Task-Tree A-Star: A Hierarchical Framework for Long-Horizon Task Planning with Large Language Models

Zihao Li[1,2], Zhilin Zhang[1,2], Yun Lu[1,2], Guang Li[1,*]
[1]*Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences*, Chongqing 400714, China
[2]*Chongqing School, University of Chinese Academy of Sciences*, Chongqing 400714, China
[*]Corresponding author:liguang@cigit.ac.cn

***Abstract*—While Large Language Models show promise in agentic planning, their reliability drops sharply in long-horizon scenarios. Existing methods often face a trade-off between robustness and planning granularity. Local replanning often misses root errors buried in earlier steps, causing repetitive failures, whereas global replanning wastes valid progress by starting over. Additionally, LLM-generated plans often lack physical grounding, leading to actions that violate environmental constraints. We propose Task-Tree A-Star, a hierarchical framework that decouples high-level task decomposition from low-level action execution. By structuring tasks into a tree, TTA* enables backtracking to specific decision points rather than restarting, thereby preserving completed subtasks. Inside each node, a localized A* search generates executable actions without the combinatorial overhead of global search. Experiments in a complex simulation environment demonstrate that TTA* achieves higher success rates than baselines, particularly in unseen scenarios.**

***Index Terms*—Large Language Models, Task Planning, Hierarchical Planning, A* Search**

I. INTRODUCTION

Task planning enables intelligent agents to decompose high-level goals into sequences of executable actions [1]. Recently, Large Language Models (LLMs) have become widely used in this field because of their commonsense knowledge [2]. LLMs support planning in several ways: they decompose complex tasks into manageable subgoals [3], [4], serve as implicit world models for environmental reasoning [5], [6], or directly generate action sequences [7].

Existing LLM-based task planning methods generally follow three strategies. (1) Open-loop methods generate a static action sequence in a single pass, which the agent executes without intermediate feedback [8], [9]. (2) Closed-loop methods incorporate environmental feedback. At each step, the model determines the next action based on the current state and previous outcomes [10], [11]. (3) Search-based methods explore multiple candidate sequences. These approaches typically construct a planning tree to evaluate future paths for more robust decision making [12]–[14].

This work is supported in part by Key Project of Chongqing Natural Science Foundation (No. CSTB2025NSCQLZX0061), in part by the Science and Technology Innovation Key R&D Program of Chongqing(CSTB2025TIADSTX0023, CSTB2023TIADSTX0031), in part by Chongqing University of Technology Undergraduate Education and Teaching Reform Research Project (General Project), (Project No.: 2024YB76).

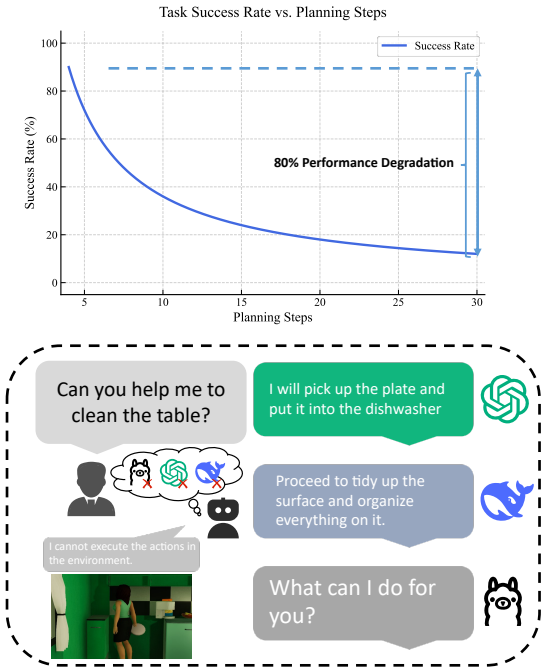

Fig. 1. Challenges in LLM-based task planning. Top: the success rate of CoT planning decreases as the required number of steps increases, which reflects the difficulty of long-horizon tasks. Bottom: LLM outputs may lack grounding, and generated natural language instructions may fail to map to admissible actions in the environment.

