Reasoning with Language Model is Planning with World Model

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Large language models (LLMs) have shown remarkable reasoning capabilities, particularly with chain-of-thought (CoT) prompting. However, LLMs can still struggle with problems that are easy for humans, such as generating action plans for executing tasks in a given environment, or performing complex math or logical reasoning. The deficiency stems from the key fact that LLMs lack an internal *world model* to predict the world *state* (e.g., environment status, intermediate variable values) and simulate long-term outcomes of actions. This prevents LLMs from performing deliberate planning akin to human brains, which involves exploring alternative reasoning paths, anticipating future states and rewards, and iteratively refining existing reasoning steps. To overcome the limitations, we propose a new LLM reasoning framework, **Reasoning via Planning (RAP)**. RAP repurposes the LLM as both a world model and a reasoning agent, and incorporates a principled planning algorithm (based on Monte Carlo Tree Search) for strategic exploration in the vast reasoning space. During reasoning, the LLM (as agent) incrementally builds a reasoning tree under the guidance of the LLM (as world model) and rewards, and efficiently obtains a high-reward reasoning path with a proper balance between exploration *vs.* exploitation. We apply RAP to a variety of challenging reasoning problems including plan generation, math reasoning, and logical inference. Empirical results on these tasks demonstrate the superiority of RAP over various strong baselines, including CoT and least-to-most prompting with self-consistency. RAP on LLaMA-33B surpasses CoT on GPT-4 with 33% relative improvement in a plan generation setting.

1 Introduction

Large language models (LLMs) have exhibited emergent reasoning abilities in a wide range of tasks [5, 10, 44, 2]. Recent approaches further boost their ability by prompting LLMs to generate intermediate reasoning steps (e.g., chain-of-thought, CoT [59]) or answer a series of subquestions (e.g., least-to-most prompting [66]). However, LLMs still face difficulties with tasks that humans find easy. For excample, in creating action plans to move blocks to a target state, GPT-3 [5] achieves a success rate of only 1%, compared to 78% for humans [57]; these models also struggle when solving complex tasks that require multiple steps of math, logical, or commonsense reasoning [65, 22, 41, 6].

Humans possess an internal **world model**, a mental representation of the environment [28, 27, 15], which enables humans to simulate actions and their effects on the world's state for deliberate **planning** during complex tasks of motor control, imagery, inference, and decision making [54, 55, 4, 49, 17, 33]. For example, to make an action plan towards a goal, planning with the world model involves exploring various alternative courses of actions, assessing the likely outcomes by rolling out possible future scenarios, and iteratively refining the plan based on the assessment [25, 14, 52, 19, 48, 21]. This is in stark contrast to the current LLM reasoning, which instinctively generates a reasoning trace in

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

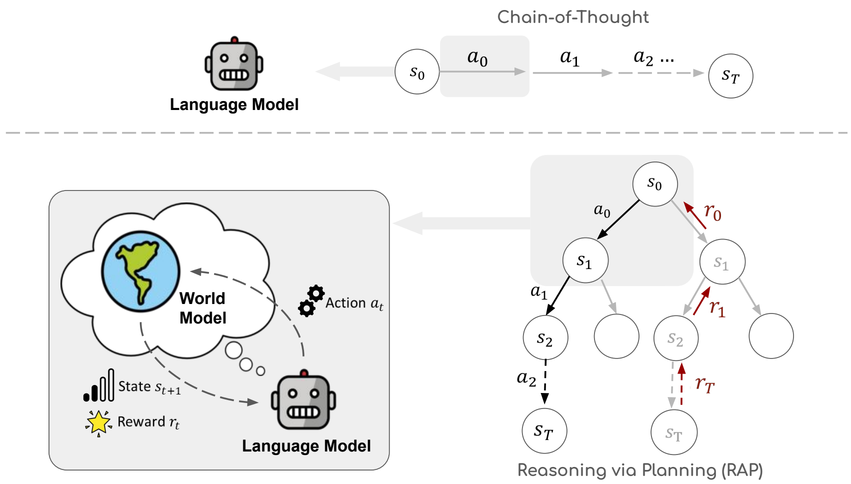

Figure 1: An overview of Reasoning via Planning (RAP). Compared with previous LLM reasoning methods like Chain-of-Thought [59], we explicitly model the world state from a world model (repurposed from the language model), enabling us to leverage advanced planning algorithms to solve the reasoning problems.

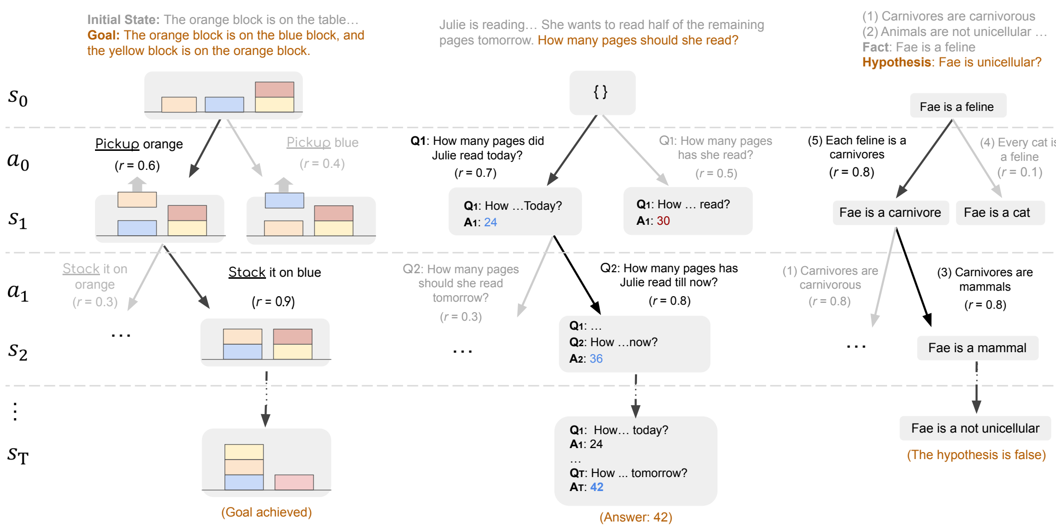

Figure 2: Examples of RAP for plan generation (left), math reasoning (middle), and logical reasoning (right).

an autoregressive manner. In particular, we identify several key limitations of the current reasoning with LLMs, including **(1)** the lack of an internal world model to simulate the *state* of the world (e.g., the configuration of blocks, the values of intermediate variables), which is the foundation of human planning; **(2)** the absence of a *reward* mechanism to assess and guide the reasoning towards the desired state; and due to both of these limitations, **(3)** the incapability of balancing *exploration vs. exploitation* to efficiently explore the vast reasoning space.

To address these limitations, this paper proposes a new framework, **Reasoning via Planning (RAP)**, that enables LLMs to reason in a manner close to humans' conscious planning. RAP augments the LLM with a world model, and reasons with principled planning (specifically *Monte Carlo Tree Search, MCTS*) to produce high-reward reasoning traces after efficient exploration (Figure 1). Notably, we acquire the world model by repurposing the LLM itself with appropriate prompts. During the reasoning, the LLM strategically builds a reasoning tree by iteratively considering the most promising reasoning steps (*actions*) and using the world model (the same, repurposed LLM) to look ahead for future outcomes. The estimated future rewards are then backpropagated to update the LLM's beliefs about the current reasoning steps, guiding it to refine the reasoning by exploring better alternatives. Our MCTS-based planning effectively maintains a proper balance between exploration (of unvisited reasoning traces) and exploitation (of the best reasoning steps identified so far).

