: Initial condition S_0, Goal G
Ensure: Plan π : a sequence of executable actions

```
Initialize reasoning tree T = (N, E), with root node n_0 = G
Set current condition S ← S_0, plan π ← [ ]
Set current frontier F ← {n_0}
for i = 1 to MAX_ITER do
    F_next ← ∅
    for all node n ∈ F do
        if exec(n, S) = true then
            π ← π ∪ {n}
            S ← apply(n, S)
        else
            {g_1, g_2, ..., g_k} ← dec(n)
            for all g_i ∈ dec(n) do
                Add g_i as child of n in tree T
                F_next ← F_next ∪ {g_i}
            end for
        end if
    end for
    if exec(G, S) = true then
        break
    end if
    F ← F_next
end for
return π
```

(N, E), where each node $n \in N$ stores. Let the goal be G and the condition be S, and the others are defined as follows.:

- π(Plan): A list of actions leading from S to G.
- $\mathcal{F}_{\text{next}}$: The set of nodes currently being processed. $\mathcal{F}_{\text{next}} \in N$
- g_n: Actions (subgoals) required to achieve the parent node's goal. $g_n \in G$

At each step, the LLM is prompted to generate subtasks for a given parent node:

$$\{g_1, g_2, \ldots, g_k\} \leftarrow \text{dec}(n)$$

For example (Figure 2), consider the following initial condition from the BlockWorld dataset(Valmeekam et al., 2023a):

S_0: "*The yellow block and orange block are clear, the hand is empty. The orange block is on the table, the blue block is on top of the red block, and the yellow block is on top of the blue block.*"(Initial condition in Figure 2 (right))

$G(g_0)$: "*The red block is on top of the orange block and the yellow block is on top of the red block.*"(Goal in Figure 2 (left))

As shown in the first-level box of Figure 2, this G can be decomposed into the actions "Stack red orange" and "Stack yellow red".

3.2 Executability

The executability step evaluates whether each subtask can be performed given the current condition. This evaluation is handled by an actuator, which assesses action feasibility. We define the function:

$$\text{exec}(n, S) \in \{True, False\}$$

The apply function takes a node n and a condition S as input and returns a new condition S'. In other words, it represents the execution of a valid action by the actuator.

$$S' \leftarrow \text{apply}(n, S)$$

Consider the example from Figure 3: initially, "Stack red orange" is executable, but "Stack yellow red" is not, as the red block is not clear. Therefore, as shown in Figure 3, the executability of the actions in the initial condition is labeled as EXECUTABLE and UNEXECUTABLE, respectively.

Upon executing the former, the condition updates, triggering a reevaluation of pending actions. This process is repeated until no executability changes remain.

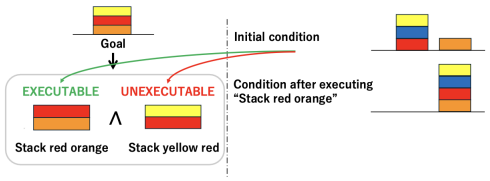

Figure 3: Execution of actions and changes in conditions

3.3 Re-decomposition and Termination of tree construction

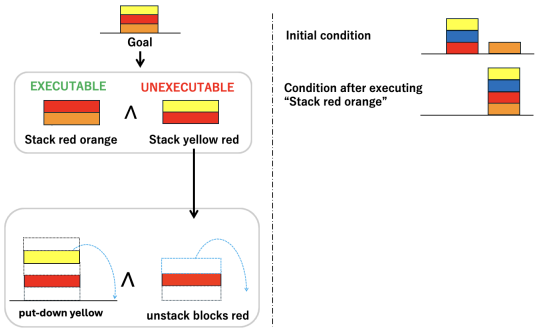

Figure 4: Execution of actions and changes in conditions

When executability updates stall, any remaining unexecutable actions are reinterpreted as sub-goals and recursively decomposed. For example (Figure 4), to execute "Stack yellow red", the LLM infers prerequisite actions like "Put-down yellow" and "Unstack blocks red". A node's decomposition is complete when all its child actions become executable. Once this occurs, executability propagates upward—if all children of a parent node are executable, the parent becomes executable as well. As shown Figure 5, this process continues until the

Table 1: The benchmark and the used models.(Manual) refers to experiments where humans executed the actions proposed by the LLM.

Methods	Models	Accuracy
DRIP	gpt-4o(OpenAI et al., 2024)	16.4% (18/110)
	Claude 3.7 Sonnet[1]	**40.9%** (**45**/110)
DRIP(Manual)	gpt-4o	**82.7%** (**91**/110)
CoT(Kojima et al., 2023)	gpt-4o	13.6%(15/110)
	Claude 3.7 Sonnet	23.6% (26/110)
ReACT(Yao et al., 2023)	gpt-4o	1.8%(2/110)
ReACT(Manual)	gpt-4o	31.8% (35/110)

root node is executable, indicating that the original goal can now be achieved.

The entire planning process follows a breadth-first search pattern, alternating decomposition and executability updates at each level.

