Cost-Awareness in Tree-Search LLM Planning: A Systematic Study

Anonymous ACL submission

Abstract

Planning under resource constraints is central to real-world decision making, yet most large language model (LLM) planners assume uniform action costs. We systematically analyze whether tree-search LLM planners are cost-aware and whether they efficiently generate budget-feasible plans. In contrast to black-box prompting, explicit search trees expose intermediate decisions, node evaluations, and failure modes, which allows for controlled ablations of planner behavior. We study *depth-first search*, *breadth-first search*, *Monte Carlo Tree Search*, and *bidirectional search* within a unified framework. Our experiments show that existing tree-based LLM planners often struggle to find cost-optimal plans, and that additional search computation does not reliably improve optimality. Among the methods evaluated, bidirectional search achieves the best overall efficiency and success rate. MCTS achieves the highest optimality on short-horizon tasks. Tree-search planners are especially valuable for studying LLM planning because their reasoning steps are explicit, in contrast to plain LLMs that internalize planning dynamics through post-training trajectories. Our findings suggest that improving LLM planning under resource constraints will likely require new search algorithms, rather than solely scaling inference-time compute.

1 Introduction

Planning is a fundamental component of human intelligence, involving the generation of a sequence of actions to achieve a desired goal (Russell et al., 1995). Recently, large language models (LLMs) have emerged as promising planners due to their broad world knowledge, strong reasoning capabilities, and demonstrated success across a wide range of complex domains (Huang et al., 2024b; Wei et al., 2025), including task scheduling, e.g., travel and time management (Zheng et al., 2024; Oh et al., 2025), game playing (Wu et al., 2023; Chen et al., 2023; Huang et al., 2024a; Yang et al., 2025; Wang et al., 2023), tool use (Wang et al., 2025), and embodied decision-making (Ahn et al., 2022; Huang et al., 2022; Zhou et al., 2024b).

However, most existing LLM-based planners implicitly assume uniform action costs (Yao et al., 2022; Singh et al., 2023; Sun et al., 2023; Prasad et al., 2024), treating all actions as equally expensive regardless of the resources they consume. In reality, actions often incur heterogeneous costs in terms of energy, time, or monetary expense, by an executor such as a human or a robot. This simplifying assumption significantly limits the applicability of LLM planners in real-world domains, where resource constraints are often critical. In practical settings (e.g., fire-fighting robots with limited battery capacity), ignoring heterogeneous action costs can lead to plans that exceed budget constraints, resulting in infeasible execution or even catastrophic failures.

(RQ1). We investigate whether tree-search LLM planners such as Tree-of-Thoughts (ToT; Yao et al. (2023)) and MCTS (Hao et al., 2023) are inherently **cost-aware**, that is, whether they account for heterogeneous action costs and explicit budget constraints during plan generation. We consider a broad class of search strategies, including depth-first search (DFS), breadth-first search (BFS), Monte Carlo Tree Search (MCTS; Świechowski et al. (2022)), and bidirectional search (Sadhukhan, 2012; Ren et al., 2024), the last of which to the best of our knowledge has not been thoroughly explored in the LLM planning literature.

Compared to Input–Output (IO) and Chain-of-Thought (CoT) prompting (Wei et al., 2022), tree-based search methods generally achieve superior planning performance due to their ability to explore a larger solution space, leverage search policies to guide exploration, and incorporate self-correction through backtracking or rollouts. These properties increase the likelihood of finding feasible and potentially optimal plans under complex constraints. Moreover, unlike classical cost-sensitive search

algorithms such as A* search (Su et al., 2025), tree-search methods investigated in this study do not require task-specific reward or heuristic design, which simplifies their deployment and enables a more general and flexible planning framework across diverse domains.

(RQ2). We examine whether tree-search methods can **efficiently** generate cost-feasible plans under explicit budget constraints while requiring fewer tree-node expansions. This question is critical because tree-based methods are computationally expensive due to extensive node expansion and exploration, a challenge that is further amplified when LLMs are used to propose actions, evaluate states, and model state transitions.

We evaluate planner performance under three budget regimes with increasing flexibility: (a) TIGHT, where the budget equals the cost of the ground-truth optimal plan; (b) LOOSE, which allows a bounded cost overrun defined as the optimal cost plus a margin; solutions within this margin are considered acceptable even if they are not strictly optimal; and (c) UNLIMITED, which imposes no budget constraint and accepts all valid plans. For each LLM planner, we measure: **Success rate**, defined as the proportion of tasks for which the planner generates a valid, executable plan that satisfies the specified budget regime; **Optimality**, how close the cost of generated plans is to the minimum-cost optimal plan; and **Search efficiency**, measured by the number of tree-node expansions required to find a budget-feasible plan.

Our study yields three main observations. First, tree-based LLM planners often struggle to identify cost-optimal plans. Second, increasing search computation does not consistently lead to improved optimality. Third, we compare multiple tree-search algorithms and find substantial differences in efficiency: bidirectional search emerges as the most efficient approach, while also achieving higher success rates on long-horizon tasks. Collectively, these findings highlight two promising directions for advancing cost-aware LLM planning: (a) learning cost-aware reward models to better guide node expansion, and (b) incorporating principled infeasibility pruning based on cumulative cost and lower bounds on remaining cost to eliminate branches that cannot satisfy budget constraints or improve upon the current best plan.

2 Related Work

Tree-Search LLM Planners. Prior work on LLM-assisted planning has explored various tree-search paradigms, such as *depth-first search (DFS)*, *breadth-first search (BFS)*, and *Monte Carlo Tree Search (MCTS)*. BFS and DFS form the basis of Tree-of-Thoughts (ToT; Yao et al., 2023), which enables systematic exploration and self-correction through intermediate reasoning states, with subsequent analyses examining bounded variants to balance solution quality and computational cost (Katz et al., 2024). MCTS-based planning methods integrate LLMs as policy, value, or world models to balance exploration and exploitation via rollouts, achieving improved robustness and generalization on complex planning tasks (Świechowski et al., 2022; Hao et al., 2023; Zhou et al., 2024a; Zhang et al., 2025). Moreover, implicit search shifts the burden of exploration from inference time to training time. During post-training, the LLM learns to generate action sequences that yield high rewards, effectively internalizing a search policy over a latent space of trajectories (Zhou et al., 2024a; Parmar et al., 2025).

However, existing approaches primarily optimize for task success or reasoning quality and largely ignore heterogeneous action costs and explicit budget constraints. In contrast, our work systematically studies the cost awareness and search efficiency of LLM-based tree-search planners, evaluating their ability to generate cost-feasible plans under budget constraints while minimizing tree-node expansions.