We show RAP is a general framework applicable to a diverse range of challenging problems and achieves substantial improvements over recent popular LLM reasoning methods. In Blocksworld [57] for 2/4/6-step plan generation, RAP achieves an average success rate of 64% while CoT fails almost completely. Moreover, LLaMA-33B with RAP surpasses GPT-4 with CoT by 33% relative improvement. In math reasoning (GSM8K [11]) and logical inference (PrOntoQA [47]), RAP also consistently improves over CoT, least-to-most prompting, and their self-consistency variants.

2 Related Work

Reasoning with LLMs. In the realm of LLMs [22, 41, 6], reasoning typically entails decomposing complex questions into sequential intermediate steps (a.k.a. chains) before producing the final answer, exemplified by chain-of-thought (CoT) prompting and its variants [43, 59, 32]. The basic CoT approaches, which generate chains all at once, can induce additional errors as the step count increases. One line of improvement methods involves sampling multiple chains and choosing the best answer via majority voting, such as self-consistency (SC) [58]. Another line of work focuses on decomposition, aiming to tackle the problem by solving multiple simple subproblems. For instance, least-to-most prompting [66] reduces the question into subquestions and answers them sequentially. More relevantly, similar to our reward formulation, some recent works have explored self-evaluation approaches, which leverage LLMs themselves to provide feedback for intermediate steps and then continue the reasoning [60, 51, 45]. For example, Paul et al. [45] fine-tune a critic model to provide structured feedback iteratively in each step, and Madaan et al. [38] directly reuse the same LLM to generate multi-aspect feedback and refine the previously generated output. Besides, aligned with our state formulation, Li et al. [34] incorporates latent "situations" into LLMs, referring to the state of entities from the context. Nevertheless, none of the above methods formally introduce the world model and instantiates the reward and state into a unified framework.

Search-guided Reasoning with LLMs. Most of CoT approaches discussed above are based on a linear reasoning structure. Self-consistency built onto CoT decodes multiple chains parallelly, but it remains hard to explore the reasoning space sufficiently. Recent efforts have been made to investigate non-linear reasoning structures by sampling more reasoning steps efficiently guided by some search algorithms [30, 67, 63, 64]. For example, Jung et al. [30] generate a tree of explanations to enforce logical consistency, and Xie et al. [63] adopt beam search to decode a better CoT reasoning chain. More recently, CoRe [67] proposes to fine-tune both the reasoning step generator and verifier for solving math word problems, also using MCTS for reasoning decoding. Concurrently to our work, Yao et al. [64] apply heuristic-based approach, like depth-/breadth-first search, to search for better reasoning paths. Compared with these search-guided methods, RAP is a more principled framework that combines world model and reward with MCTS planning. The RAP formulation of LLM reasoning with state, action, and reward also presents a more general approach applicable to a wide range of reasoning problems.

Planning with LLMs. Planning, a central ability in intelligent agents, involves generating a series of actions to achieve a specific goal [40, 7]. Classical planning methods have been widely adopted in robots and embodied environments [9, 42, 8, 61, 26]. Recently, prompting LLMs to do planning direcly has gained attention and shown potential [24, 23, 53, 13, 35]. SayCan [1], for instance, combines LLMs with affordance functions to generate feasible plans. Moreover, based on LLMs' powerful programming ability [37, 29, 36], some recent works first translate natural language instructions into the executable programming languages, such as Planning Domain Description Language (PDDL), and runs classical planning algorithms, such as LLM+P [36]. However, code-based planning is constrained by its narrow domains and the predefined environment, while RAP can handle open domain problems, including numerical and logical reasoning (see Section 4.2 and 4.3). More related works on planning and world model are discussed in Appendix A.

3 Reasoning via Planning (RAP)

In this section, we present the Reasoning via Planning (RAP) framework that enables LLMs to strategically plan a coherent reasoning trace for solving a wide range of reasoning tasks. We first build the world model by repurposing the LLM with prompting (Section 3.1). The world model serves as the foundation for deliberate planning, by allowing the LLM to plan ahead and seek out the expected outcomes in the future. We then introduce the rewards for assessing each state during reasoning in Section 3.2. Guided by the world model and rewards, the planning with Monte Carlo Tree Search (MCTS) efficiently explores the vast reasoning space and finds optimal reasoning traces

(Section 3.3). Finally, when multiple promising reasoning traces are acquired during planning, we further introduce an aggregation method in Section 3.4 that yields an integrated result and further boosts the reasoning performance.

3.1 Language Model as World Model

In general, a world model predicts the next *state* of the reasoning after applying an *action* to the current state [17, 39]. RAP enables us to instantiate the general concepts of state and action in different ways depending on the specific reasoning problems at hand. For example, in Blocksworld (Figure 2 left), it is natural to set a state to describe a configuration of blocks (with natural language), and an action to be a behavior of moving a block (e.g., `‘‘pickup the orange block’’`). In a math reasoning problem (Figure 2 middle), we use the state to represent the values of intermediate variables, and set an action to be a subquestion that drives the reasoning to derive new values (i.e., new state).

After the definition of state and action, the reasoning process can thus be described as a Markov decision process (MDP): given the current state $s_{t,t=0,1,\dots,T}$, e.g., the initial state s_0, the LLM (as a reasoning agent) samples several actions as the action space, following its generative distribution $a_t \sim p(a|s_t, c)$, where c is a proper prompt (e.g., in-context demonstrations) to steer the LLM for action generation. The world model then predicts the next state s_{t+1} of the reasoning. Specifically, we repurpose the *same* LLM to obtain a state transition distribution $p(s_{t+1}|s_t, a_t, c')$, where c' is another prompt to guide the LLM to generate a state. For instance, in Blocksworld, the LLM (as the world model) generates text s_{t+1} to describe the new configuration of blocks, given the previous state description s_t and the action a_t.

Continuing the process results in a reasoning trace, which consists of a sequence of interleaved states and actions $(s_0, a_0, s_1, \dots, a_{T-1}, s_T)$. This differs from the previous reasoning methods, such as Chain-of-Thought [59], where the intermediate reasoning steps consist of only a sequence of actions, e.g., ($a_0 =$ `‘‘pickup red block’’`, $a_1 =$ `‘‘stack on yellow block’’`, ...) (see comparisons in Figure 1). Augmenting the reasoning with the (predicted) world states helps the LLM with a more grounded and coherent inference. Note that the full reasoning trace is simulated by the LLM itself (as a reasoning agent with an *internal* world model) without interacting with the *external* real environment. This resembles humans contemplating a possible plan in their minds. The capability of simulating future states, due to the introduction of the world model, allows us to incorporate principled planning algorithms to efficiently explore the vast reasoning space as described in Section 3.3.

3.2 Reward Design

During reasoning, we want to assess the feasibility and desirability of each reasoning step, and guide the reasoning based on the assessment (Section 3.3). The assessment of each reasoning step (i.e., applying an action a_t to the state s_t) is performed by a *reward* function $r_t = r(s_t, a_t) \in \mathbb{R}$. Similar to the state and action, the reward function can be specified in different ways to accommodate any knowledge or preferences about the reasoning problem of interest. Here we introduce several common rewards applicable to different tasks and shown to be effective in our experiments.

Likelihood of the action. When an action is generated by the LLM conditioning on the in-context demonstration and the current state, the probability of the specific action reflects the LLM's preference. We thus can incorporate the log probability of the action as a reward. This reward reflects the "instinct" of LLMs as an agent, and can be also used as a prior for which action to explore.

Confidence of the state transition. State prediction is nontrivial in some problems, e.g., in math reasoning (Figure 2, middle), given an action (i.e., a subquestion), the world model updates the set of known variables by answering the subquestion. Since LLMs may make mistakes when answering these questions, We incorporate the confidence of the state transition (i.e., the answer to a subquestion in this case) as a reward. Specifically, we sample multiple answers from the language model, and use the proportion of the most frequent answer as the confidence. A high confidence indicates a reliable reasoning step, which is worth more exploration in the future.