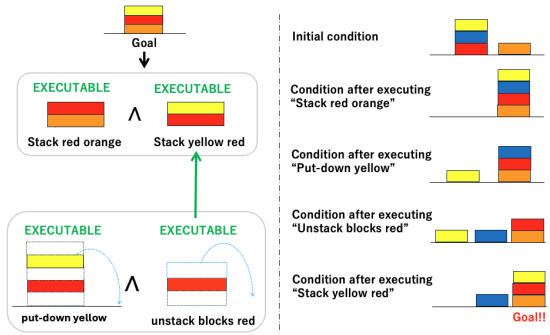

Figure 5: Execution of actions and changes in conditions

4 Experiment

4.1 BlockWorld

The BlockWorld task involves stacking blocks to achieve a specified goal state, making it a widely studied problem in classical planning. For this study, we utilized the BlockWorld_hard dataset (Valmeekam et al., 2023b, 2024a), which includes scenarios with stacking tasks involving between 6 and 15 blocks. This dataset is particularly challenging due to the increased complexity of the goal states and the number of actions required to achieve them. Detailed statistics regarding the number of blocks and configurations in the dataset are provided in Appendix A.1.

4.1.1 Experiment setup

In the original BlockWorld setting (Valmeekam et al., 2023b, 2024a), the available actions include Pick up, Unstack, Put down, and Stack. However, for this study, we simplified the action space to focus on three core actions: "Stack [blockA] [blockB]," "Put-down [block]," and "Unstack blocks [block]." The Pick up action was excluded as it is inherently performed as part of the other three actions. Additionally, while the original setting restricts the agent to holding only one block at a time, we relaxed this constraint to allow multiple blocks to be held simultaneously. This modification was made to better utilize the current condition for planning purposes. The specific experimental settings, including the prompts used for the LLM, are fully described in Appendix A.3. All experiments were conducted in Japanese.

4.1.2 Benchmark

We evaluated DRIP against baseline methods summarized in Table 1. DRIP uses LLMs for both decomposition and executability evaluation. In contrast, DRIP (Manual) uses a human to perform the actions. The LLM is only responsible for decompose, while the human checks if the actions are possible and then carries them out. The planning ends when the main goal (root action) is confirmed to be executable.

For comparison, we include CoT (Kojima et al., 2023) using GPT-4o and Claude 3.7 Sonnet[2], and ReACT (Yao et al., 2023), which alternates between reasoning and acting. In ReACT (LLM), actions are generated based on the initial condition and goal. In ReACT (Manual), humans execute the actions and provide the updated condition to the LLM, enabling iterative planning. Unlike other methods, ReACT (Manual) does not immediately fail on invalid actions. A run is considered failed only if five consecutive unexecutable actions are proposed or if the plan exceeds 40 steps. For all automated settings, humans evaluate whether the final plan achieves the goal.

[2]https://www.anthropic.com/claude/Sonnet

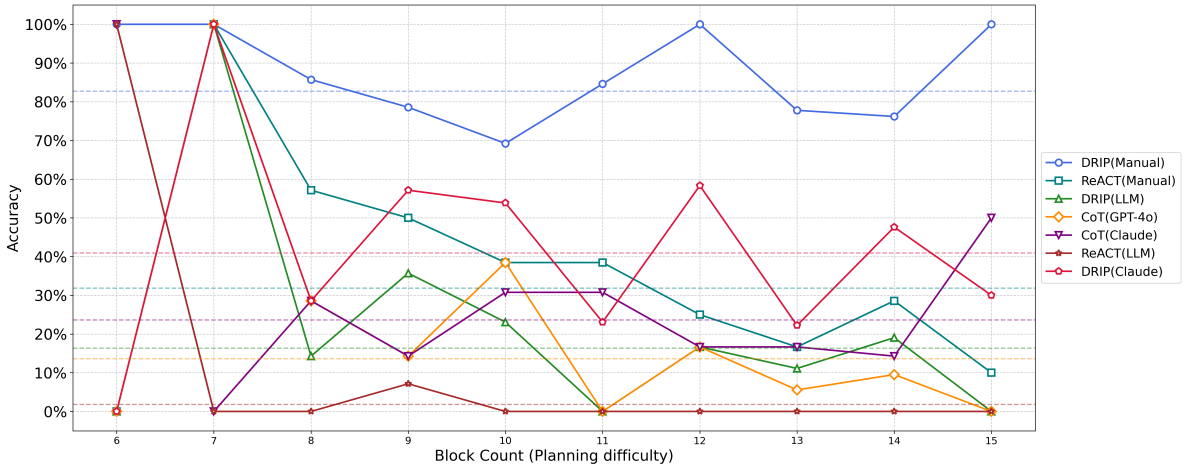

Figure 6: Experimental results.The horizontal axis represents the number of blocks, while the vertical axis indicates the accuracy for each block count. The blue is DRIP (Manual), the red is DRIP (Claude), the green is DRIP (LLM), the yellow is CoT (GPT-4), the purple is CoT (Claude) the cyan is ReACT (Manual), and the brown is ReACT (LLM). The dotted lines indicate the overall accuracy for each method.

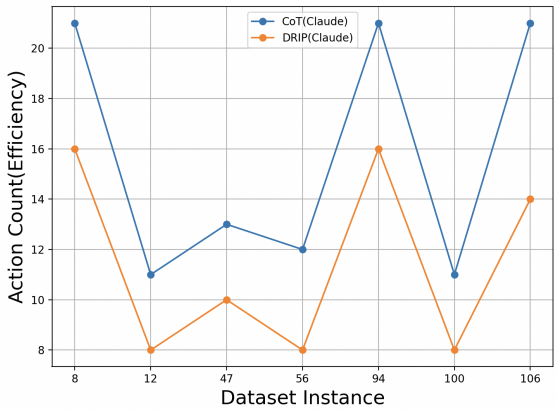

Figure 7: The difference in the number of actions included in the planning.The horizontal axis represents the instance numbers correctly solved by both methods., while the vertical axis represents the number of actions included.A lower action count indicates more efficient planning.