Despite these advances, performance often drops as task horizons increase [3]. As illustrated in Fig. 1, long-horizon task planning remains vulnerable to both error accumulation and weak action grounding. This limitation is mainly related to two issues in current planning paradigms. First, current replanning mechanisms often struggle to balance robustness and efficiency. Local replanning can miss decomposition errors introduced in earlier steps, which may lead to repeated but ineffective revisions [15]. Global replanning discards valid progress by reconstructing the full plan and may introduce new errors [15]. Second, LLM outputs often lack grounding. Generated actions are often invalid that are unaccepted by the environment, ultimately causing task failure [16].

To address these limitations, we propose Task-Tree A*

(TTA*), a hierarchical planning framework. TTA* constructs a high-level task tree where nodes represent distinct subtasks, which the agent executes sequentially. If a subtask fails, the system returns to the nearest ancestor node with an unexplored branch instead of restarting the full task. This mechanism pivots to an alternative path while preserving progress from successful parent subtasks. A localized A* search then generates low-level actions within the selected subtask. This hierarchical structure limits the search scope to the selected subtask. The A* search uses expert trajectories in path evaluation to improve action grounding. It scores the current path using consistency with past successful trajectories. This filters out invalid actions from the LLM. It also uses a heuristic term to estimate future task progress. Together, these components improve action executability during planning.

The main contributions of this work are:

- We introduce a Task Tree constructed by merging diverse plan proposals. By identifying failure-prone nodes and appending alternative branches, we establish a subtask-level backtracking mechanism. This allows the agent to switch branches after subtask failure while preserving progress on completed subtasks.
- We design a hybrid evaluation function for A* search. It uses an LLM-based heuristic to predict future costs and integrates consistency with expert trajectories to improve action grounding. This combination helps reduce invalid actions without relying on simulation rollouts during A* scoring.
- We propose TTA*, a hierarchical framework that improves the trade-off between replanning robustness and planning scope. By decoupling high-level task management from low-level action execution, this architecture mitigates the performance degradation in long-horizon tasks.

II. RELATED WORK

A. Generative and Refinement-based Task Planning

Task decomposition is a common application of LLMs in planning. LLMs act as high-level planners to translate natural language goals into sequences of manageable subtasks [3], [17]. This process links user goals to executable agent behavior. By converting goals into subtasks, LLMs support more autonomous agent behavior [2], [18]. Most existing approaches generate subtasks as a linear sequence, whereas our method organizes them into a task tree.

Early LLM-based planning focused on generating linear action sequences. Following Chain-of-Thought (CoT) reasoning [8], frameworks like ReAct [19] and SayCan [16] interleave reasoning with action execution [9], [10]. Because of their linear structure, these techniques are often effective for simple tasks but less reliable in complex long-horizon settings.

Feedback mechanisms were introduced to address this limitation. Methods such as Reflexion [20] and Self-Refinement [21] enable agents to adjust plans based on prior actions and feedback [11], [22]. However, reliability often drops as task horizons increase. Current methods still struggle to balance robustness and planning cost. For example, local replanning can overlook decomposition errors introduced in earlier steps, which may lead to repeated failures. Global replanning, on the other hand, wastes valid progress by starting over. In addition, LLM-generated plans may lack grounding, which can lead to actions that violate environmental constraints [15].

B. Search-based Task Planning with LLMs

Search-based methods provide an alternative to iterative refinement by exploring a tree of action sequences. This approach can improve planning robustness by considering multiple candidate paths [2], [4]. Algorithms like Tree of Thoughts (ToT) [12] use LLMs to generate multiple potential next steps for general problem-solving. In task planning, recent works integrate LLMs with Monte Carlo Tree Search (MCTS). These methods simultaneously propose actions and evaluate future states [13], [14], [23], [24].

The main drawback of these methods is high computational cost. Existing methods typically conduct a global search directly on the action space. Finding a feasible plan often involves exploring many nodes, where each expansion requires expensive LLM calls [13]. Errors further increase costs by expanding the search space needed for recovery [12]. Our method differs by decoupling the search process into task and action levels. We perform a high-level search for subtasks and a localized search for actions. This separation limits the search scope for each step. It effectively reduces the computational load by avoiding the massive node expansions found in global action search planners.

III. METHOD

We introduce a hierarchical framework that combines high-level LLM planning with classical search. The framework consists of two stages: (1) Guided Task Tree Generation, where the LLM produces multiple subtask sequences that are merged into a task tree with alternative branches; and (2) A* Search with LLM Action Proposals, which searches for low-level actions within the current subtask. The A* algorithm uses a custom evaluation function to rank the actions proposed by the LLM. We describe the high-level tree construction in Section A and the low-level search formulation in Section B.