Cost-Aware Planning. Planning is a core component of human intelligence, centered on generating action sequences to explore solutions and support decision-making (Hayes-Roth and Hayes-Roth, 1979; Mattar and Lengyel, 2022; Su, 2023). Much of the prior work emphasizes planning completeness, i.e., whether a planner can find a feasible plan when one exists (Puig et al., 2018; Shridhar et al., 2020; Valmeekam et al., 2023; Li et al., 2025; Parashar et al., 2025). More recent studies incorporate action costs, making benchmarks more realistic by requiring solutions to balance feasibility with optimality (Wu et al., 2025; Liu et al., 2025; Qian et al., 2025). TravelPlanner is a commonly used cost-aware benchmark, but it primarily evaluates executability: the objective is to produce a plan that satisfies all constraints rather than to minimize cost (Xie et al., 2024). In contrast, many other cost-aware benchmarks are difficult to scale in difficulty or to evaluate reliably, especially with respect to intermediate states and plan correctness. For these reasons, we use BlocksWorld as our testbed (Kambhampati et al., 2024).

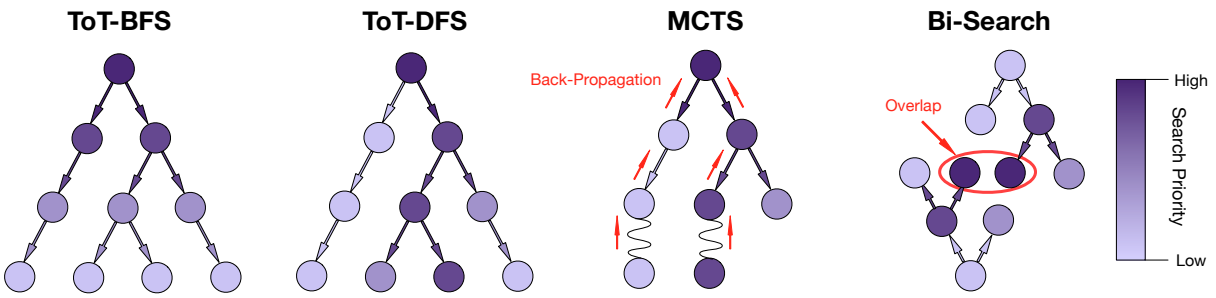

Figure 1: A visual comparison of four tree-based LLM planners included in this study, where node color intensity indicates search priority. **ToT-BFS** combines an LLM with breadth-first search, exhaustively exploring all nodes at each depth. **ToT-DFS** uses depth-first search, prioritizing deeper expansions during the search process. **MCTS** implements the Monte Carlo Tree Search (MCTS), combining exploration and previous roll-out by Upper Confidence Bounds for Trees (UCT equation 1) to dynamically guide the search process. **Bidirectional Search (Bi-Search)** incorporates two search trees, one from the initial state, one from the goal state, and alternates expansions to find a connecting state, thereby reducing effective search depth and pruning the search space.

3 Methods

We study cost-constrained planning problems where a planner is given an initial state s_0, a goal state s_g, a budget limit $\mathcal{B}$, and an action set $\mathcal{A}$. Executing an action $a \in \mathcal{A}$ in a state induces a transition to a successor state and incurs an action-dependent cost $c(a)$. The planner's objective is to select an action sequence $\mathcal{A}' = [a_1, a_2, \ldots, a_k]$ such that applying these actions from s_0 reaches the goal state s_g. In addition to goal reachability, the plan must satisfy the budget constraint, i.e., the total cost $\sum_{i=1}^{k} c(a_i)$ does not exceed $\mathcal{B}$.

3.1 Tree-based LLM planner

We study representative tree-search planners that integrate an LLM into the search loop, differing mainly in (1) how search policy is determined and (2) tree search structure, which determines whether the planner expands a single forward tree rooted at the initial state or two coupled trees rooted at the initial and goal states that terminate when the frontiers meet (Figure 1). Across all methods, a search node represents a partial plan prefix $A_{partial} = [a_1, \ldots, a_t]$, with $t \in [1, K]$ where K is the maximum allowed number of actions per trajectory, and executing this prefix from s_0 yields the intermediate state s_t. Each node maintains the proposed action, state, the accumulated execution cost, and a scalar score used to prioritize further expansion (defined in §3.2). A node becomes terminal when it reaches the goal state or exceeds the maximum number of allowed actions per trajectory

Tree-of-Thought (BFS/DFS). ToT explores the tree in a BFS/DFS-style manner by prompting the LLM to propose multiple candidate next actions from the current state s_t. At each expansion, we sample m actions $\{a_{t+1}^{(j)}\}_{j=1}^{m}$, execute them to obtain successor states $\{s_{t+1}^{(j)}\}$, score each child with reward $r_{t+1}^{(j)}$, and select which nodes to continue expanding using either depth-first (DFS) or breadth-first (BFS) traversal

Monte Carlo Tree Search (MCTS). MCTS balances exploration and exploitation via iterative *selection, expansion, simulation, backpropagation*. In the selection stage, starting from the root, we choose a leaf node from the current search tree that maximizes a UCT-style criterion:

$$\mathrm{UCT}(u) = Q(u) + \beta\sqrt{\frac{\ln N(\mathrm{parent}(u))}{1 + N(u)}}, \quad (1)$$

where $N(u)$ is the visit count and $Q(u)$ is the running estimate of node value. In the expansion stage, we query the LLM for m candidate actions at the selected leaf node and add feasible children. In the simulation stage, we compute each new child's reward r (§3.2) and treat it as a rollout/value estimate. We keep running the roll-out greedily till the terminal state. In the backpropagation stage, we propagate the value upward, e.g., by updating Q with running averages from terminal node to root node. Compared to ToT, MCTS tends to focus computation on a smaller set of promising branches while still occasionally exploring under-visited alternatives based on previous rollout results. MCTS outperforms BFS and DFS by dynamically balancing exploration and exploitation, focusing computation on high-value outcomes rather than exhaustively or rigidly traversing the search tree.

Bidirectional Search. Bidirectional search (Bi-Search) grows two trees simultaneously: a forward tree rooted at s_0 and a backward tree rooted at s_g. A forward node represents a partial plan prefix from s_0 to some s_t, while a backward node represents a partial *reverse* plan from s_g to an intermediate state (when reverse transitions are available) or, more generally, a goal-conditioned subgoal state proposed by the LLM. The algorithm alternates expanding the frontier that appears more promising (according to reward and/or remaining budget), and declares success when the two frontiers meet at a compatible intermediate state. The detailed pseudocode is included in Appendix C. Compared to unidirectional tree search, bidirectional search significantly shrinks the search space by meeting in the middle. By replacing one deep expansion with two shallower ones, it avoids the rapid combinatorial blow-up that dominates long-horizon unidirectional search.