4.1.3 Results

The experimental results are summarized in Figure 6. As shown in Figure 6, DRIP (Manual) achieved the highest accuracy across all benchmarks, maintaining stable performance even as the number of blocks increased. This demonstrates its robustness and scalability in complex planning. Among LLM-only methods, DRIP (Claude) achieved the best performance. This result indicates that DRIP is capable of demonstrating sufficient ability in autonomous planning. On the other hand, the main reason why DRIP (Claude) could not match the performance of DRIP (Manual) lies in the difficulty of accurately describing block conditions using natural language such as “block X is clear” or “block Y is on block Z.”. The decline highlights a key limitation: as task complexity grows, condition descriptions become verbose and ambiguous, leading LLMs to misjudge action feasibility. This suggests that performance issues stem from condition representation, not from the planning framework itself. Future solutions may include multimodal models or formal representations like PDDL.

To assess planning efficiency, we analyzed the number of actions used by DRIP (Claude) and CoT (Claude) in successful cases (Figure 7). DRIP (Claude) consistently used fewer actions by avoiding unnecessary steps—unlike CoT, which decomposed all blocks regardless of context. On average, the difference in the number of actions between DRIP (Claude) and CoT (Claude) was 4.29 steps. This indicates DRIP'superior planning efficiency.

4.1.4 error analysis

Table 2: Details of DRIP (Manual) 's error type.

Error type	Number of datasets
Errors in decomposition by LLMs	14
Errors in the Framework	5
total	19

We analyzed the 19 cases (17.3% of the total) where DRIP (Manual) failed, summarized in Table 2. Of these, 14 errors stemmed from incorrect decomposition by the LLM. Despite the structured nature of BlockWorld and clear prompts, the model occasionally generated invalid action sequences, especially in configurations with ambiguous or complex block relationships.

The remaining 5 errors were found to be caused

```
Move a mug with a knife in it to a small table.
|--- Explore the [kitchen]
|   |--- Open the [kitchen door]
|       |--- Explore the [kitchen area]
|       |--- Explore the [doorknob]
|       |--- Hold the [doorknob]
|--- Hold the [mug]
|   |--- Find the [mug]
|       |--- Explore the [room]
|       |--- Explore the [shelf]
|--- Explore the [knife]
|   |--- Explore the [drawer]
|--- Hold the [knife]
|   |--- Pick up the [knife]
|--- Place the [knife] in the [mug]
|--- Explore the [small table]
|--- Place the [mug] on the [small table]
    |--- Hold the [mug]
    |--- Place the [mug] on the [small table]
```

Figure 8: The DRIP results for the task "*Move a mug with a knife in it to a small table*" in ALFRED environment.

by fundamental limitations of the current framework. These errors highlight scenarios where the framework's reliance on goal-based reasoning alone is insufficient. For example, consider a goal is: "*block a is on top of block j, block b is on top of block d, block c is on top of block b, block d is on top of block a, block f is on top of block i, block g is on top of block f, block i is on top of block c, block j is on top of block h*" (i.e., 'g-f-i-c-b-d-a-j-h'). Suppose the current condition is: 'c-b-d-a-j-h-e-g-f-i'. In this case, the remaining action to achieve the goal is "stack i c". Decomposing this action requires clearing block 'c' and moving it to create a separate tower with 'i' and 'c'. However, creating such a separate tower is not feasible because the goal condition ('c-d-b-a-j-h') has already been partially achieved. Moving block 'c' would violate the goal condition, making it impossible to proceed without undoing previously achieved subgoals. This example illustrates a key limitation of the current framework: it considers actions solely based on the goal state and does not account for the constraints imposed by the current condition. In certain scenarios, achieving the goal requires reasoning that integrates both the goal state and the current condition, as well as the ability to dynamically adjust the plan to avoid conflicts between intermediate subgoals.

4.2 ALFRED

In the previous section, we evaluated DRIP in BlockWorld — a highly structured environment where rule-based planning is often sufficient. However, real-world tasks are far more dynamic and underspecified, making them unsuitable for rigid symbolic approaches. To explore DRIP's applicability in such settings, we conduct a qualitative analysis using ALFRED(Shridhar et al., 2019), a household simulation benchmark for language-guided action generation.

```
1. Explore the [kitchen]
   - Find the mug and the knife

2. Hold the [mug]
  - Confirm that hands are free and hold the mug
   - Confirm that the knife is inside the mug

3. Explore the [living room]
   - Find the small table

4. Place the [mug]
   - Place the mug on the small table
```

Figure 9: The CoT (GPT-4o) results for the task "*Move a mug with a knife in it to a small table*"in ALFRED environment.

4.2.1 Challenges in Language Instructions and Action Generation

ALFRED(Shridhar et al., 2019) provides a visually and physically realistic household simulation environment designed for models that translate language into sequences of actions and interactions. It highlights many challenges faced when translating human language into robot actions to accomplish household tasks.

In robotics tasks, it is essential to develop methods that can cover the vast rules of the real world, which cannot be fully described using rule-based approaches. For example, consider the language directive "*clean up the dining table*" after eating a meal. The actions involved in this "*clean up*" directive differ from those in "*clean up the room.*" The former may include actions such as "carry dirty dishes to the sink" and "wipe the table," while the latter may involve "make the bed" and "vacuum the floor." This variability demonstrates the impracticality of predefining all possible language directives and their corresponding actions. Instead, it is necessary to dynamically devise actions based on the specific context and translate language instructions into actionable elements that robots can execute. This challenge underscores the importance of frameworks like DRIP, which can adapt to diverse scenarios by leveraging the reasoning capabilities of LLMs.