A. Guided Task Tree Generation

We use the LLM to act as multiple planners with different styles. These planners generate high-level task sequences using a standard format, such as Navigate(place). We then combine these sequences into a single subtask tree. Fig. 2 and Algorithm 1 show this process. We merge nodes that have the same parent and content. The main trunk of the tree shows the path where most planners agree.

The algorithm then goes over this task tree to find nodes that could fail because of environmental uncertainty, like Acquire(plate, fridge). We identify these nodes using predefined action-type rules. In our implementation, actions whose success depends on uncertain object existence or location are

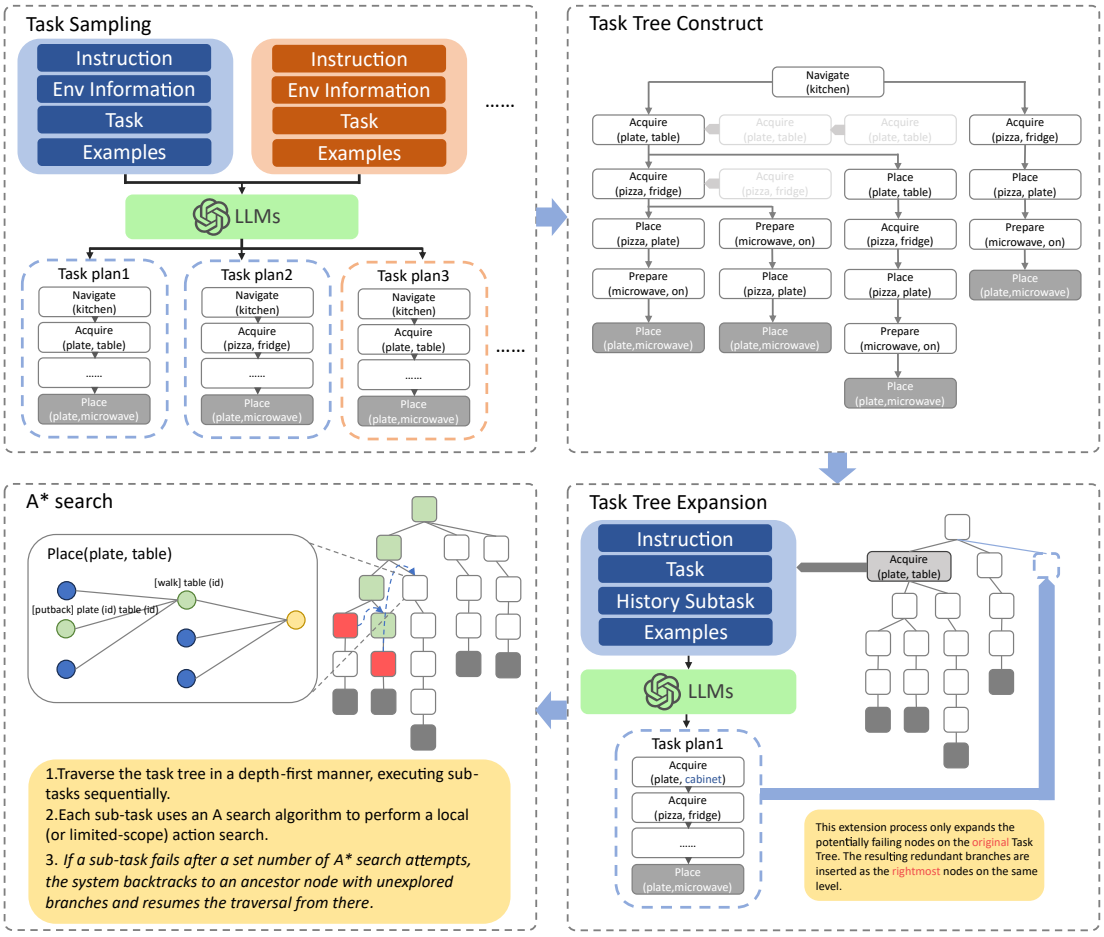

Fig. 2. The workflow of TTA*. First, TTA* guides an LLM to generate diverse plans. These plans are then synthesized into a task tree. To improve robustness, the program detects possible failure nodes in the tree and supplements them with alternative branches. Ultimately, TTA* goes through the tree, using A* search to generate executable actions for each subtask.

marked as failure-prone, while deterministic state-transition actions are not expanded by default. The LLM is then used to infer alternative locations for the target object. For example, a plate may also appear in a cabinet or on a table. This inference leads to the creation of new plan branches, which are added as right siblings to the original node (see Fig. 2 Task Tree Expansion and Algorithm 2). In this structure, the leftmost branch is used as the default execution path, and the right siblings are reserved as alternative branches.