3.2 Reward Design

To guide general search-tree expansion, we assign each non-root node a reward score that reflects how promising it is for reaching a valid solution. In our setting, we adopt the reward design from RAP and apply it uniformly across all tree-search algorithms evaluated in our experiments (Hao et al., 2023). Based on RAP's implementation, the reward model consist of two component, action evaluation and self evaluation. We attached the detailed prompt in Appendix E. Concretely, we compute each node's reward by combining two LLM-derived confidence signals—one assessing the selected action relative to available choices, and another self-checking the selected action under the budget constraint. We next describe how each component is computed and how its confidence signals are used to score nodes during search-tree expansion

Confidence Reward. *Action evaluation.* Consider a leaf node v with parent node v_p. Let a be the action that transforms v_p into v, and let c_a be its cost. The parent node has an accumulated cost c_{v_p}. To assess the quality of a, we prompt the LLM with the parent state v_p, the available action set $\mathcal{A}$, the budget limit $\mathcal{B}$, and the accumulated cost c_{v_p}. The LLM is asked to predict the best next action. Since a was the action taken, we use the log-probability that the LLM assigns to a as the confidence reward for node v. *Self-evaluation.* Given the action a, the budget limit $\mathcal{B}$, the parent node v_p, and its cost c_{v_p}, we prompt the LLM to assess whether a is a good choice by outputting either *good* or *bad*. We then use the log-probability of generating *good* as an additional confidence signal for node v. The final reward for node v is obtained by summing these two log-probability signals (action-evaluation and self-evaluation).

3.3 Metrics

We evaluate the performance using three metrics: **Success Rate**, **Optimality**, **Efficiency**. The **success rate** checks whether the planner finds a valid, cost-adherent plan within the node expansion limits. **Optimality** evaluates execution cost quality by comparing the generated plan's cost to that of the ground-truth optimal plan. We use a ratio to make optimality comparable across instances with different cost scales and to reflect proportional sub-optimality. **Efficiency** measures how much search compute a method expends before finding the first budget-feasible plan, operationalized as the unused fraction of a fixed node-expansion limit. A higher efficiency value means the planner reached a solution while consuming a smaller portion of its allotted node expansion. We include the detailed equation in the Appendix B.

4 Experiments

4.1 Datasets

We introduce our testbed **Budget-BlocksWorld**, a cost-augmented version of BlocksWorld benchmark (Valmeekam et al., 2022). BlocksWorld is a widely used planning benchmark with several distinct blocks on a table. The objective of this task is to rearrange the blocks on the table from its initial state to the goal state using four deterministic actions: pick-up, put-down, stack, and unstack. We adopt BlocksWorld with action costs to transform it into Budget-BlocksWorld. We assign non-uniform costs to actions, making some actions (e.g., put-down/pick-up) more expensive than others (e.g., stack/unstack), which forces planners to trade off plan length against execution cost. Building on top of this setup, Budget-BlocksWorld obtains 1,008 tasks with 6-block instances, and categorizes the difficulty by the optimal plan length (minimum number of actions from initial state to goal), where longer horizons require reasoning over more complex state transitions and larger search space.

Ground Truth Generation. To establish the cost distribution for our experiments, we evaluate several cost schedules and recompute ground-truth plans to ensure cost optimality. Using exhaustive search, we enumerate all possible plans and select the lowest-cost plan as the new ground truth.

		(Easy Tasks) ← **Success Rate** → (Hard Tasks)							(Easy Tasks) ← **Optimality** → (Hard Tasks)						
	Plan Length (L)	L=2	4	6	8	10	12	**SR**	L=2	4	6	8	10	12	**Opt**
	Task Count	15	58	99	138	154	183	**Avg**	15	58	99	138	154	183	**Avg**
TIGHT	CoT w/ Qwen3	0.6	0.08	0.00	0.00	0.00	0.00	**0.11**	1.00	1.00	–	–	–	–	**0.33**
	CoT w/ GPT4.1	0.4	0.08	0.00	0.00	0.00	0.00	**0.08**	1.00	1.00	–	–	–	–	**0.33**
	CoT w/ Claude	0.8	0.75	0.26	0.26	0.07	0.01	**0.36**	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**
	ToT-BFS	0.80	0.19	0.00	0.00	0.00	0.00	**0.17**	1.00	1.00	–	–	–	–	**0.33**
	ToT-DFS	0.86	0.33	0.04	0.00	0.00	0.00	**0.21**	1.00	1.00	1.00	–	–	–	**0.50**
	MCTS	1.00	0.92	0.50	0.11	0.05	0.00	**0.43**	1.00	1.00	1.00	1.00	1.00	–	**0.83**
	Bi-Search	1.00	0.92	0.48	0.12	0.11	0.07	**0.45**	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**
LOOSE	CoT w/ Qwen3	0.6	0.16	0.00	0.00	0.00	0.00	**0.13**	1.00	0.67	–	–	–	–	**0.28**
	CoT w/ GPT4.1	0.6	0.33	0.23	0.15	0.17	0.15	**0.27**	0.39	0.75	0.31	0.36	0.57	0.54	**0.49**
	CoT w/ Claude	0.6	0.91	0.61	0.48	0.35	0.24	**0.53**	0.52	0.92	0.54	0.59	0.56	0.58	**0.62**
	ToT-BFS	1.00	0.26	0.00	0.00	0.00	0.00	**0.21**	0.90	0.81	–	–	–	–	**0.29**
	ToT-DFS	1.00	0.58	0.09	0.01	0.00	0.00	**0.28**	0.92	0.88	0.38	0.29	–	–	**0.41**
	MCTS	1.00	1.00	0.85	0.39	0.11	0.01	**0.56**	1.00	0.93	0.58	0.49	0.51	0.48	**0.67**
	Bi-Search	1.00	0.92	0.93	0.88	0.79	0.71	**0.87**	1.00	0.84	0.58	0.43	0.41	0.36	**0.60**
UNLIMITED	CoT w/ Qwen3	0.6	0.16	0.00	0.00	0.00	0.00	**0.13**	1.00	0.67	–	–	–	–	**0.28**
	CoT w/ GPT4.1	0.8	0.41	0.29	0.18	0.23	0.19	**0.35**	0.30	0.85	0.42	0.40	0.45	0.49	**0.48**
	CoT w/ Claude	0.6	0.83	0.58	0.47	0.42	0.32	**0.54**	0.70	0.90	0.49	0.52	0.63	0.44	**0.61**
	ToT-BFS	1.00	0.38	0.01	0.00	0.00	0.00	**0.23**	0.90	0.81	0.52	–	–	–	**0.37**
	ToT-DFS	1.00	0.67	0.12	0.00	0.00	0.00	**0.30**	0.90	0.78	0.36	–	–	–	**0.34**
	MCTS	1.00	1.00	0.86	0.55	0.20	0.05	**0.61**	1.00	0.93	0.58	0.44	0.52	0.42	**0.65**
	Bi-Search	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	1.00	0.84	0.60	0.43	0.40	0.32	**0.60**