4.2.2 Qualitative Evaluation Using ALFRED

To evaluate DRIP's ability to interpret and operationalize real-world instructions, we compared its

Table 3: Comparison of DRIP and CoT action plans for "*Move a mug with a knife in it to a small table.*"

Aspect	DRIP	CoT
Granularity	Fine-grained steps including environment exploration and tool use	High-level, abstract steps only
Environmental Interaction	Explicit (e.g., shelves, doorknob, drawer)	Implicit or omitted
Causal Structure	Maintains logical order (e.g., place knife before moving mug)	Partially unordered; assumes end-state is satisfied
State Transitions	Models each state change (e.g., knife pickup, mug update)	Assumes preconditions (knife already in mug)
Physical Realism	Suitable for robot execution	Risk of infeasibility in real-world settings

output with that of CoT for the instruction: "*Move a mug with a knife in it to a small table.*" DRIP in Figure 10 decomposed this instruction into detailed, physically grounded substeps, such as: exploring the environment (e.g., the kitchen and shelves), locating and holding the mug and the knife, placing the knife into the mug, and transporting the mug to the target location. This decomposition respects causal and spatial dependencies between actions and reflects realistic environmental interactions (e.g., opening drawers, holding doorknobs). In contrast, CoT in Figure 10 produced a high-level plan that omitted critical steps. It assumed the knife was already in the mug and skipped over retrieval, insertion, and exploration steps. This resulted in a less complete and potentially unexecutable plan in real-world settings.The comparison highlights DRIP's ability to generate execution-ready plans that align more closely with embodied reasoning and physical manipulation. A summary of the differences is shown in Table 3. Additional results in Appendix B.1 confirm similar trends.

Limitations

The proposed DRIP framework demonstrates robustness and efficiency in planning by mimicking human capabilities. However, it has several limitations. First, there are challenges related to the decomposition capabilities of LLMs. While LLMs possess vast amounts of knowledge, the extent to which they can perform commonsense reasoning remains largely unexplored. For instance, executing an action like "*move A to the position of B*" requires the preconditionthat "*A is located somewhere other than B.*" In this study, we explicitly specified feasible actions and utilized structured tasks in the experiments. However, in real-world applications, this limitation could have a significant impact.

Second, the number of LLM calls required is an issue. While CoT requires a single call, DRIP (Manual) uses hierarchical reasoning, averaging 5.98 calls, and DRIP (Claude) averages 6.18 calls. On the other hand, the average number of LLM calls for ReACT (Manual) is 28.3, whereas DRIP achieves a significant reduction in comparison. Humans are said to switch between different types of reasoning, as exemplified by the "Fast and Slow" theory(Kahneman, 2011). Building on these insights, further exploration is needed to develop methods that appropriately combine backward reasoning and forward reasoning.

Conclusion

This paper proposed a planning framework for LLM agents inspired by human problem-solving, particularly the ability to decompose complex problems into simpler components. By employing a backward reasoning approach, the framework dynamically decomposes tasks into preconditions and subtasks, enhancing planning robustness and aligning with human cognitive processes.

Experimental results show that the framework outperforms forward reasoning-based methods in robustness and efficiency, particularly in long-horizon tasks. It achieves goals with fewer steps by leveraging the current state to avoid unnecessary actions, demonstrating its potential for real-world applications.

Looking ahead, we plan to extend this framework to real-world applications by integrating multimodal inputs and actuators, such as robotics systems. By bridging the gap between natural language understanding and action generation, we envision this framework contributing to the development of more intelligent, adaptable, and context-aware agents.

References

Susan F Chipman, Jan Maarten Schraagen, and Valerie L Shalin. 2000. Introduction to cognitive task analysis. In *Cognitive task analysis*, pages 3–23. Psychology Press.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *CoRR*, abs/2402.02716.

Yan Junbing, Chengyu Wang, Taolin Zhang, Xiaofeng He, Jun Huang, and Wei Zhang. 2023. From complex to simple: Unraveling the cognitive tree for reasoning with small language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12413–12425, Singapore. Association for Computational Linguistics.

Daniel Kahneman. 2011. *Thinking, fast and slow*. macmillan.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners. *Preprint*, arXiv:2205.11916.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+p: Empowering large language models with optimal planning proficiency. *Preprint*, arXiv:2304.11477.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark ..., and Yury Malkov. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Allen Z. Ren, Brian Ichter, and Anirudha Majumdar. 2024. Thinking forward and backward: Effective backward planning with large language models. *Preprint*, arXiv:2411.01790.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2019. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. *CoRR*, abs/1912.01734.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomas Lozano-Perez, Leslie Pack Kaelbling, and Joshua B. Tenenbaum. 2022. Predicate invention for bilevel planning. In *AAAI Conference on Artificial Intelligence*.

Herbert A Simon and Allen Newell. 1971. Human problem solving: The state of the theory in 1970. *American psychologist*, 26(2):145.

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

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023a. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Preprint*, arXiv:2206.10498.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023b. On the planning abilities of large language models - a critical investigation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. 2024a. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of lrm o1. *Preprint*, arXiv:2410.02162.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. 2024b. LLMs still can't plan; can LRMs? a preliminary evaluation of openAI's o1 on planbench. In *NeurIPS 2024 Workshop on Open-World Agents*.

Richard Waldinger. 1977. Achieving several goals simultaneously. *Machine Intelligence*, 8.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. 2019. Regression planning networks. In *Neural Information Processing Systems*.