During execution, as shown in Algorithm 3, the agent traverses the subtask tree in a depth-first, left-to-right order. Each subtask node is associated with a deterministic termination condition, which is checked against the current environment state. If a subtask fails, the algorithm backtracks to the nearest ancestor with an unexplored branch and continues with the next branch. This process is repeated upon each failure. If a path to a leaf node is successfully completed, the overall task is considered accomplished. The search terminates unsuccessfully only when all possible paths in the task tree have been exhausted. This recursive backtracking mechanism allows the agent to revise earlier subtask choices without restarting the full plan, while preserving completed ancestor subtasks.

B. A* Search with LLM Action Proposals

After the task tree is constructed, the agent selects one node as the current subtask. To generate a low-level action sequence for this subtask, we use an LLM-guided A* search algorithm. Given the current observation and action history, the LLM proposes a set of candidate actions. Each candidate is then scored by a heuristic function. The algorithm then expands the candidate with the best evaluation score, and this evaluation-expansion cycle is iteratively applied to the newly generated candidates. If the subtask cannot be completed within a preset number of search iterations, it is deemed a failure, triggering the backtracking mechanism of the task tree.

The effectiveness of the A* algorithm is fundamentally dependent on the precision of its evaluation function in estimating the cost of a specific node [25]. This function is formally defined as:

$$f(n) = g(n) + h(n) \tag{1}$$

Here, n denotes the input state, while $g(n)$ and $h(n)$ represent the known cost from the initial state to the current state

Algorithm 1 Task Tree Construct

```
function CONSTRUCT(plan_list)
    root_node ← new Node(null)
    for each plan in plan_list do
        current_node ← root_node
        for each subtask in plan do
            match ← FindMatch(current_node, subtask)
            if match is null then
                new_child ← new Node(subtask)
                AddChild(current_node, new_child)
                current_node ← new_child
            else
                current_node ← match
            end if
        end for
    end for
    return root_node
end function
function FINDMATCH(parent_node, subtask)
    child_node ← parent_node.first_child
    while child_node is not null do
        if child_node.subtask == subtask then
            return child_node
        end if
        child_node ← child_node.next_sibling
    end while
    return null
end function
```

Algorithm 2 Task Tree Expansion

```
function EXPAND(root_node)
    failure_nodes ← []
    FindFailures(root_node, failure_nodes)
    for each target_node in failure_nodes do
        alt_plan_list ← LLM.Generate(target_node)
        current_node ← target_node
        for each plan in alt_plan_list do
            new_node ← ConstructBranch(plan)
            InsertSibling(current_node, new_node)
            current_node ← new_node
        end for
    end for
    return root_node
end function
procedure FINDFAILURES(node, list)
    if node is null then return
    end if
    if IsFailureProne(node.subtask) then
        list.append(node)
    end if
    FindFailures(node.first_child, list)
    FindFailures(node.next_sibling, list)
end procedure
```

and the heuristic estimate of the cost from current state to the goal state, respectively. Inspired by [25], we use $g(n)$ to measure how consistent the current trajectory is with successful reference trajectories. Specifically, it compares the current trajectory with expert trajectories from similar tasks. This term combines a path consistency cost C_{path} and a state consistency cost C_{state}, formulated as:

$$g(n) = \alpha \cdot C_{path}(H_n, G) + (1 - \alpha) \cdot C_{state}(s_n, G) \quad (2)$$

C_{path} first constructs a dataset of similar trajectories $D_{rel} \subset D_{expert}$ based on the textual descriptions of the task goal and the current state. The semantic similarity between each reference trajectory in this dataset and the agent's current trajectory is then calculated. These similarity values are aggregated by a weighted sum to obtain a single consistency score. The path consistency cost is defined as one minus this score, as formulated below:

$$C_{path}(H_n, G) = 1 - \sum_{\tau_i \in D_{rel}} w_i \cdot sim(E(H_n), E(H_{\tau_i})) \quad (3)$$