Table 1: **Success rate and optimality by plan lengths under budget constraints.** Columns correspond to task subsets grouped by ground-truth plan length L (shown with task counts n in the header; shorter plans are easier). Left block reports **Success Rate** (fraction of tasks solved; ↑ better). Right block reports **Optimality** computed on *successful* tasks only (e.g., Cost(Optimal plan) / Cost(Generated plan); ↑ better); entries are '–' when no instance is solved for that length. Under TIGHT budgets, any plan that is considered valid must match the optimal (minimum) cost, so optimality is trivially 1.00 whenever a method succeeds within the node limit.

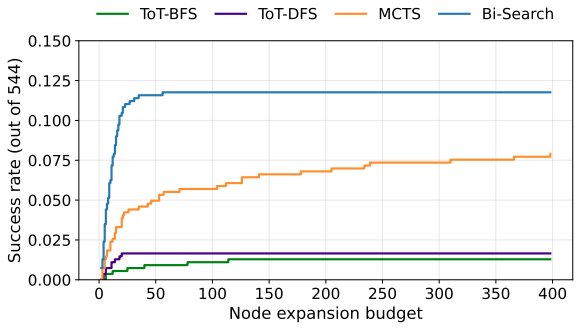

Figure 2: Node expansion vs Success rate. This plot demonstrates the number of node usage to find the optimal plan under **TIGHT** constraint.

Solution Shift Rate. To ensure that our optimal solutions are clearly distinguishable from those of the vanilla BlocksWorld, we randomly select the task from the top-3 configurations with the highest ratio of shifted plans as our testbed.

A task's solution is considered as shifted if the cost-optimal ground truth plan is not identical to the uniform ground truth plan. We compute the solution shift rate for each cost schedule. This metric measures the proportion of tasks in Budget-BlocksWorld whose cost-optimal plans differ from the uniform-cost BlocksWorld baseline. We provide more statistics on our cost-augmented BlocksWorld in Appendix A.

Budget regimes. Budget-BlocksWorld has three budget conditions to simulate different levels of cost strictness. **TIGHT** budget is set exactly to the ground-truth cost-optimal plan, permitting only plans whose cost matches the optimum. **LOOSE** budget is set to the optimal cost plus a small margin. More specifically, we set the margin to $2c_{max}$, where c_{max} denotes the maximum cost among all actions. In BlocksWorld, plans typically consist of paired actions (e.g., pick-up/unstack followed by put-down/stack). Thus, setting the margin to $2c_{max}$ allows additional one high-cost action pair in the generated plan. **UNLIMITED** condition applies no budget constraints, and all valid plans are accepted. This setup allows us to systematically study planner's behavior under strict, relaxed, and unconstrained cost settings.

4.2 Implementation details

We compare a set of representative LLM planning approaches spanning (1) direct (raw) LLM planners with CoT that generate plans without explicit search (e.g., GPT-4.1 and Claude-Opus-4.1) and (2) tree-search-based planners, Tree-of-Thought (ToT-BFS/DFS), MCTS, and bidirectional search

with Qwen3-8B as the backbone model. We adopt Qwen3-8B for tree-based planners for two reasons. First, tree search requires a large number of model calls, making closed-source models prohibitively expensive under our compute budget. Second, our reward computation relies on token-level scoring.

LLM planner w/ CoT. We directly prompt the model with in-context learning examples and require it to generate the entire plan in one pass. The cost budget limit specified in the prompt is varied according to the given constraint. The full prompting details are included in the Appendix E.

Tree-based LLM planners. We evaluate tree-search prompting approaches describe in Section 3.1. For consistency, all the tree-based LLM planners use the same LLM-based confidence scoring signal described in Section 3.2 to rank candidate continuations. For each state, the LLM picks the top-5 actions from the action pool. To reduce hallucinated state transitions and ensure a shared notion of validity across methods, we use the PDDLGym (Silver et al., 2016) as the transit function to execute all proposed actions that deterministically update states and reject invalid actions.

Feasibility pruning and search computation limits. To assess whether cost-aware behavior arises from the model and search procedure *without* relying on an explicit rule, we consider an optional *hard budget feasibility filter* for ToT-BFS/DFS, RAP, and Bidirectional Search: any action (or partial plan) whose accumulated cost exceeds the budget is pruned immediately. We impose a common search budget of at most 500 node expansions for ToT-BFS/DFS, RAP, and Bidirectional Search to ensure comparable compute across methods and to prevent degenerate exhaustive search.

5 Experimental Results

5.1 Overall performance

Table 1 reports Success Rate and Optimality across ground-truth plan length $L = \{2, 4, 6, 8, 10, 12\}$, under TIGHT, LOOSE, and UNLIMITED budget constraints, from which we draw the following insights:

Tree-based LLM planners struggle to find cost-optimal plans for long-horizon tasks. To illustrate these trends, we focus on the LOOSE and UNLIMITED budget settings, as optimality under the TIGHT constraint is uninformative: any successful plan must match the minimum cost, causing an optimality of 1.00 for all nonempty entries. Under both settings, Table 1 shows that tree-based methods (particularly MCTS and bidirectional search)

(Easy Tasks) ← **Efficiency** → (Hard Tasks)