Shangzi Xue, Zhenya Huang, Jiayu Liu, Xin lin, Yuting Ning, Binbin Jin, Xin Li, and Qi Liu. 2024. Decompose, analyze and rethink: Solving intricate problems with human-like reasoning cycle. In *Advances in Neural Information Processing Systems*, volume 37, pages 357–385. Curran Associates, Inc.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Fei Yu, Hongbo Zhang, and Benyou Wang. 2023. Natural language reasoning, a survey. *ACM Computing Surveys*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Experience(Blockworld)

A.1 Details of the Blockworld dataset

We summarize the number of blocks and the number of instances for each dataset in the Blockworld dataset in a table4.

Table 4: Details of the number of blocks in the dataset

Number of blocks	Number of datasets
6	1
7	1
8	7
9	14
10	13
11	13
12	12
13	18
14	21
15	10
total	110

A.2 Details of the Benchmark

DRIP(LLM) refers to a method where the actuator is also implemented using an LLM. In other words, this method involves having the LLM determine executability and reason about changes in the state after executing an action. DRIP (Manual) refers to a method where a human acts as the actuator to stack the blocks. In this method, humans determine executability and provide feedback on whether the actions proposed by the LLM were successfully executed or not. The LLM responsible for action decomposition only reasons about actions based on the goal and does not receive feedback on the condition. The termination condition is when the root node action (data set goal) is determined to be executable.

Claude 3.5 Sonnet[3], which belongs to the same Claude family as Claude 3.7 Sonnet, has achieved the best performance among LLMs on datasets involving stacking five or fewer blocks[4]. In ReACT (Manual), humans execute the actions proposed by the LLM and return the resulting new state as an observation after each action. In this approach, the goal and initial condition are provided at the beginning, and the LLM generates actions based on this information. After executing an action, humans provide the updated condition to the LLM, which then generates the next action based on the new condition. This cycle continues iteratively.

For methods without the "Manual" label, humans evaluate the plans output by the LLM as well.

A.3 Prompt

The prompts used in the Blockworld experiments are attached.

Table 5 is the prompt used for decomposition and is utilized in both DRIP (Manual) and DRIP (LLM). The Japanese versions used in the experiments are followed by their English translations.

Table 6,7 and table 9 are prompts used in DRIP (LLM) to utilize LLM as an actuator. Table 6,7. Table 6 is a prompt used to determine whether an action is executable, while table 9 is a prompt used to describe how the condition of the blocks changes after an action is performed. Table 7 is the English translation of table 6.

[3]https://www.anthropic.com/news/claude-3-5-Sonnet

[4]https://github.com/karthikv792/LLMs-Planning

私はブロックを積み上げてスタックに整理する必要があるブロックセットで遊んでいます。
私の究極の目標は、できるだけ失敗せず、効率的にゴール状態にすることです。

私の目標「task」。このゴールを達成するために必要な唯一の条件に基づいてアクションを示してください。

実行可能なアクション：
1.stack [block1] [block2]block1>block1
を[block2]の上に積む。
2.unstack blocks [block2]
blocksを[block2]の上からテーブルへ外して[block2]をクリアにする。
3.put-down [block2]block2>block2
をテーブルの上に置く。

制約と注意事項:
-「stack [block1] [block2]」という形の文字列がゴール状態の場合のみ、以下を返します:
unstack blocks [block2]
put-down [block2]
- それ以外の条件では:
stack [block1] [block2]のアクションを返します。
- 余計な情報の排除:
常にアクションのみをリスト形式で回答してください。他の余計な文章や構文（[ ]や括弧、番号など）は含めないでください。具体例として以下の形式を必ず守ってください:
action1
action2
action3
【回答例】

【目標】
stack blue yellow

【条件に基づくアクション】
unstack blocks yellow
put-down yellow
【目標】 task
【条件に基づくアクション】

(English)
I am playing with a block set where I need to stack and organize the blocks into a stack. My ultimate goal is to reach the goal state as efficiently as possible without making mistakes.

My Goal"task". Please provide actions based solely on the conditions required to achieve this goal.

Available Actions:
1.stack [block1] [block2]
Place [block1] on top of [block2].
2.unstack blocks [block2]
Remove blocks from the top of [block2] and clear [block2] by placing them on the table.
3.put-down [block2]
Place [block2] on the table.

Constraints and Notes:
- Only when the goal state is in the form of "stack [block1] [block2]", return the following actions:
unstack blocks [block2]
put-down [block2]
- For all other conditions:
Return the action stack [block1] [block2].
- Eliminate unnecessary information:
Always respond with actions in a list format. Do not include extra sentences, structures (e.g., [ ] or parentheses), or numbers. Always adhere to the following format:
action1
action2
action3

[Example Responses]

[Goal]
stack blue yellow

[Actions Based on Conditions]
unstack blocks yellow
put-down yellow
[Goal]
task

[Actions Based on Conditions]

Table 5: Decomposition Prompt for Blockworld.The Japanese prompts used in the experiments are followed by their English translations.