Self-evaluation by the LLM. It's sometimes easier to recognize the errors in reasoning than avoid generating them in advance. Thus, it's beneficial to allow the LLM to criticize itself with the question "`Is this reasoning step correct?`", and use the next-word probability of the token "`Yes`" as a reward. The reward evaluates LLM's own estimation of the correctness of reasoning. Note that the specific problems for self-evaluation can be different depending on the tasks.

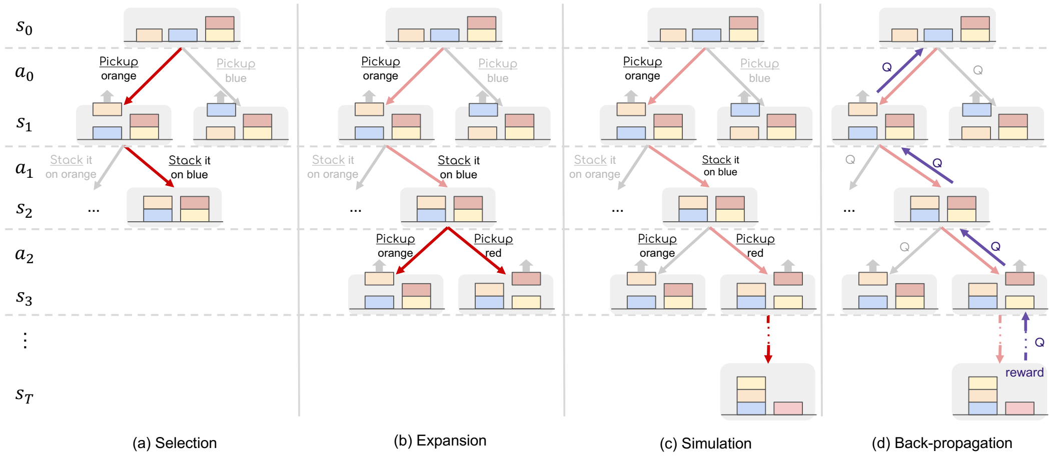

Figure 3: An illustration of the four phases in an iteration in MCTS planning (Section 3.3).

Task-specific heuristics. We can also flexibly plug-in other diverse task-specific heuristics into the reward function. For example, in plan generation for Blocksworld, we compare the predicted current state of blocks with the goal to calculate a reward (Section 4.1). The reward encourages the plan of movements to actively pace towards the target.

3.3 Planning with Monte Carlo Tree Search

The world model (Section 3.1) and rewards (Section 3.2) enable LLMs to reason with advanced planning algorithms, where we adopt Monte Carlo Tree Search (MCTS) [31, 12], a powerful planning algorithm that strategically explores the space of reasoning trees, and strikes a proper balance between exploration and exploitation to find a good reasoning trace efficiently.

MCTS builds a reasoning tree iteratively, where each node represents a state, and each edge represents an action and the transition from the current state to the next state after applying the action (Figure 1). To guide the LLM agent to expand and explore the most promising nodes of the tree, the algorithm maintains a state-action value function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, where $Q(s, a)$ estimates the *expected future reward* of taking action a in state s. That is, we assess the potential of a node (or a reasoning step) by looking ahead and anticipating the reward in future trajectories starting from this node. This fundamentally differs from the current reasoning methods that generate a reasoning trace autoregressively from left to right without accounting for the future.

Specifically, as illustrated in Figure 3, the MCTS planning performs four operations in each iteration to expand the tree and update Q values, i.e., *selection*, *expansion*, *simulation*, and *back-propagation*. The process continues until a specified computational budget (e.g., the number of iterations) is reached, and the resulting reasoning traces are acquired from the tree, as we articulated later. The psuedo-code and more implementation details are presented in Algorithm 1 and Appendix B.

Selection. The first phase selects a portion of the existing tree that is most promising for further expansion in the next phase. Specifically, starting from the root node (i.e., initial state s_0), at each level of the tree, the algorithm selects a child node as the next node. The phase finishes when a leaf node of the current tree is reached. Figure 3(a) highlights the selected path in red. To balance between exploration (of less-visited nodes) and exploitation (of high-value nodes), we use the well-known *Upper Confidence bounds applied to Trees (UCT)* algorithm [31] to select each child node. Specifically, at node s, we select the action (which leads to a transition to a child node) in the tree by considering both the Q value (for exploitation) and uncertainty (for exploration):

$$a^* = \arg\max_{a \in A(s)} \left[Q(s,a) + w \sqrt{\frac{\ln N(s)}{N(c(s,a))}} \right], \tag{1}$$

where $N(s)$ is the number of times node s has been visited in previous iterations, and $c(s, a)$ is the child node of applying a in state s. Therefore, the less a child node was visited before (i.e., the more uncertain about this child node), the higher the second term in the equation. The weight w controls the balance between exploration and exploitation.

Expansion. This phase expands the tree by adding new child nodes to the leaf node selected above. Specifically, given the state of the leaf node, we use the LLM (as agent) to sample d possible actions (e.g., subquestions in math reasoning), and then use the LLM (as world model) to predict the respective next state, resulting in d child nodes. Note that if the leaf node selected above is a terminal node (the end of a reasoning chain) already, we will skip expansion and jump to back-propagation.

Simulation. This phase simulates the reasoning chain to the end in order to estimate the expected future rewards (Q values). Specifically, starting from the current node s, we iteratively select an action following a *roll-out policy* and use the world model to predict the next state. The roll-out process continues until a terminal state is reached. While the design of the roll-out policy is flexible, in our experiments, we generate d candidate actions and pick the one with the largest local reward $a' = \max_{a'} r(s, a)$. In practice, as the roll-out process will evaluate the reward function for multiple nodes, for efficiency, we discard the computationally expensive components in r (for example, the reward of the state transition confidence requires sampling the answer multiple times), and use the resulting light-weight reward function for selecting actions during simulation.

Back-propagation. Once we reach a terminal state in the above phases, we obtain a reasoning path from the root node to the terminal node. We now back-propagate the rewards on the path to update the Q value of each state-action pair along the path. That is, $Q(s, a)$ is updated by aggregating the rewards in all future steps of node s. We may adopt the aggregation method according to the nature of different tasks and reward design, as discussed in Section 4.

As mentioned earlier, once a predetermined number of MCTS iterations is reached, we terminate the algorithm and select final reasoning trace from the constructed tree. There could be various ways for the selection. One approach is to start from the root node and iteratively choose the action with the highest Q value until reaching a terminal. Alternatively, one can directly select the path from the iterations that yielded the highest reward, or opt to choose the leaf node (and the respective root-to-leaf path) that has been visited the most. In practice, we observed that the second strategy often yields the best results.

3.4 RAP-Aggregation: Aggregating Multiple Reasoning Outputs

Ensemble-based methods, such as self-consistency CoT [58], can effectively improve performance by aggregating multiple valid reasoning traces. Therefore, for problems, such as math reasoning (Section 4.2) where only the final answer is required, RAP could produce multiple traces and answers from different MCTS iterations, which will be aggregated to produce the final answer. We refer to such a mechanism as RAP-Aggregation. Note that problems like plan generation or logical inference require a complete reasoning trace as output; thus, RAP-Aggregation will not be applied.

More importantly, there is a concern that some incorrect reasoning steps may appear in the early stage of multiple iterations, thus polluting the aggregation. As a result, we further devise a new weighting strategy for aggregating candidate answers. Specifically, for each candidate answer, we accumulate the reward of each reasoning step in the answer's reasoning traces. We choose the answer with the highest accumulative reward as the final aggregated answer.