	Plan Length / **Task Count**	L=2 / 15	4 / 58	6 / 99	8 / 138	10 / 154	12 / 183	**Eff Avg**
TIGHT	ToT-BFS	0.97	0.71	–	–	–	–	**0.28**
	ToT-DFS	0.99	0.96	0.86	–	–	–	**0.47**
	MCTS	0.99	0.98	0.95	0.11	0.05	–	**0.51**
	Bi-Search	0.99	0.98	0.97	0.96	0.93	0.91	**0.96**
LOOSE	ToT-BFS	0.96	0.54	0.37	–	–	–	**0.31**
	ToT-DFS	0.98	0.94	0.89	0.45	–	–	**0.54**
	MCTS	0.99	0.96	0.81	0.61	0.43	0.23	**0.67**
	Bi-Search	0.99	0.99	0.98	0.97	0.93	0.91	**0.96**
UNLIMITED	ToT-BFS	0.95	0.78	0.35	–	–	–	**0.35**
	ToT-DFS	0.98	0.92	0.83	–	–	–	**0.46**
	MCTS	0.99	0.89	0.79	0.62	0.51	0.45	**0.71**
	Bi-Search	0.99	0.98	0.96	0.94	0.91	0.87	**0.94**

Table 2: **Efficiency by plan lengths under budget constraints.** Efficiency is calculated by the ratio of unused node and node expansion limit (↑ better; please refer to Eq. (3)). Same as Optimality, Efficiency is only computed on successful tasks. '-' means the algorithm does not find a feasible plan in the node expansion limit.

achieve high optimality for shorter plan lengths ($L = \{2, 4\}$), where the planning problem is relatively easy. In these cases, MCTS outperforms closed-source LLMs using CoT prompting. However, as problem difficulty increases with longer horizons ($L = \{8, 10, 12\}$), the optimality of tree-based methods declines sharply and no longer surpasses strong closed-source baselines without tree search (e.g., CoT with Claude).

Overall, these results suggest that: although tree search, especially bidirectional search, substantially improves *feasibility* (i.e., success rate) over CoT-based methods using both open- and closed-source LLMs (shown in the left part of Table 1), node selection in existing tree-search planners is not consistently cost-aware, often producing cost-inefficient solutions even when valid plans are found in more challenging settings.

More search compute does not necessarily yield optimal plan. Figure 2 shows optimality as a function of node expansions under the **TIGHT** budget constraint, where any successful plan is *cost-optimal*. ToT-BFS/DFS and bidirectional search achieve most of their feasible solutions within the early stages of search, after which additional node expansions provide little benefit. MCTS continues to improve with increased search budget but quickly exhibits diminishing returns. This difference stems from MCTS, which updates node values based on rollout outcomes and provides more cost-aware guidance, whereas other tree-based planners rely on largely static search policies and can prematurely commit to suboptimal trajectories. Overall, these results show that additional computation

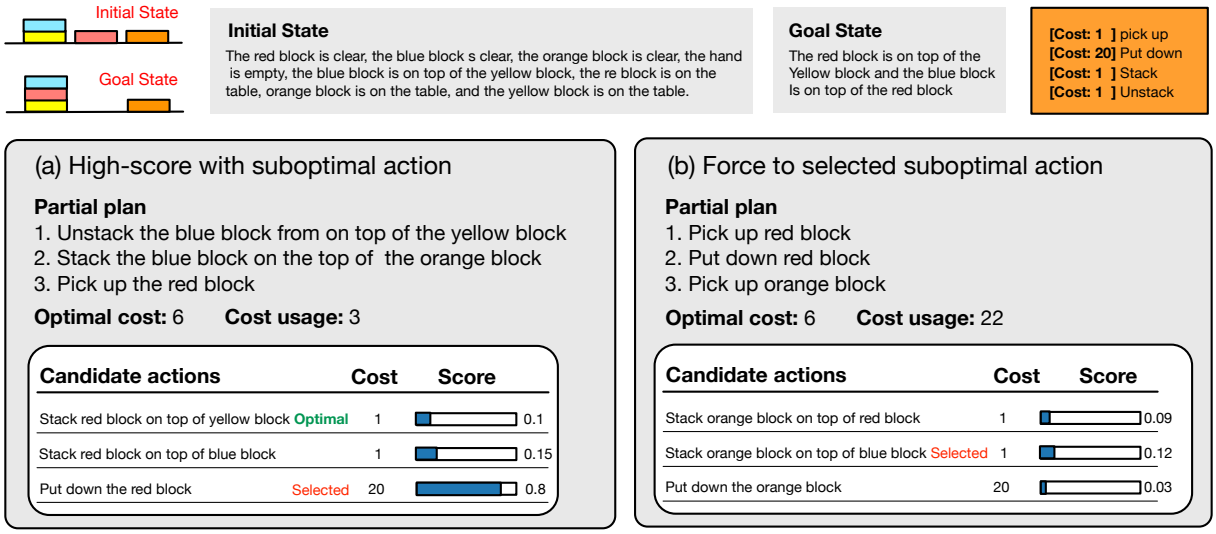

Figure 3: Decision-level diagnostics of cost-aware planning failures. We show the initial and goal states (top) and the action-cost schedule. Each panel zooms into a search step, listing the partial plan, cost usage versus the optimal cost, and the candidate actions with their model scores. (a) **Assign high score to suboptimal action**: the model assigns a high score to a costly suboptimal action (selected), diverting the search from the optimal continuation. (b) **Force to selected suboptimal action:** all candidates receive low, similar scores, forcing selection among suboptimal options and keep searching on the high cost branch.

alone does not ensure cost awareness, underscoring the need for explicit cost-aware guidance in LLM-based tree-search planning.

Tree-search algorithm directly affects the efficiency. Table 2 shows large efficiency gaps across different tree-search strategies: bidirectional search can substantially improve the efficiency of finding feasible plans, even when the problem becomes more challenging. Moreover, Table 1 demonstrates that compared with single-tree search methods, bidirectional search also achieves higher success rates on longer-horizon tasks. In contrast, *ToT-BFS* tends to exhaustively expand nodes level by level; this layer-wise exploration quickly inflates the number of expansions, leading to poorer efficiency and a sharper drop in success as the horizon grows.

5.2 Cost-Aware Failure Analysis

In this section, we investigate failure cases in cost-aware planning to reveal the key limitations and challenges faced by tree-based LLM planners.

5.2.1 Failure Modes Distribution

We categorize unsuccessful runs into two failure modes aligned with our research questions mentioned in Section 1. (1) **Budget Violation** denotes goal-reaching plans that exceed the cost budget, indicating limited cost awareness (RQ1). (2) **Search Exhaustion** denotes failures to find any valid plan within the node-expansion limit, reflecting poor search efficiency (RQ2).