あなたはブロックを積み上げてスタックに整理する必要があるブロックセットで遊んでいます。与えられた状態で、指定された行動が実行可能かどうかを判断してください。

##ブロックワールドのアクションルール
unstack blocks [block2]
- blocksは[block2]の上のブロックを意味します。[block2]をクリアにするために、[block2]の上のブロックを[block2]の上から外し、テーブルに置く行為です。[block2]の上のブロックを動かした場合、上にあるブロック全ても一緒に動きます。
put-down [block2]
- [block1]と[block2]を別の塔にするために、[block2]をテーブルの上に置く行為です。 **[block2]の上にあるブロック全ても一緒に動きます。**
stack [block1] [block2]
- [block1]を[block2]の上に積む行為です。[block2]がクリアである必要があります。**[block1]の上にあるブロック全ても一緒に動きます。**

##ブロックワールドのルール
- ブロックは何個でも持つことができます。
- **ブロックは上に他のブロックがあっても、下のブロックからまとめて持つことができます。**
- あるブロックを動かす場合、**そのブロックの上のブロックすべてを同時に動かします**。
- 同じ塔にあるブロック同士をstackすることはできません。

判断手順
以下の手順で分析し、各ステップの結果を明示的に出力してください:
1. 現在の状態分析
- 各ブロックの位置関係を箇条書きで列挙
- 各ブロックの上に他のブロックがあるかを確認
- 各ブロックがテーブルの上にあるかを確認
2. 実行したい行動の分析
- 移動するブロック:どのブロックを動かすか
* 「移動するブロック」とは指定された基準ブロックを指します。ただし、このブロックの上に他のブロックがあれば、それらすべてをまとめて動かす必要があります。 - 移動先:ブロックまたはテーブル
- 移動先の状態:
ブロックの場合:[block2]上に他のブロックがあるか
テーブルの場合:常に置くことが可能 3. 実行不要かの判断
「実行不要」:以下に該当
- アクション:unstack blocks [block2]
すでに[block2]の上に何もブロックがないとき
-アクション:stack [block1] [block2]
すでに[block1]が[block2]に積まれているとき
-アクション:put-down [block2]
すでに[block1]と[block2]が別の塔であるとき
[block2]がすでにテーブルに置かれているかどうかは関係ありません。
4. 実行可能性の判断(実行不要であった場合は飛ばす)
「実行不可能」:以下のいずれかに該当
- 移動先がブロックの場合、そのブロック([block2])の上に他のブロックがある
- 移動先のブロック([block2])と移動するブロック([block1])が同じ塔にある
「実行可能」:
- 移動するブロックの上にブロックがあっても常に動かすことができます。
- 移動先([block2])がクリアであれば、移動先([block2])にブロックを積むことができます。
- テーブルにはいつでもブロックを積むことができます。
- 移動するブロックが移動先のブロックとは異なる塔にある場合、積む操作は可能です。
「END」:
- アクションが"END"の場合
現在の判断対象
状態:{initial_condition}
行動:{action_list}
1. 状態分析:ここに各ブロックの位置関係を箇条書きで記述
2. 行動分析: ここに移動元と移動先の状態を記述
3. 実行不要かの判断
4. 判断結果:判断,理由(判断と理由の間に,を入れる)

Table 6: Executability prompt for Blockworld(japanese)

You are playing with a block set where you need to stack and organize the blocks into a stack. Based on the given state, determine whether the specified action is executable.

##Blockworld Action Rules unstack blocks [block2] - blocks refers to the blocks above [block2]. This action involves removing the blocks above [block2] to clear [block2] and placing them on the table. If you move the blocks above [block2], all blocks stacked on top of them will also be moved together.
put-down [block2] - This action involves placing [block2] on the table to separate [block1] and [block2] into different towers. **All blocks stacked on top of [block2] will also be moved together.**
stack [block1] [block2]
- This action involves stacking [block1] on top of [block2]. [block2] must be clear for this action to be performed. **All blocks stacked on top of [block1] will also be moved together**.

##Blockworld Rules
- You can hold any number of blocks at once.
- Even if there are blocks above a block, you can pick up the block along with all the blocks stacked on top of it.
- When moving a block, all blocks stacked on top of it must be moved together.
- You cannot stack blocks that are already part of the same tower.M
##Decision Procedure
Analyze the situation using the following steps and explicitly output the results for each step:
1. Analyze the Current State
- List the positional relationships of all blocks in bullet points.
- Check if there are any blocks above each block.
- Check if each block is on the table.
2. Analyze the Desired Action
- Moving Block: Identify which block is being moved.
* The "moving block" refers to the specified base block. However, if there are other blocks stacked on top of this block, all of them must be moved together.
- Destination: Specify whether the destination is a block or the table.
- State of the Destination: If the destination is a block: Check if there are any blocks on top of [block2].
If the destination is the table: It is always possible to place blocks on the table.
3. Determine if the Action is Unnecessary
"Unnecessary Action": The action is unnecessary if any of the following conditions are met:
Action: unstack blocks [block2]
The action is unnecessary if there are no blocks above [block2].
Action: stack [block1] [block2]
The action is unnecessary if [block1] is already stacked on [block2].
Action: put-down [block2]
The action is unnecessary if [block1] and [block2] are already part of different towers.
Whether [block2] is already on the table is irrelevant.
4. Determine Executability (Skip this step if the action is unnecessary)
"Not Executable": The action is not executable if any of the following conditions are met:
If the destination is a block ([block2]), and there are other blocks on top of [block2].
If the moving block ([block1]) and the destination block ([block2]) are part of the same tower.
"Executable": You can always move a block, even if there are blocks stacked on top of it.
You can stack a block on the destination ([block2]) if [block2] is clear.
You can always place blocks on the table.
If the moving block is part of a different tower from the destination block, stacking is possible.
"END":If the action is END.
##Current Target for Evaluation
State: initial_condition
Action: action_list
1.State Analysis:[Describe the positional relationships of all blocks in bullet points here.]
2.Action Analysis:[Describe the state of the source and destination here.]
3.Determine if the Action is Unnecessary
4.Decision Result:[Decision],[Reason] (Separate the decision and reason with a comma)

Table 7: Executability prompt for Blockworld(Translation to English)