Algorithm 3 Recursive Execution with Backtracking

```
function EXECUTE(node)
    if node is null then return FAILURE
    end if
    status ← AStarExecute(node)
    if status is FAILURE then
        fallback ← FindFallback(node)
        return Execute(fallback)
    end if
    if node.is_leaf then return SUCCESS
    end if
    if Execute(node.first_child) is SUCCESS then
        return SUCCESS
    end if
    fallback ← FindFallback(node.first_child)
    return Execute(fallback)
end function
```

$$w_i = \text{Softmax}(\frac{sim(E(G), E(G_{\tau_i}))}{T}) \quad (4)$$

Where H_n and H_{τ_i} represent the historical action sequences of the current state and the expert trajectory, respectively. $E(\cdot)$ is the semantic embedding model and $sim(\cdot)$ is the cosine similarity. T is a hyperparameter that regulates the sharpness of the Softmax probability distribution, determining whether the weights are focused on the optimal match (low T) or distributed more uniformly (high T).

C_{state} measures how different the current state s_n is from states at a comparable stage of expert trajectories for similar tasks. First, we build a state set S from trajectories with similar subtask goals and with either the same trajectory length or the same last action as the current trajectory. The state consistency score is the maximum similarity between the current observation $obs(s_n)$ and all state observations $obs(s_\tau)$

in this set S. The corresponding state consistency cost is then defined as follows:

$$C_{state}(s_n, G) = 1 - maxsim(obs(s_n), obs(s_\tau)) \quad (5)$$

The future cost term $h(n)$ estimates the remaining cost from the current action and the relevance of this action to future task completion. The formula is defined as follows:

$$h(n) = \alpha \cdot C_{action}(s_n, a, G) + (1 - \alpha) \cdot C_{key}(a, G) \quad (6)$$

For a given state s_n and goal description G, C_{action} uses one LLM query to sample m future action sequences. The mean length of these sequences is then normalized to the $(0, 1)$ interval to estimate the remaining cost. C_{action} is formally defined as:

$$C_{action} = \text{Norm}\left(\frac{1}{m}\sum_{i=1}^{m} |LLM(obs(s_n), a, G)|_i\right) \quad (7)$$

$$Norm(x) = 1 - e^{-\left(\frac{x}{8}\right)^3} \quad (8)$$

In the calculation of C_{key}, we treat action a as a key action, namely an action that appears in many successful trajectories. Actions that tend to appear later in successful trajectories receive higher scores. Based on this heuristic assumption, the key action cost C_{key} is defined as follows:

$$C_{key} = 1 - \frac{1}{|D_{prog}|}\sum_{\tau_i \in D_{prog}} \frac{\text{index}(a, \tau_i)}{|\tau_i|} \quad (9)$$

D_{prog} is a subset of data extracted from D_{expert} that contains the action a. This dataset is partitioned into three classes: trajectories similar to the current subtask objective, trajectories dissimilar to it, and a remaining miscellaneous set. We construct D_{expert} from ground-truth trajectories in the training split, and use it only for cost estimation during A* search. We ensure strictly non-overlapping tasks between the training and evaluation sets.

The proposed framework combines high-level subtask-tree traversal with low-level A* search for hierarchical long-horizon planning. The low-level A* algorithm searches within the current subtask, which limits the scope of action search. When the high-level plan needs revision, the framework returns to the nearest ancestor with unexplored branches instead of rebuilding the full plan. It then backtracks on the subtask tree to explore alternative branches.

IV. EXPERIMENTS

A. Experimental Setup

1) Environment: We evaluate our approach in VirtualHome [26]. VirtualHome is a large-scale indoor simulator for evaluating household task planning. The environment includes multiple rooms, hundreds of interactive objects and containers, and physical constraints on action execution. Its tasks are long-horizon household activities, such as cleaning a table or preparing food, which require multi-step planning and execution. After each action, VirtualHome provides an updated scene graph, which enables closed-loop evaluation. The LLM used in our experiment is Qwen3-8B [27].

2) Metrics: We follow [15] and adopt three core metrics: Execution Rate (Exec.), Success Rate (SR), and Goal Condition Rate (GCR). Exec. measures the proportion of generated actions that satisfy environment constraints and can be executed successfully. SR measures whether the agent completes the full task and reaches a final state that satisfies the task goal. GCR is a softer metric that measures the proportion of goal conditions satisfied in the final state.