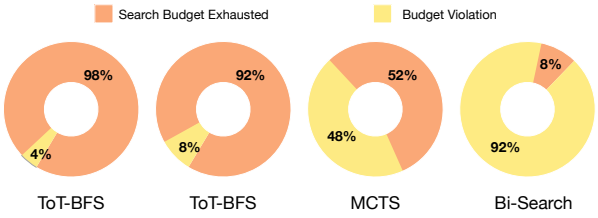

Figure 4: Failure mode distribution of TIGHT condition from $L = 2$ to $L = 8$

Overall Failure Distribution. We conduct failure analysis under the TIGHT budget setting, as the LOOSE and UNLIMITED settings permit suboptimal plans and therefore do not meaningfully capture budget violations. Because ToT rarely finds feasible solutions for longer horizons (see Table 1), we restrict our analysis to tasks with plan lengths $L = \{2, 4, 6, 8\}$.

Figure 4 presents the distribution of failure modes under the TIGHT budget across these plan lengths. We observe substantial differences across search methods: *ToT-BFS* fails almost entirely due to Search Exhaustion (98%), reflecting that its layer-wise breadth-first strategy cannot discover cost-optimal solutions within the node budget. *ToT-DFS* exhibits a similar pattern (92% Search Exhaustion, 8% Budget Violation), with its depth-first commitment occasionally producing complete but cost-inefficient plans. *MCTS* shows a more balanced failure distribution (52% Search Exhaustion,

	ToT-BFS	ToT-DFS	MCTS	Bi-Search
TIGHT	88.1%	89.3%	58.0%	77.3%
LOOSE	83.5%	85.7%	36.4%	69.8%

Table 3: Failure rate of identifying impossible trajectories (↓ better). We evaluate 1000 nodes from $L = \{2, 4, 6, 8\}$ of the failure tasks. We describe the detailed definition of failure rate in §5.2.2.

48% Budget Violation); while it expands the search coverage compared to ToT methods, roughly half of its discovered solutions exceed the tight budget, revealing limited cost awareness. Finally, *Bi-Search* experiences predominantly Budget Violations (92%), suggesting that while bidirectional search efficiently finds feasible plans, it often fails to identify truly cost-optimal solutions.

5.2.2 Root Cause Analysis

Search Trajectory Analysis and Failure Rate. To further investigate why tree-based LLM planners violate budget constraints, we analyze their search traces through the lens of cost-optimal feasibility. For a node s_v along a generated plan prefix, let $g(s_v)$ denote the accumulated cost from the initial state, and let $h(s_v)$ denote the *optimal* remaining cost from s_v to the goal, when such a path exists. Let C be the *ground-truth* optimal total cost. We say that the LLM fails to identify an infeasible trajectory at s_v if $g(s_v) + h(s_v) > C$, meaning that even an optimal continuation from s_v cannot satisfy the budget constraint.

Based on this criterion, we define the **failure rate** as the proportion of nodes at which the LLM fails to recognize such infeasible trajectories. We randomly sampled 1000 nodes from the failure tasks. We report failure rates for different tree-based methods under the TIGHT and LOOSE budget constraints in Table 3, since the UNLIMITED setting accepts all feasible plans regardless of cost. From Table 3, we derive the following insights, which highlight how different search algorithms perform under cost-aware planning:

LLM Fails to identify impossible trajectory. Table 3 shows that, even when explicitly prompted with action costs and cumulative cost, the LLM frequently fails to recognize prefixes that are already doomed under the budget, leading to budget-violating continuations in 36.4–89.3% of node expansions. We observe two key insights from these results: (1) MCTS exhibits a substantially lower failure rate than the other search algorithms, suggesting that incorporating feedback from exploration can partially correct for miscalibrated confidence scores; and (2) holding the reward fixed, bidirectional search consistently outperforms ToT-BFS/DFS, implying that search structure and frontier management can reduce exposure to infeasible path even without changing the underlying reward signal. We discuss each finding in turn below.

MCTS demonstrates notable advantages over heuristic-only approaches for cost-aware planning. MCTS differs from the other search algorithms in how it selects nodes to expand: ToT-BFS/DFS and Bi-Search rely primarily on the LLM's confidence score, whereas MCTS also uses feedback accumulated from prior exploration. As a result, MCTS achieves lower in failure rates than other method in both budget settings (Table 3). This stems from MCTS's dual guidance mechanism: in addition to heuristic scoring, it leverages value backpropagation to incorporate empirical evidence from rollouts, partially correcting for LLM scoring errors by learning which branches actually lead to budget-feasible solutions.

Tree-search algorithms matter under same reward signal. Even using the same reward, failure rates are varied across BFS, DFS, and Bi-Search. In the DFS and BFS, when an LLM assigns uniformly low confidence to the available actions, the algorithm will still commit to a local choice and keep expanding an unpromising branch, shown on Figure 3(b). While bidirectional search always selects the most promising branch on the search tree for both directions, this design can avoid getting trapped in the infeasible branches during the search process. Overall, these results show that the search strategy itself, not just the reward, critically determines whether the planner can avoid wasting computational resources on infeasible paths and improve the efficiency.

6 Conclusion

We analyze cost-aware, tree-based LLM planners under explicit budget constraints. Tree search substantially improves feasibility; however, increasing search compute does not reliably yield better cost-quality trade-offs (answer to RQ1). An analysis of search trajectories shows that the choice of search algorithm critically affects how efficiently a budget-feasible plan is found (answer to RQ2). Looking forward, we identify two promising directions: (i) learning cost-aware reward models to effectively guide node expansion, and (ii) introducing principled pruning via cost-aware lower bounds to eliminate branches unlikely to satisfy budget constraints or improve upon the current best plan.

7 Limitations

Our study intentionally focuses on tree-search planners, aiming to isolate how search structure alone affects budgeted planning; accordingly, we do not incorporate additional inference-time heuristics or self-refinement procedures that could confound this comparison. In addition, our results are conditioned on a specific prompting and LLM-based reward design. Finally, our experiments use a fixed set of LLM(s), and performance may vary in magnitude across other models even when the qualitative trends remain similar.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, and 1 others. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Jiangjie Chen, Siyu Yuan, Rong Ye, Bodhisattwa Prasad Majumder, and Kyle Richardson. 2023. Put your money where your mouth is: Evaluating strategic planning and execution of llm agents in an auction arena. *arXiv preprint arXiv:2310.05746*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Barbara Hayes-Roth and Frederick Hayes-Roth. 1979. A cognitive model of planning. *Cognitive science*, 3(4):275–310.

Jen-tse Huang, Eric John Li, Man Ho Lam, Tian Liang, Wenxuan Wang, Youliang Yuan, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Michael R Lyu. 2024a. How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments. *arXiv preprint arXiv:2403.11807*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, and 1 others. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

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

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't plan, but can help planning in llm-modulo frameworks. *Preprint*, arXiv:2402.01817.

Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi Araghi. 2024. Thought of search: Planning with language models through the lens of efficiency. *Advances in Neural Information Processing Systems*, 37:138491–138568.

Haoming Li, Zhaoliang Chen, Jonathan Zhang, and Fei Liu. 2025. Planet: A collection of benchmarks for evaluating llms' planning capabilities. *Preprint*, arXiv:2504.14773.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R Fung. 2025. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents. *arXiv preprint arXiv:2511.02734*.

Marcelo G Mattar and Máté Lengyel. 2022. Planning in the brain. *Neuron*, 110(6):914–934.

Juhyun Oh, Eunsu Kim, and Alice Oh. 2025. Flextravelplanner: A benchmark for flexible planning with language agents. *Preprint*, arXiv:2506.04649.

Shubham Parashar, Blake Olson, Sambhav Khurana, Eric Li, Hongyi Ling, James Caverlee, and Shuiwang Ji. 2025. Inference-time computations for llm reasoning and planning: A benchmark and insights. *Preprint*, arXiv:2502.12521.

Mihir Parmar, Palash Goyal, Xin Liu, Yiwen Song, Mingyang Ling, Chitta Baral, Hamid Palangi, and Tomas Pfister. 2025. Plan-tuning: Post-training language models to learn step-by-step planning for complex problem solving. *Preprint*, arXiv:2507.07495.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4226–4252.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502.

Cheng Qian, Zuxin Liu, Shirley Kokane, Akshara Prabhakar, Jielin Qiu, Haolin Chen, Zhiwei Liu, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025. xrouter: Training cost-aware llms orchestration system via reinforcement learning. *Preprint*, arXiv:2510.08439.

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

Stuart Russell, Peter Norvig, and Artificial Intelligence. 1995. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25(27):79–80.

Samir K Sadhukhan. 2012. A new approach to bidirectional heuristic search using error functions. In *Proc. 1st International Conference on Intelligent Infrastructure at the 47th Annual National Convention COMPUTER SOCIETY of INDIA (CSI-2012)*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and 1 others. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: program generation for situated robot task planning using large language models. *Autonomous Robots*, 47(8):999–1012.

DiJia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qinqing Zheng. 2025. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. *Preprint*, arXiv:2410.09918.

Yu Su. 2023. Language agents: a critical evolutionary step of artificial intelligence. yusu. substack. com.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36:38975–38987.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025. Acting less is reasoning more! teaching model to act efficiently. *Preprint*, arXiv:2504.14870.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. Plangenllms: A modern survey of llm planning capabilities. *Preprint*, arXiv:2502.11221.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*.

Duo Wu, Jinghe Wang, Yuan Meng, Yanning Zhang, Le Sun, and Zhi Wang. 2025. Catp-llm: Empowering large language models for cost-aware tool planning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8699–8709.

Yue Wu, Xuan Tang, Tom M Mitchell, and Yuanzhi Li. 2023. Smartplay: A benchmark for llms as intelligent agents. *arXiv preprint arXiv:2310.01557*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Ruihan Yang, Jiangjie Chen, Yikai Zhang, Siyu Yuan, Aili Chen, Kyle Richardson, Yanghua Xiao, and Deqing Yang. 2025. Selfgoal: Your language agents already know how to achieve high-level goals. In *Proceedings of the NAACL-HLT*, pages 799–819.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Yifan Zhang, Giridhar Ganapavarapu, Srideepika Jayaraman, Bhavna Agrawal, Dhaval Patel, and Achille Fokoue. 2025. Spiral: Symbolic llm planning via grounded and reflective search. *Preprint*, arXiv:2512.23167.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, and 1 others. 2024. Natural plan: Benchmarking llms on natural language planning. *arXiv preprint arXiv:2406.04520*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024a. Language agent tree search unifies reasoning acting and planning in language models. *Preprint*, arXiv:2310.04406.

Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. 2024b. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2022. Monte carlo tree search: a review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562.

A Budget-Blocksworld

For each task in the BlocksWorld, we recompute the cost-optimal solution via exhaustive search. At each step, we expand all feasible actions and retain the minimum-cost path upon reaching a goal state. This non-trivial process ensures that our benchmark is grounded in reliable optimal solutions. Details are reported in Table 4.

B Evaluation Metrics

This appendix formally defines the metrics used in our experiments. **Optimality** measures solution quality by comparing the execution cost of a generated plan to the ground-truth optimal cost, with higher values indicating closer-to-optimal solutions. **Efficiency** measures computational cost in terms of node expansions, normalized by the fixed expansion budget; higher values indicate that a method finds a solution with fewer expansions.

$$\text{Optimality} = \frac{\text{Cost}_{opt}}{\text{Cost}_{gen}} \tag{2}$$

$$\text{Efficiency} = 1 - \frac{\#\,\text{Nodes Expanded}}{\text{Node Budget}} \tag{3}$$

C Bidirectional Search Algorithm

Bidirectional search in our setting grows two complementary search trees to connect the initial and goal states more efficiently than a single forward tree. Bi-Search constructs a forward tree T_f rooted at the initial state s_0 and a backward tree T_b rooted at the goal state s_g, and expands them in alternating turns under a fixed node-expansion limit L. Each expansion selects a promising leaf node in the current tree, enumerates all feasible actions from that node (respecting the budget constraint), and adds the resulting successor states as children while tracking the cumulative path cost from the corresponding root. After each round, the algorithm checks whether the two trees contain an overlapping state; when an overlap is found, it terminates and extracts a complete plan by concatenating the forward path from s_0 to the overlap state with the backward path from the overlap state to s_g.

To guide convergence, a confidence reward from the LLM (how promising the chosen action is given the state, remaining budget, and accumulated cost). This bidirectional structure reduces the depth each tree must explore before meeting, mitigating the exponential blow-up faced by single-tree search on long-horizon tasks. Algorithm 1 is the detailed pseudocode of bidirectional search.

D Use of LLMs

Large Language Models were used only to improve the clarity and presentation of the manuscript. They did not contribute to the research design, experiments, analysis, or conclusions, for which the authors take full responsibility.