4 Experiments

In this section, we demonstrate the flexibility and effectiveness of our RAP framework by applying it to a wide range of problems, including plan generation in an embodied environment (4.1), mathematical reasoning for solving math word problems (4.2), and logical reasoning for verifying hypotheses (4.3). The subsequent sections demonstrate how the world model formulation in RAP enables a versatile design of the state and action, catering to various reasoning contexts. We also discuss the choice of reward in Appendix C.

We primarily compare RAP with chain-of-thought (CoT) [59], and its variants like least-to-most prompting [66] as baselines. We also consider ensembling multiple reasoning paths if applicable (also known as self-consistency [58]). Moreover, we compare RAP with GPT-4 [44] when computation resources allow. By default, we use the LLaMA-33B model [56] as the base LLM for both our methods and baselines, with a sampling temperature of 0.8. All prompts are shown in Appendix D.

4.1 Plan Generation

The plan generation task aims to produce a sequence of actions to achieve a given goal, possibly with additional constraints. The ability to generate plans is important for intelligent embodied agents,

Table 1: Results on Blocksworld. RAP$^{(10)}$ and RAP$^{(20)}$ refer to our method where the iteration number is set to 10 and 20, respectively. "pass@10" is a relaxed metric, where 10 plans are sampled for each test case, and the test case regarded as solved if at least one plan is successful. For all other settings including RAP, only a single plan is evaluated.

Method	2-step	4-step	6-step
CoT	0.17	0.02	0.00
CoT - pass@10	0.23	0.07	0.00
CoT (GPT-4)	0.50	0.63	0.40
RAP$^{(10)}$	1.00	0.86	0.26
RAP$^{(20)}$	**1.00**	**0.88**	**0.42**

e.g. household robots [46]. This task has also been widely used to evaluate the reasoning ability of LLMs given their challenging requirements of long-horizon reasoning, e.g., Blocksworld is a classic problem, where an agent is asked to rearrange the blocks into stacks in a particular order.

Task setup. To explore the viability of the RAP framework for plan generation tasks, we adapt and evaluate RAP on the Blocksworld benchmark [50]. We define a **state** as the current orientation of the blocks and an **action** as an instruction that moves blocks. Specifically, an action is composed of one of the 4 verbs (i.e., STACK, UNSTACK, PUT, and PICKUP) and manipulated objects. For the action space, we generate the currently valid actions given the domain restrictions on actions and the current orientation of the blocks. To transit between states, we take the current action and query the LLM to predict the state changes to the relevant blocks. We then update the current state by adding the new block conditions and removing the conditions that are no longer true. Once a state has met all of the conditions listed in the goal or the depth limit of the tree is reached, we terminate the associated node.

To assess the quality of actions within this domain, we use two separate **rewards**. First, we prompt the LLM with some example test cases along with their solutions, and then calculate the log probability of the action given the current state (*"Likelihood of action"* reward in Section 3.2), denoted as r_1. This reward reflects the intuition of the LLM as the reasoning agent. It's typically indicative when there are few steps left to the goal, while not as reliable for a distant goal. Additionally, we compare the new state after performing an action with the goal and provide a reward, r_2, scaling with the number of conditions met (*"Task-specific heuristics"* reward). Specifically, when all the conditions are met, we assign a super large reward to make sure this plan will be selected as the solution.

Results. We use test cases from the Blocksworld dataset [57] and group them by solvable steps, resulting in 30 cases solvable with 2 steps, 57 cases with 4 steps, and 114 cases with 6 steps. There are at most 5 blocks in each test case. As the baseline method, we prompt the LLM with 4 test cases with corresponding solutions, and ask it to generate a plan for a new question. This setting is the same as one described in Valmeekam et al. [57], and we denote it as Chain-of-Thought (CoT) for briefness. For RAP, the same prompt is shown to help LLMs calculate r_1.

As shown in Table 1, CoT with LLaMA-33B can only generate successful plans for a few 2-step cases, and completely fails on harder problems. RAP substantially improves over CoT by nearly solving all problems within 4 steps, and a part of 6-step problems, achieving an average success rate of 64%. It's worth noting that the searching space of 6-step problems can be as large as 5^6, while our algorithm can find a successful plan 42% of the time within 20 iterations. Even more, our framework allows LLaMA-33B to outperform GPT-4 by 33% relative improvement [44], which is known to have much stronger reasoning ability [6].

We further present a case study of comparing the reasoning paths from Cot and RAP. As illustrated in Figure 4, we find the improvement can be mainly attributed to the following reasons: (1) By maintaining the world state during reasoning, RAP can recognize valid actions for the current state, avoiding generating illegal plans. (2) RAP is capable of backtracking and trying out other solutions when the first intuition from the LLM doesn't work. Specifically, CoT attempts to achieve the second goal, i.e. "orange on red", and achieve that with the first two steps. However, accomplishing the second goal first would prevent the first goal from being satisfied. On the contrary, even though RAP makes the same mistakes in the first iterations, our framework drives the agent to explore other possible paths (as described in Section 3.3) and finally generate a successful plan. (3) When calculating r_t, we can feed only the current state to the LLM and hide the history. E.g., in the case

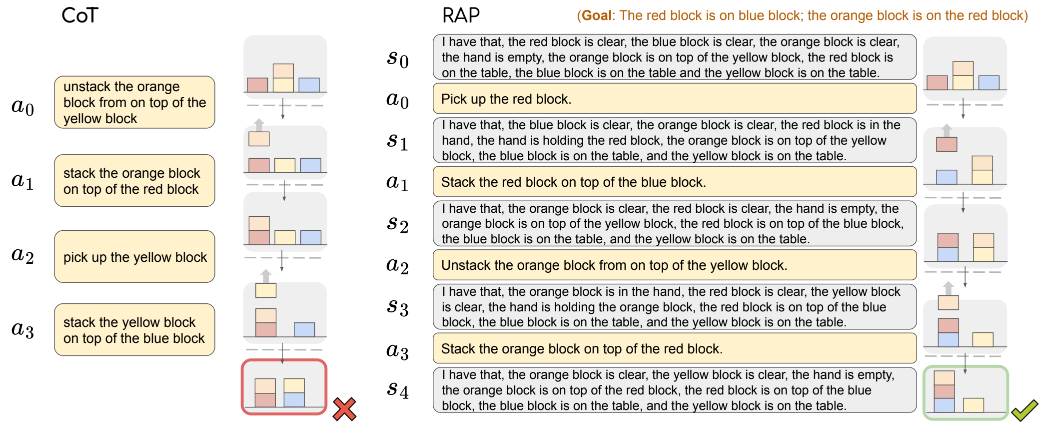

Figure 4: Comparing reasoning traces in Blocksworld from CoT (left) and RAP (right).

of Figure 4, to calculate the reward for a_2, the LLM is provided with a "new" test case, in which s_2 is the initial state. This significantly lowers the difficulties of the last few steps, and saves more iterations for harder decisions of the first few steps.