あなたはブロックを積み上げるブロックセットで遊んでいます。{condition}という状況から{action}という行動を取った場合、状況がどう変わるかを考え、文章で記載してください。

ブロックワールドのアクションルール
unstack [block1] [block2]
- [block1]は[block2]の上のブロックを意味します。[block2]をクリアにするために、[block1]を[block2]の上から外し、テーブルに置く行為です。[block1]を動かした場合、[block1]の上にあるブロック全ても一緒に動きます。[block2]はクリアになります。
put-down [block2]
- [block1]と[block2]を別の塔にするために、[block2]をテーブルの上に置く行為です。**[block2]の上にあるブロック全ても一緒に動きます。**[block2]はクリアになり、テーブルに置かれます。
stack [block1] [block2]
- [block1]を[block2]の上に積む行為です。[block2]がクリアである必要があります。**[block1]の上にあるブロック全ても一緒に動きます。**
ブロックワールドのルール
- ブロックは何個でも持つことができます。
- **ブロックは上に他のブロックがあっても、下のブロックからまとめて持つことができます。**
- あるブロックを動かす場合、**そのブロックの上のブロックすべてを同時に動かします**。

- 同じ塔にあるブロック同士をstackすることはできません。
- 移動先のブロック([block2])がクリアでない場合、stack [block1] [block2]は行うことができません。
-ブロックが空である/クリアである、ということはそのブロックの上に何も載っていないことを意味します。
最終的な回答として、与えられた状況と同じように、空のブロック、ブロックの状態、テーブルの上にあるブロックの順で記述してください。
記述の仕方は「〜〜は〜〜にある。」で統一してください。
答え方は以下の通りにしてください。
現在の判断対象
状態:{condition}
行動:{action}
考え方:
最終的な状態:[答え]

(English)
You are playing with a block set where you stack blocks. Consider how the situation will change if you take the action {action} from the situation {condition}, and describe it in writing.
Block World Action Rules unstack [block1] [block2]
- [block1] refers to the block above [block2]. This action removes [block1] from [block2] and places it on the table, clearing [block2]. If [block1] is moved, all blocks above [block1] move with it. [block2] is cleared.

put-down [block2]
- This action places [block2] on the table to separate [block1] and [block2] into different towers. **All blocks on top of [block2] move together.** [block2] is cleared and placed on the table.
stack [block1] [block2]
- This action places [block1] on top of [block2]. [block2] must be cleared. **All blocks above [block1] will move together.**
Block World Rules
- You can have as many blocks as you want.
- **Blocks can be picked up together from the bottom even if there are other blocks on top of them.**
- When moving a block, **all blocks above it will move together.**
- You cannot stack blocks that are in the same tower.
- If the destination block ([block2]) is not clear, you cannot stack [block1] [block2].
- An empty/clear block means that there is nothing on top of it.

As your final answer, describe the empty blocks, the state of the blocks, and the blocks on the table in the same order as the given situation. Use the format “X is in Y” consistently.
Answer in the following format.
Current target for judgment
State: {condition}
Action: {action}
Thought process:
Final state: [answer]

Table 8: The prompt for updating conditions for Blockworld. The Japanese prompts used in the experiments are followed by their English translations.

B Experience(ALFRED)

B.1 Results

The results for other tasks in ALFRED are provided here. For all tasks, the characteristics shown in Table 3 were observed.

```
Place a heated tomato slice in the sink.
|--- 1) Explore the kitchen
|   |--- - Explore the entrances
|   |   |--- - Explore entrance surroundings
|   |   |--- - Explore the doorknob
|   |--- - Explore the refrigerator
|   |   |--- - Locate the refrigerator
|   |   |--- - Open the refrigerator
|   |--- - Explore the cabinets
|       |--- Move to the front of the cabinets
|       |--- Open the cabinets
|       |--- Explore the cabinets interior
|--- 2) Pick up the tomato
|   |--- - Find the tomato
|   |   |--- 2) Explore the refrigerator
|   |   |--- 3) Explore the pantry
|   |--- - Pick up/hold the tomato
|--- 3) Cut the tomato
|   |--- Locate the knife
|   |   |---  Open the drawer
|   |   |---  Find the knife
|   |--- Hold the knife
|   |--- Place the cutting board
|   |   |--- - [Explore the kitchen]
|   |   |--- - [Explore the cutting board]
|   |   |--- - [Hold the cutting board]
|   |   |--- - [Explore the desired location]
|   |--- Locate the tomato
|   |--- Hold the tomato
|   |--- Place the tomato on the cutting board
|   |--- Cut the tomato with the knife
|--- 4) Explore the stove
|--- 5) Turn on the stove
|   |--- - Locate the stove
|   |--- - Hold the stove switch
|   |   |--- - Explore the stove switch
|   |--- - Turn on the stove switch
|--- 6) Hold the tomato slices
|--- 7) Place the tomato slices on the stove
|--- 10) Explore the sink
    |--- [Explore the room]
```

Figure 10: The DRIP results for the task "Place a heated tomato slice in the sink."