E Prompting

Action Evaluation. We prompt the LLM with the current state, partial plan, candidate actions, goal state, and budget usage, and ask it to classify each action as good or bad. We use the log-likelihood of the *good* label as a confidence for each 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 and the
    corresponding time consumption:
Pick up a block, 1 min
Unstack a block from on top of another block,
    1 min
Put down a block, 20 min
Stack a block on top of another block, 1 min
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
```

Action costs	L=2	L=4	L=6	L=8	L=10	L=12
$[20, 1, 1, 1]$	0%	8%	33%	63%	64%	70%
$[1, 20, 1, 1]$	0%	8%	32%	59%	59%	59%
$[1, 1, 20, 1]$	0%	16%	52%	74%	71%	78%
$[1, 1, 1, 20]$	0%	0%	14%	39%	43%	52%
$[20, 1, 20, 1]$	0%	0%	12%	35%	30%	35%
$[1, 20, 1, 20]$	0%	4%	14%	28%	26%	29%
$[20, 20, 1, 1]$	0%	4%	8%	16%	12%	18%
$[1, 1, 20, 20]$	0%	4%	8%	12%	32%	49%
$[20, 1, 1, 20]$	0%	8%	9%	37%	46%	54%
$[1, 20, 20, 1]$	0%	8%	40%	66%	63%	70%

Table 4: **Solution shift rate under nonuniform action costs.** The tasks are classified by the length of optimal solution from L=2 to L=12. The first column represents the cost schedule allocated for each action: pick-up(pu), unstack(un), put-down(pd), stack(st). For example, [20, 1, 1, 1] means the cost of pick-up is 20, and the cost of other actions is 1. An optimal solution is counted as *shifted* if its *sequence of action types* under the given schedule is not exactly the same as under the uniform-cost solution. The rest columns represent the percentage of tasks whose optimal solutions are shifted.

Algorithm 1 Bidirectional Search

Require: Initial node $v_0 \in \mathcal{V}$, weight ω, goal node $v_g \in \mathcal{V}$, max iterations L, budget limit $\mathcal{B}$

Initialize confidence reward $R_{conf} : \mathcal{V} \to \mathbb{R}$
Initialize feasible actions $A : \mathcal{V} \times \mathcal{B} \to \mathbb{A}$, child nodes $c : \mathcal{V} \times \mathcal{A} \to \mathcal{V}$
Initialize unvisited verifier $U : \mathcal{V} \to \{0, 1\}$
Initialize node set $\mathcal{V}_f \leftarrow \{v_0\}$, $\mathcal{V}_b \leftarrow \{v_g\}$
Initialize empty plan $\pi \leftarrow \emptyset$
for $t \leftarrow 0$ **to** $L - 1$ **do**
  $v_f^* \leftarrow \arg\max_{v \in Leaf(v_0)} (R_{conf}(v))$
  for $a \in A(v_f^*, \mathcal{B})$ **do**
    $v^* \leftarrow c(v_f^*, a)$
    $\mathcal{V}_f \leftarrow \mathcal{V}_f \cup \{v^*\}$
  end for
  $v_b^* \leftarrow \arg\max_{v \in Leaf(v_g)} (R_{conf}(v))$
  for $a \in A(v_b^*, \mathcal{B})$ **do**
    $v^* \leftarrow c(v_b^*, a)$
    $\mathcal{V}_b \leftarrow \mathcal{V}_b \cup \{v^*\}$
  end for
  $v_{\text{overlap}} \leftarrow \mathcal{V}_f \cap \mathcal{V}_b$
  if $v_{\text{overlap}} \neq \emptyset$ **then**
    $\pi \leftarrow \textsc{ExtractPlan}(v_{\text{overlap}})$
    break
  end if
end for
return π

```
    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 blue block is
     clear, the orange block is clear, the
    hand is empty, 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 blue block is on
    top of the orange block.
My previous plan is:
[PREVIOUS PLAN]
pick up the blue block
[PREVIOUS PLAN END]
The current time usage for the previous plan
    is 1 min
My plan is as follows:
[PLAN]
stack the blue block on top of the orange
    block
[PLAN END]

As initial conditions I have that, <init state
    >\\
My goal is to <goals>
My previous plan is:
[PREVIOUS PLAN]
<previous plan>
[PREVIOUS PLAN END]
The current time usage for the previous plan
    is <current time usage> min
My plan is as follows:
[PLAN]
```

Self-Evaluated Confidence Reward. We prompt the LLM with the current state, partial plan, selected action, goal state, and budget usage, and ask it to classify the action as *good* or *bad*. The confidence score is computed as the log-likelihood assigned to the *good* label.

```
I am playing with a set of blocks where I need
     to arrange the blocks into stacks. Here
    are the actions I can do and the
    corresponding time consumption:
Pick up a block, 1 min
Unstack a block from on top of another block,
    1 min
Put down a block, 20 min
Stack a block on top of another block, 1 min

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] The initial state: I have that,
    the yellow block is on the table. The red
    block is on the top of the yellow block.
    The green block is on the top of the red
    block. The blue block is on the green
    block. The blue block is clear. The hand
    is clear.
The current state: I have that, the yellow
    block is on the table. The red block is on
     the top of the yellow block. The green
    block is on the top of the red block. The
    blue block is on the table. The blue block
     is clear. The hand is clear.
Goal: The green block is on the top of blue
    block.
[PREVIOUS ACTION]
unstack the blue block from the green block.
put down the blue block.
We have used 21 minutes.
Our time limit is 33 minutes
[ACTION] unstack the green block from the red
    block
[EVALUATION] good

[STATEMENT]
The initial state: I have that, <init state>
The current state: I have that, <current state
    >
Goal: <goal>
[PREVIOUS ACTION] <previous actions>
We have used <current time usage> minutes so
    far.
Our time limit is <time limit> minutes.
[ACTION] <action>
[EVALUATION]
```

Chain-of-Thought (CoT) prompt used for GPT-4.1, Claude-Opus-4.1, and Qwen3-8B:

```
I am playing with a set of blocks where I need
     to arrange the blocks into stacks. Here
    are the actions I can do and the
    corresponding time consumption:
Pick up a block, 1 min
Unstack a block from on top of another block,
    1 min
Put down a block, 20 min
Stack a block on top of another block, 1 min

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.

IMPORTANT: You must respond with ONLY the plan
     in the exact format shown below. Do not
    include any explanations, analysis, or
    additional text.

[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 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 blue block is on
    top of the orange block.
The time limit is 22 min
My plan is as follows:
[PLAN]
pick up the blue block
stack the blue 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 red block is on top of
    the yellow block, the blue 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 yellow block and the orange
    block is on top of the blue block.
The time limit is 45 min
My plan is as follows:

[PLAN]
unstack the red block from on top of the
    yellow block
put down the red block
pick up the blue block
stack the blue block on top of the yellow
    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,
     <init state>
My goal is to <goals>
The time limit is <time limit> min
My plan is as follows:
[PLAN]
```