4.2 Math Reasoning

Task setup. Numerical reasoning tasks, such as GSM8k [11], often include a description and a final question. To arrive at the answer to the final question, it is necessary to undertake multi-step mathematical calculations based on the problem's context. It is thus natural to decompose the final question into a sequence of smaller sub-questions (Figure 2, right). To adapt RAP, we define a **state** as the values of intermediate variables, while an **action** is to propose an incremental sub-question about a new intermediate variable. The world model then responds to the sub-question using the intermediate variables and the problem description, adding the new intermediate variable value into the next state. We combine the self-evaluation by LLM $r_{t,1}$ and the confidence of state transition $r_{t,2}$ using weighted geometric mean $r_t = r_{t,1}^{\alpha} * r_{t,2}^{1-\alpha}$ as the **reward** function. This reward encourages more relevant and useful sub-questions. To account for the impact of the reasoning path's length on the reward, we compute **the Q value** by using the maximum of average rewards in future steps.

$$Q^*(s_t, a_t) = \max_{s_t, a_t, r_t, \ldots, s_l, a_l, r_l, s_{l+1}} \operatorname{avg}(r_t, \ldots, r_l). \quad (2)$$

As a related work, Least-to-Most prompting [66] shares a similar idea to us in sub-question decomposition, but they generate sub-questions all at once. On the contrary, RAP considers each action a_t based on the current state s_t, which enables more informed decisions.

Results. We evaluate our framework on GSM8k, a dataset of grade high school math problems. We also evaluate the base model with CoT prompting [59], Least-to-Most prompting [66], and their self-consistency [58] variants, as the baselines. We use the same 4-shot examples demonstrations for both our framework and the baselines.

As shown in Table 2, our RAP framework answers 48.8% of the problems correctly, outperforming both the Chain-of-Thought and the Least-to-Most prompting with self-consistency. All methods use the same set of examples as the in-context demonstration. Notably, this result is achieved when RAP selects only one reasoning trace based on the reward. The introduction of RAP-Aggregate further improves the accuracy by $\sim 3\%$. We also calculate the accuracy with different numbers of iterations in MCTS and self-consistency samples in baselines, as illustrated in Figure 2. We find that across all numbers of iterations/samples, RAP-Aggregation outperforms baselines consistently, which indicates that when only a few iterations/samples are allowed, our framework is significantly better at finding reliable reasoning paths with the guide of reward.

4.3 Logical Reasoning

Task setup. A logical reasoning task (e.g. PrOntoQA [47]) typically provides a set of *facts* and *logical rules*, and a model is required to verify if a *hypothesis fact* is true or false by applying the logical rules to the given facts, as illustrated in Figure 2. These tasks not only require the correct final answer (true/false), but also a detailed proof demonstrating the result. To apply our framework,

Table 2: Results on GSM8k. The superscripts indicate the number of samples or iterations.

Method	Accuracy (%)
Chain-of-Thought	29.4
+ $\text{SC}^{(10)}$	46.8
Least-to-Most	25.5
+ $\text{SC}^{(10)}$	42.5
$\text{RAP}^{(1)}$	40.0
$\text{RAP}^{(10)}$	48.6
+ aggr	**51.6**

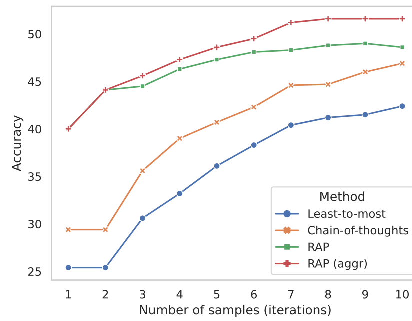

Figure 5: The performance of RAP and baselines on GSM-8K, with different numbers of sampled paths or iterations.

we define the **state** as a fact we are focusing on, analogous to the human's working memory [3] for inference. An **action** is defined as selecting a rule from the fact set. The world model performs a one-hop reasoning step to get a new fact as the next state. The **reward** is calculated with Self-evaluation (Section 3.2. Specifically, we prompt the LLM with a few examples with their labels to help it better understand the quality of reasoning steps. We use the average reward of future steps to update **the Q function**, the same as Equation (2) for GSM8k.

Results. We assess the performance of our RAP framework on PrOntoQA [47]. We adopt their settings of "true" ontology (using real-world knowledge), "random" ordering of rules. We mix the examples requiring 3, 4, and 5 reasoning hops in a correct proof to prevent LLM from memorizing when to finish the reasoning. We sample 500 examples from the generation script released by Saparov and He [47]. We compare both the prediction accuracy of the final answer and the accuracy of the entire proof. We do 20 iterations for MCTS and 20 samples for self-consistency in baselines.

Table 3: Results on ProntoQA.

Method	Pred Acc	Proof Acc
CoT	87.8	64.8
CoT + SC	89.8	-
RAP (Ours)	**94.2**	**78.8**

As the results presented in Table 3, our framework achieves an output accuracy of 94.2% and a proof accuracy of 78.8%, surpassing the CoT baseline by 14% proof accuracy and the self-consistency CoT baseline by 4.4% prediction accuracy. Such substantial improvements clearly demonstrate the effectiveness of RAP in solving logical reasoning problems in the PrOntoQA dataset. That is because RAP can effectively recognize the error when a reasoning chain comes to a dead end, and propagate the signal back to earlier reasoning steps, with the planning algorithm allowing it to explore alternatives to the previous steps (Figure 2). The self-evaluation reward performs as a prior, which prioritizes the most promising actions for exploration.

5 Conclusion

In this paper, we present Reasoning via Planning (RAP), a novel LLM reasoning framework that equips LLMs with an ability to reason akin to human-like strategic planning. By coupling the LLMs' reasoning capabilities with a world model and principled planning via Monte Carlo Tree Search, RAP bridges the gap between LLMs and human planning capabilities. Our framework, which repurposes the LLM to act as both a world model and a reasoning agent, enables the LLM to simulate states of the world and anticipate action outcomes, while achieving an effective balance between exploration and exploitation in the vast reasoning space. Extensive experiments on a variety of challenging reasoning problems demonstrate RAP's superiority over several contemporary CoT-based reasoning approaches, and even the advanced GPT-4 in certain settings. RAP's flexibility in formulating rewards, states, and actions further proves its potential as a general framework for solving diverse reasoning tasks. We posit that RAP, with its innovative melding of planning and reasoning, has the potential to redefine the way we approach LLM reasoning - essentially forging a new pathway toward achieving human-level strategic thinking and planning in artificial intelligence.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[2] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[3] Alan Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.

[4] Robert Eamon Briscoe. Mental imagery and the varieties of amodal perception. *Pacific Philosophical Quarterly*, 92(2):153–173, 2011.

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

[6] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[7] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[8] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[9] Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso. Prodigy: An integrated architecture for planning and learning. *ACM SIGART Bulletin*, 2(4):51–55, 1991.

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[12] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pages 72–83. Springer, 2007.

[13] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247*, 2023.

[14] Wojciech W Gasparski and Tufan Orel. *Designology: Studies on Planning for Action*, volume 1. Transaction Publishers, 2014.

[15] Dedre Gentner and Albert L Stevens. *Mental models*. Psychology Press, 2014.

[16] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

[17] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[18] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[19] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[20] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[21] Mark K Ho, David Abel, Carlos G Correa, Michael L Littman, Jonathan D Cohen, and Thomas L Griffiths. Control of mental representations in human planning. *arXiv e-prints*, pages arXiv–2105, 2021.

[22] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*, 2022.

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

[24] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[25] Quentin JM Huys, Neir Eshel, Elizabeth O'Nions, Luke Sheridan, Peter Dayan, and Jonathan P Roiser. Bonsai trees in your head: how the pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS computational biology*, 8(3):e1002410, 2012.

[26] Yu-qian Jiang, Shi-qi Zhang, Piyush Khandelwal, and Peter Stone. Task planning in robotics: an empirical comparison of pddl-and asp-based systems. *Frontiers of Information Technology & Electronic Engineering*, 20:363–373, 2019.

[27] Philip N Johnson-Laird. Mental models and human reasoning. *Proceedings of the National Academy of Sciences*, 107(43):18243–18250, 2010.

[28] Philip Nicholas Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Number 6. Harvard University Press, 1983.