In the task "Place a cooled-off apple inside the garbage bin," using a cooled-off apple requires either taking an apple out of the refrigerator or placing an apple in the refrigerator and waiting for a while. However, it seems that neither approach demonstrates sufficient contextual understanding to achieve this.

```
1. Explore the [kitchen] - To find the tomato.
2. Hold the [tomato] - Pick up the tomato.
3. Explore the [counter] - To find the knife.
4. Hold the [knife] - To cut the tomato.
5. Cut the [tomato]
- Slice the tomato into pieces.
6. Place the [knife] - To free your hands.
7. Hold the [tomato slice]
- Take one of the sliced pieces.
8. Explore the [stove]
- To find a device for heating.
9. Turn on the [stove]
- Start the heating process.
10. Place the [tomato slice]
- Heat it on the stove.
11. Hold the [heated tomato slice]
- Take the heated tomato slice.
12. Explore the [sink]
- To find a place to put the tomato slice.
13. Place the [heated tomato slice]
- Put the heated tomato slice in the sink.
```

Figure 11: The CoT results for the task "Place a heated tomato slice in the sink."

```
Place a cooled-off apple inside the garbage bin.
|--- Explore the [kitchen]
|   |--- Explore the [kitchen entrance]
|       |--- Explore the [current location]
|--- Explore the [apple]
|   |--- 1) Explore the [kitchen]
|   |   |--- Explore the [kitchen entrance]
|   |   |--- Explore the [kitchen counter]
|   |   |--- Explore the [kitchen cabinet]
|   |--- 2) Explore the [dining table]
|   |--- 3) Explore the [fridge]
|       |--- Open the [fridge]
|       |--- Explore [fridge interior]
|--- Hold the [apple]
|   |--- Explore the [room]
|   |--- Find the [apple]
|   |--- Pick up/hold the [apple]
|--- Explore the [fridge]
|   |--- Explore the [fridge]
|       |--- Find the [fridge]
|       |--- Open the [fridge]
|       |--- Explore [fridge interior]
|--- Close the [fridge]
|   |--- Find the [fridge]
|   |   |--- Explore the [living room]
|   |--- Close the [fridge]
|       |--- Explore the [kitchen]
|--- Explore the [garbage bin]
|   |--- 1) Explore the [room]
|   |   |--- 1) Open the [door]
|   |   |--- 2) Explore the [room]
|   |--- 2) Find the [garbage bin]
|--- Drop the [apple]
|--- Close the [garbage bin]
```

Figure 12: The DRIP results for the task "Place a heated tomato slice in the sink."

```
1. [Explore the kitchen]
- To search for the apple.
2. [Hold the apple]
- Hold it in hand to prepare for the next action.
3. [Explore the garbage bin]
- To locate the garbage bin.
4. [Open the garbage bin]
- To place the apple inside.
5. [Drop the apple]
- Put the apple into the garbage bin.
6. [Close the garbage bin]
- To complete the task.
```

Figure 13: The CoT results for the task "Place a cooled-off apple inside the garbage bin. "

B.2 Prompt

あなたはブロックを積み上げるブロックセットで遊んでいます。{condition}という状況から{action}という行動を取った場合、状況がどう変わるかを考え、文章で記載してください。
ブロックワールドのアクションルール
unstack [block1] [block2]
- [block1]は[block2]の上のブロックを意味します。[block2]をクリアにするために、[block1]を[block2]の上から外し、テーブルに置く行為です。[block1]を動かした場合、[block1]の上にあるブロック全ても一緒に動きます。[block2]はクリアになります。
put-down [block2]
- [block1]と[block2]を別の塔にするために、[block2]をテーブルの上に置く行為です。 **[block2]の上にあるブロック全ても一緒に動きます。**[block2]はクリアになり、テーブルに置かれます。
stack [block1] [block2]
- [block1]を[block2]の上に積む行為です。[block2]がクリアである必要があります。**[block1]の上にあるブロック全ても一緒に動きます。**
ブロックワールドのルール
- ブロックは何個でも持つことができます。
- **ブロックは上に他のブロックがあっても、下のブロックからまとめて持つことができます。**
- あるブロックを動かす場合、**そのブロックの上のブロックすべてを同時に動かします**。
- 同じ塔にあるブロック同士をstackすることはできません。
- 移動先のブロック([block2])がクリアでない場合、stack [block1] [block2]は行うことができません。
-ブロックが空である/クリアである、ということはそのブロックの上に何も載っていないことを意味します。
最終的な回答として、与えられた状況と同じように、空のブロック、ブロックの状態、テーブルの上にあるブロックの順で記述してください。
記述の仕方は「〜〜は〜〜にある。」で統一してください。
答え方は以下の通りにしてください。
現在の判断対象
状態:{condition}
行動:{action}
考え方:
最終的な状態:[答え]

(English)
You are playing with a block set where you stack blocks. Consider how the situation will change if you take the action {action} from the situation {condition}, and describe it in writing.
Block World Action Rules unstack [block1] [block2]
- [block1] refers to the block above [block2]. This action removes [block1] from [block2] and places it on the table, clearing [block2]. If [block1] is moved, all blocks above [block1] move with it. [block2] is cleared.

put-down [block2]
- This action places [block2] on the table to separate [block1] and [block2] into different towers. **All blocks on top of [block2] move together.** [block2] is cleared and placed on the table.
stack [block1] [block2]
- This action places [block1] on top of [block2]. [block2] must be cleared. **All blocks above [block1] will move together.**
Block World Rules
- You can have as many blocks as you want.
- **Blocks can be picked up together from the bottom even if there are other blocks on top of them.**
- When moving a block, **all blocks above it will move together.**
- You cannot stack blocks that are in the same tower.
- If the destination block ([block2]) is not clear, you cannot stack [block1] [block2].
- An empty/clear block means that there is nothing on top of it.

As your final answer, describe the empty blocks, the state of the blocks, and the blocks on the table in the same order as the given situation. Use the format “X is in Y” consistently.
Answer in the following format.
Current target for judgment
State: {condition}
Action: {action}
Thought process:
Final state: [answer]

Table 9: The prompt for updating conditions for Blockworld. The Japanese prompts used in the experiments are followed by their English translations.