[29] Ana Jojic, Zhen Wang, and Nebojsa Jojic. Gpt is becoming a turing machine: Here are some ways to program it. *arXiv preprint arXiv:2303.14310*, 2023.

[30] Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. Maieutic prompting: Logically consistent reasoning with recursive explanations. *arXiv preprint arXiv:2205.11822*, 2022.

[31] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, pages 282–293. Springer, 2006.

[32] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

[33] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.

[34] Belinda Z Li, Maxwell Nye, and Jacob Andreas. Language modeling with latent situations. *arXiv preprint arXiv:2212.10012*, 2022.

[35] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

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

[37] Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023.

[38] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

[39] Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. Deep learning, reinforcement learning, and world models. *Neural Networks*, 2022.

[40] John McCarthy. Situations, actions, and causal laws. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1963.

[41] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

[42] Dana S Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *Journal of artificial intelligence research*, 20: 379–404, 2003.

[43] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[44] OpenAI. Gpt-4 technical report, 2023.

[45] Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*, 2023.

[46] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs, 2018.

[47] Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.

[48] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[49] Jay Schulkin. *Action, perception and the brain: Adaptation and cephalic expression*. Springer, 2012.

[50] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[51] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *ArXiv*, abs/2303.11366, 2023.

[52] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[53] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[54] Edward C Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

[55] Marc Toussaint. Learning a world model and planning with a self-organizing, dynamic neural system. *Advances in neural information processing systems*, 16, 2003.

[56] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[57] Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. On the planning abilities of large language models (a critical investigation with a proposed benchmark). *arXiv preprint arXiv:2302.06706*, 2023.

[58] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[59] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[60] Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. *arXiv preprint arXiv:2211.00053*, 2022.

[61] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018.

[62] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning*, pages 2226–2240. PMLR, 2023.

[63] Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. Decomposition enhances reasoning via self-evaluation guided decoding. *arXiv preprint arXiv:2305.00633*, 2023.

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

[65] Ping Yu, Tianlu Wang, Olga Golovneva, Badr Alkhamissy, Gargi Ghosh, Mona Diab, and Asli Celikyilmaz. Alert: Adapting language models to reasoning tasks. *arXiv preprint arXiv:2212.08286*, 2022.

[66] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

[67] Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. Solving math word problem via cooperative reasoning induced language models. *arXiv preprint arXiv:2210.16257*, 2022.

Algorithm 1 RAP-MCTS

Require: Initial state s_0, state transition probability function p_θ, reward function r_θ, action generator p_ϕ, number of generated actions d, depth limit L, number of roll-outs N, and exploration weight w

Initialize memory of actions $A : \mathcal{S} \mapsto \mathcal{A}$, children $c : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ and rewards $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$
Initialize the state-action value function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ and visit counter $N : \mathcal{S} \mapsto \mathbb{N}$
for $n \leftarrow 0, \ldots, N-1$ **do**
    $t \leftarrow 0$
    while $N(s_t) > 0$ **do** ▷ Selection
        $N(s_t) \leftarrow N(s_t) + 1$
        $a_t \leftarrow \arg\max_{a \in A(s_t)} \left[Q(s_t, a) + w \sqrt{\frac{\ln N(s_t)}{N(c(s_t, a))}} \right]$
        $r_t = r(s_t, a_t)$, $s_{t+1} \leftarrow c(s_t, a_t)$
        $t \leftarrow t + 1$
    end while
    while s_t is not a terminal state $\wedge\ t \leq L$ **do**
        for $i \leftarrow 1, \ldots, d$ **do** ▷ Expansion
            Sample $a_t^{(i)} \sim p_\phi(a \mid s_t)$, $s_{t+1}^{(i)} \sim p_\theta(s_t, a_t^{(i)})$, and $r_t^{(i)} \sim r_\theta(s_t, a_t^{(i)})$
            Update $A(s_t) \leftarrow \{a_t^{(i)}\}_{i=1}^{d}$, $c(s_t, a_t^{(i)}) \leftarrow s_{t+1}^{(i)}$, and $r(s_t, a_t) \leftarrow r_t^{(i)}$
        end for
        $a_{t+1} \leftarrow \arg\max_{a \in A(s_t)} r(s_t, a_t)$ ▷ Simulation
        $r_t \leftarrow r(s_t, a_t)$, $s_{t+1} \leftarrow c(s_t, a_t)$
        $t \leftarrow t + 1$
    end while
    for $t' \leftarrow t, \ldots, 0$ **do** ▷ Back propagation
        Update $Q(s_{t'}, a_{t'})$ with $\{r_{t'}, r_{t'+1}, \ldots, r_t\}$
    end for
end for

A Related work: Planning and World Model

Recent years have witnessed successful applications of planning algorithms [50], such as AlphaZero [52], and MuZero [48]. These algorithms are typically based on tree-structured search and are designed to effectively maintain the balance of exploration and exploitation. Knowledge of transition dynamics is the prerequisite for planning, and recent research on model-based reinforcement learning proposed to learn a world model (or dynamics model) to plan or assist policy learning. To improve sample efficiency, previous research attempts to learn a world model from offline trajectories, and directly learn a policy within the world model [16, 17]. With latent imagination in a world model, RL agents can be trained to solve long-horizon tasks [18, 20]. Besides, the world model is also shown to be helpful to physical robot learning [62]. In this paper, we use LLMs as world models and apply a planning algorithm to search for a reasoning path. This is similar in spirit to model predictive control [8]. Compared with previous works, our framework uses general LLMs as the world model and can be adapted to a wide range of open-domain reasoning tasks.

B MCTS Planning

We adapt MCTS to search for the optimal reasoning path (Algorithm 1). Compared with traditional applications of MCTS, we are faced with a large reasoning space, and the heavy computational cost of LLMs. Thus, we made several modifications to the classic MCTS in our implementation: (1) For open domain problems, e.g., math problems, it's impossible to enumerate all actions (subquestions), so we reduce the action space by sampling a fixed number of potential actions from LLMs, conditioned on a prompt of the current state and in-context demonstration. (2) In the selection phase, if there are actions that haven't been visited before, we estimate the Q value with lightweight local rewards, e.g., self-evaluation reward, and then select the action with UCT. This provides prior knowledge for the exploration, which is crucial given the limited iteration budgets.

Table 4: Ablation study on Blocksworld. R_1 is action likelihood reward, R_2 is task-specific reward, and R_3 is self-evaluation reward.

R_1	R_2	R_3	Success
✓	✓	✗	0.88
✓	✓	✓	0.91
✓	✗	✗	0.46
✗	✓	✗	0.21
✗	✗	✓	0.14
✗	✗	✗	0.02

Table 5: Ablation study on GSM8k (first 300 examples). R_1 is state transition confidence reward, R_2 is action likelihood reward, and R_3 is self-evaluation reward.

R_1	R_2	R_3	RAP$^{(1)}$	RAP$^{(10)}$	+aggr
✓	✗	✓	0.410	0.450	0.503
✓	✗	✗	0.350	0.447	0.490
✓	✓	✗	0.373	0.423	0.443

C Reward Choice

Similar to the reward design in RL research, the rewards in LLM reasoning also require some curation and design. In our main experiments, we choose the combination of rewards in our current experiments based on specific task heuristics and our exploratory experiments. To understand the effects of the reward choice for LLM reasoning, we supplement comprehensive experiments on rewards for plan generation (Table 4) and math reasoning (Table 5). Note that, in both tables, the first row indicates the setting we use in the main experiments.

Experiment results. As shown in Table 4, the combination of action likelihood and task-specific reward (row 1) can significantly outperform the single reward baselines (row 3, 4, 5). Interestingly, adding the self-evaluation reward can further improve the performance slightly (row 2). Furthermore, as the results on the first 300 samples of GSM8k shown in Table 5, we can see adding either action likelihood (row 3) or self-evaluation (row 1) on top of confidence reward (row 2) can boost the RAP performance of only using confidence reward (row 1) with one iteration, but action likelihood reward downgrades the accuracy with more iterations. The self-evaluation reward leads to the best performance overall. This indicates the importance of self-evaluation reward in guiding reasoning as an effective and computationally efficient prior to exploration.

On self-evaluation and action likelihood reward. The rewards of self-evaluation and action likelihood are of particular interest, as they can be applied to a wide range of reasoning tasks. Generally, the best usage and combination with other rewards require empirical design and understanding of the task nature, and their effectiveness can vary significantly across different tasks. Here, we provide some intuitions behind the reward choices:

(a) For the problems in which one reasoning step is short and structured, the action likelihood can be very indicative. Otherwise, it may be disturbed by unimportant tokens and become unreliable. For instance, a single step within the Blocksworld domain typically adheres to specific patterns (e.g., PICK/PUT/STACK a block. . .), rendering the action likelihood indicative. However, in the math domain, a reasoning step is expressed in natural language sentences, allowing for greater freedom and potentially introducing noise.

(b) For the problems where it's easier to recognize some errors afterward than avoid them during generation, self-evaluation emerges as a helpful mechanism for enhancing reasoning accuracy. In mathematical reasoning, LLMs may struggle to generate a correct reasoning step in the first place, but the detection of calculation or logic errors is more feasible. In Blocksworlds, however, assessing the quality of a candidate action is not straightforward and still requires multi-step reasoning. This characteristic diminishes the accuracy of the self-evaluation reward, making it less helpful especially given that likelihood already provides a good intuition for search.

D Prompt

D.1 Plan Generation

We show the prompt to calculate the action likelihood for RAP below. The same prompt is also applied in CoT baseline. <init_state> and <goals> would be instantiated by the problem to solve.

```
I am playing with a set of blocks where I need to arrange the blocks into
    stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is
    clear. A block is clear if the block has no other blocks on top of it
    and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am
    unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am
    unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block
    being stacked.
I can only stack a block on top of another block if the block onto which I
    am stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.

[STATEMENT]
As initial conditions I have that, the red block is clear, the yellow
    block is clear, the hand is empty, the red block is on top of the blue
    block, the yellow block is on top of the orange block, the blue block
    is on the table and the orange block is on the table.
My goal is to have that the orange block is on top of the red block.

My plan is as follows:

[PLAN]
unstack the yellow block from on top of the orange block
put down the yellow block
pick up the orange block
stack the orange block on top of the red block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the orange block is clear, the yellow
    block is clear, the hand is empty, the blue block is on top of the red
    block, the orange block is on top of the blue block, the red block is
    on the table and the yellow block is on the table.
My goal is to have that the blue block is on top of the red block and the
    yellow block is on top of the orange block.

My plan is as follows:

```

```
[PLAN]
pick up the yellow block
stack the yellow block on top of the orange block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the red block is clear, the blue block
    is clear, the orange block is clear, the hand is empty, the blue block
    is on top of the yellow block, the red block is on the table, the
    orange block is on the table and the yellow block is on the table.
My goal is to have that the blue block is on top of the orange block and
    the yellow block is on top of the red block.

My plan is as follows:

[PLAN]
unstack the blue block from on top of the yellow block
stack the blue block on top of the orange block
pick up the yellow block
stack the yellow block on top of the red block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the red block is clear, the blue block
    is clear, the yellow block is clear, the hand is empty, the yellow
    block is on top of the orange block, the red block is on the table,
    the blue block is on the table and the orange block is on the table.
My goal is to have that the orange block is on top of the blue block and
    the yellow block is on top of the red block.

My plan is as follows:

[PLAN]
unstack the yellow block from on top of the orange block
stack the yellow block on top of the red block
pick up the orange block
stack the orange block on top of the blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that, <initial_state>
My goal is to have that <goals>.

My plan is as follows:

[PLAN]
```

For the next state prediction with the world model, we apply the prompts conditioned on the last action. Here we show the prompt to update the state after a "`pick up`" action as an example. Again, `<state>` and `<action>` would be instantiated with the current state and action.

```
I am playing with a set of blocks where I need to arrange the blocks into
    stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
```

```
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am unstacking is clear. Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being stacked.
I can only stack a block on top of another block if the block onto which I am stacking the block is clear. Once I put down or stack a block, my hand becomes empty.

After being given an initial state and an action, give the new state after performing the action.

[SCENARIO 1]
[STATE 0] I have that, the white block is clear, the cyan block is clear, the brown block is clear, the hand is empty, the white block is on top of the purple block, the purple block is on the table, the cyan block is on the table and the brown block is on the table.
[ACTION] Pick up the brown block.
[CHANGE] The hand was empty and is now holding the brown block, the brown block was on the table and is now in the hand, and the brown block is no longer clear.
[STATE 1] I have that, the white block is clear, the cyan block is clear, the brown block is in the hand, the hand is holding the brown block, the white block is on top of the purple block, the purple block is on the table and the cyan block is on the table.

[SCENARIO 2]
[STATE 0] I have that, the purple block is clear, the cyan block is clear, the white block is clear, the hand is empty, the white block is on top of the brown block, the purple block is on the table, the cyan block is on the table and the brown block is on the table.
[ACTION] Pick up the cyan block.
[CHANGE] The hand was empty and is now holding the cyan block, the cyan block was on the table and is now in the hand, and the cyan block is no longer clear.
[STATE 1] I have that, the cyan block is in the hand, the white block is clear, the purple block is clear, the hand is holding the cyan block, the white block is on top of the brown block, the purple block is on the table and the brown block is on the table.

[SCENARIO 3]
[STATE 0] <state>
[ACTION] <action>
[CHANGE]
```

D.2 Math Reasoning

We show the prompt of RAP for math reasoning as below. The prompt is used for both action proposal and next state prediction. After instantiate `<question>`, we append a prefix `Question 5.1` to the prompt, so that we can sample the first action with the LLM. The future actions are sampled similarly, except that all previous sub-questions and sub-answers need to be appended to the prompt, following the formats of in-context demonstration. The next state prediction, i.e., answering the sub-question, works in the same way.

```
Given a question, please decompose it into sub-questions. For each
    sub-question, please answer it in a complete sentence, ending with
    "The answer is". When the original question is answerable, please
    start the subquestion with "Now we can answer the question: ".

Question 1: Four years ago, Kody was only half as old as Mohamed. If
    Mohamed is currently twice as 30 years old, how old is Kody?
Question 1.1: How old is Mohamed?
Answer 1.1: He is currently 30 * 2 = 60 years old. The answer is 60.
Question 1.2: How old was Mohamed four years ago?
Answer 1.2: Four years ago, he must have been 60 - 4 = 56 years old. The
    answer is 56.
Question 1.3: How old was Kody four years ago?
Answer 1.3: Kody was half as old as Mohamed four years ago. Thus, Kody was
    56 / 2 = 28 years old. The answer is 28.
Question 1.4: Now we can answer the question: How old is Kody?
Answer 1.4: She is currently 28 + 4 = 32 years old. The answer is 32.

Question 2: On a moonless night, three fireflies danced in the evening
    breeze. They were joined by four less than a dozen more fireflies
    before two of the fireflies flew away. How many fireflies remained?
Question 2.1: How many fireflies joined?
Answer 2.1: The fireflies were joined by four less than a dozen more
    fireflies, which are 12 - 4 = 8 fireflies. The answer is 8.
Question 2.2: Now we can answer the question: How many fireflies remained?
Answer 2.2: Three fireflies were dancing originally. They were joined by 8
    fireflies before two of them flew away. So there were 3 + 8 - 2 = 9
    remaining. The answer is 9.

Question 3: Ali has four $10 bills and six $20 bills that he saved after
    working for Mr. James on his farm. Ali gives her sister half of the
    total money he has and uses 3/5 of the remaining amount of money to
    buy dinner. Calculate the amount of money he has after buying the
    dinner.
Question 3.1: How much money does Ali have in total?
Answer 3.1: Ali has four $10 bills and six $20 bills. So he has 4 * 10 + 6
    * 20 = 160 dollars. The answer is 160.
Question 3.2: How much money does Ali give to his sister?
Answer 3.2: Ali gives half of the total money he has to his sister. So he
    gives 160 / 2 = 80 dollars to his sister. The answer is 80.
Question 3.3: How much money does Ali have after giving his sister the
    money?
Answer 3.3: After giving his sister the money, Ali has 160 - 80 = 80
    dollars left. The answer is 80.
Question 3.4: How much money does Ali use to buy dinner?
Answer 3.4: Ali uses 3/5 of the remaining amount of money to buy dinner.
    So he uses 80 * 3/5 = 48 dollars to buy dinner. The answer is 48.
Question 3.5: Now we can answer the question: How much money does Ali have
    after buying the dinner?
Answer 3.5: After buying the dinner, Ali has 80 - 48 = 32 dollars left.
    The answer is 32.

Question 4: A car is driving through a tunnel with many turns. After a
    while, the car must travel through a ring that requires a total of 4
    right-hand turns. After the 1st turn, it travels 5 meters. After the
    2nd turn, it travels 8 meters. After the 3rd turn, it travels a little
    further and at the 4th turn, it immediately exits the tunnel. If the
```

```
    car has driven a total of 23 meters around the ring, how far did it
    have to travel after the 3rd turn?
Question 4.1: How far did the car travel except for the 3rd turn?
Answer 4.1: It travels 5 meters after the 1st, 8 meters after the 2nd, and
    0 meters after the 4th turn. It’s a total of 5 + 8 + 0 = 13 meters.
    The answer is 13.
Question 4.2: Now we can answer the question: How far did the car have to
    travel after the 3rd turn?
Answer 4.2: The car has driven a total of 23 meters around the ring. It
    travels 13 meters except for the 3rd turn. So it has to travel 23 - 13
    = 10 meters after the 3rd turn. The answer is 10.

Question 5: <question>
```

D.3 Logical Reasoning

We show the prompt for action proposal, action likelihood calculation, and next state prediction. `<fact>` and `<query>` would be instantiated with the problem.

```
Given a list of facts, and a current claim, output one possible fact as
    the next step. Be sure to copy the exact sentences in the facts. Do
    not change any wording. Do not create your own words.

Facts 1: Each lepidopteran is an insect. Each arthropod is a protostome.
    Every animal is multicellular. Protostomes are invertebrates. Each
    whale is bony. Each painted lady is a butterfly. Invertebrates are
    animals. Butterflies are lepidopterans. Each insect is six-legged.
    Every insect is an arthropod. Arthropods are not bony.
Query 1: True or false: Sally is not bony.
Claim 1.1: Sally is an insect.
Next 1.1: Each insect is six-legged.
Claim 1.2: Sally is a butterfly.
Next 1.2: Butterflies are lepidopterans.
Claim 1.3: Sally is a lepidopteran.
Next 1.3: Each lepidopteran is an insect.
Claim 1.4: Sally is not bony.
Next 1.4: Finish.
Claim 1.5: Sally is an arthropod.
Next 1.5: Arthropods are not bony.
Claim 1.6: Sally is a painted lady.
Next 1.6: Each painted lady is a butterfly.

Facts 2: Prime numbers are natural numbers. Every Mersenne prime is not
    composite. Imaginary numbers are not real. Every real number is a
    number. Natural numbers are integers. Every real number is real. Every
    Mersenne prime is a prime number. Natural numbers are positive. Prime
    numbers are not composite. Integers are real numbers.
Query 2: True or false: 127 is not real.
Claim 2.1: 127 is real.
Next 2.1: Finish.
Claim 2.1: 127 is a natural number.
Next 2.1: Natural numbers are integers.
Claim 2.2: 127 is a prime number.
Next 2.2: Prime numbers are natural numbers.
Claim 2.3: 127 is a real number.
Next 2.3: Every real number is real.
Claim 2.4: 127 is a Mersenne prime.
Next 2.4: Every Mersenne prime is a prime number.
Claim 2.5: 127 is an integer.
```

```
Next 2.5: Integers are real numbers.

Facts 3: Lepidopterans are insects. Every animal is multicellular. Each
    insect is an arthropod. Each invertebrate is an animal. Insects are
    six-legged. Arthropods are small. Arthropods are invertebrates. Each
    butterfly is a lepidopteran. Whales are not small.
Query 3: True or false: Polly is not small.
Claim 3.1: Polly is an arthropod.
Next 3.1: Arthropods are small.
Claim 3.2: Polly is an insect.
Next 3.2: Each insect is an arthropod.
Claim 3.3: Polly is small.
Next 3.3: Finish.
Claim 3.4: Polly is a lepidopteran.
Next 3.4: Lepidopterans are insects.

Facts 4: Every cat is a feline. Mammals are vertebrates. Bilaterians are
    animals. Vertebrates are chordates. Carnivores are mammals. Mammals
    are not cold-blooded. Each chordate is a bilaterian. Every feline is a
    carnivore. Snakes are cold-blooded. Animals are not unicellular. Every
    carnivore is not herbivorous.
Query 4: True or false: Fae is not cold-blooded.
Claim 4.1: Fae is a feline.
Next 4.1: Every feline is a carnivore.
Claim 4.2: Fae is not cold-blooded.
Next 4.2: Finish.
Claim 4.2: Fae is a mammal.
Next 4.2: Mammals are not cold-blooded.
Claim 4.3: Fae is a cat.
Next 4.3: Every cat is a feline.
Claim 4.4: Fae is a carnivore.
Next 4.4: Carnivores are mammals.

Facts 5: Prime numbers are prime. Real numbers are numbers. Every integer
    is a real number. Real numbers are not imaginary. Mersenne primes are
    prime numbers. Complex numbers are imaginary. Each prime number is a
    natural number. Natural numbers are positive. Each Mersenne prime is
    prime. Each natural number is an integer.
Query 5: True or false: 7 is imaginary.
Claim 5.1: 7 is not imaginary.
Next 5.1: Finish.
Claim 5.1: 7 is a natural number.
Next 5.1: Each natural number is an integer.
Claim 5.2: 7 is a prime number.
Next 5.2: Each prime number is a natural number.
Claim 5.3: 7 is a real number.
Next 5.3: Real numbers are not imaginary.
Claim 5.4: 7 is an integer.
Next 5.4: Every integer is a real number.

Facts 6: Spiders are not six-legged. Insects are six-legged. Insects are
    arthropods. Every animal is not unicellular. Invertebrates are
    animals. Lepidopterans are insects. Every arthropod is segmented.
    Arthropods are invertebrates. Every butterfly is a lepidopteran.
    Stella is a butterfly.
Query 6: True or false: Stella is six-legged.
Claim 6.1: Stella is an insect.
Next 6.1: Insects are six-legged.
Claim 6.2: Stella is a lepidopteran.
```

```
Next 6.2: Lepidopterans are insects.
Claim 6.3: Stella is a butterfly.
Next 6.3: Every butterfly is a lepidopteran.
Claim 6.4: Stella is six-legged.
Next 6.4: Finish.

Facts 7: <fact>
Query 7: <query>
```