To further assess robustness under increasing task difficulty, we additionally report Brittleness. Brittleness measures the relative drop in SR as task difficulty increases. A lower value indicates better robustness to task complexity. Brittleness is formally defined as follows:

$$Brittleness_i = \frac{SR_1 - SR_i}{SR_1} \times 100\% \quad (10)$$

where SR_i denotes the success rate of the task at difficulty level i.

3) Datasets: Following [13], [28], we build evaluation sets with two difficulty levels, simple and hard, to test performance under unseen environments, objects, and tasks. These include a Base set with 850 tasks, and three Generalization sets with novel environments, novel objects, and novel tasks, containing 850, 700, and 250 tasks, respectively. Simple tasks contain one goal, while hard tasks contain two or more goals. Example goals include organizing tableware, cleaning bedrooms, and preparing food. These tasks require long-horizon planning across multiple steps. For the hard setting, we further construct a Novel Composition set with 100 tasks that combine novel objects, environments, and tasks. We construct D_{expert} by randomly sampling 550 training trajectories that follow the Base-set distribution. These were generated by an Oracle agent using regression planning with handcrafted heuristics under global observation [28].

B. Main Results

We compare our method against three baselines: CoT [8], ProgPrompt [10], and LLM-MCTS [13]. CoT and ProgPrompt generate linear action sequences. When execution fails, CoT regenerates the full plan, while ProgPrompt replans from the failure point. LLM-MCTS, a tree-search-based method, also performs holistic replanning. In our A* search, we sample 3 actions at each expansion step and set the maximum search depth to 5. For all baselines, we use their recommended configurations. Table I presents a comprehensive comparison between our proposed method and these advanced baselines in VirtualHome.

On simple tasks, our method achieves the highest SR on both datasets, although the gap over the baselines is small.

When the tasks become more complex and require longer action sequences, the performance of all methods declines. These tasks are hard for traditional methods that simply output linear action sequences since they have to plan again and again after making mistakes. On the other hand, our method has a rather high success rate. On the Base and Novel Scene and Task test sets, our method achieves 75.26% and 68.46% SR,

TABLE I
EVALUATION ON VIRTUALHOME-ENV

		Base			Novel Scene and Task			Novel Composition		
		SR	GCR	Exec.	SR	GCR	Exec.	SR	GCR	Exec.
Simple	CoT	89.25 (±0.35)	89.25 (±0.35)	95.53 (±0.19)	75.21 (±0.27)	75.21 (±0.27)	81.87 (±0.32)	-	-	-
	ProgPrompt	90.58 (±0.24)	90.58 (±0.24)	97.53 (±0.05)	77.25 (±0.48)	77.25 (±0.48)	84.10 (±0.49)	- -	- -	- -
	LLM-MCTS	92.00 (±0.35)	92.00 (±0.35)	**98.53** (±0.09)	80.54 (±0.39)	80.54 (±0.39)	96.33 (±0.32)	- -	- -	- -
	Ours	**92.17** (±0.12)	**92.17** (±0.12)	98.43 (±0.09)	**80.92** (±0.45)	**80.92** (±0.45)	**96.48** (±0.11)	- -	- -	- -
Difficult	CoT	43.04 (±0.10)	63.19 (±0.19)	71.63 (±0.25)	40.72 (±9.05)	60.22 (±1.72)	67.38 (±1.28)	5.67 (±0.94)	19.83 (±0.85)	40.47 (±0.48)
	ProgPrompt	67.63 (±0.76)	77.56 (±0.27)	83.37 (±0.24)	59.32 (±6.49)	72.65 (±2.86)	80.32 (±1.64)	16.67 (±1.89)	34.83 (±0.85)	47.70 (±2.67)
	LLM-MCTS	71.56 (±1.19)	**85.07** (±0.38)	95.57 (±0.46)	66.96 (±3.09)	79.45 (±3.21)	**91.11** (±4.24)	37.00 (±0.82)	60.67 (±0.62)	75.20 (±0.64)
	Ours	**75.26** (±0.55)	84.74 (±0.70)	**95.70** (±0.49)	**68.46** (±3.23)	**79.63** (±2.93)	91.10 (±4.11)	**52.00** (±2.45)	**69.33** (±0.85)	80.93 (±0.69)

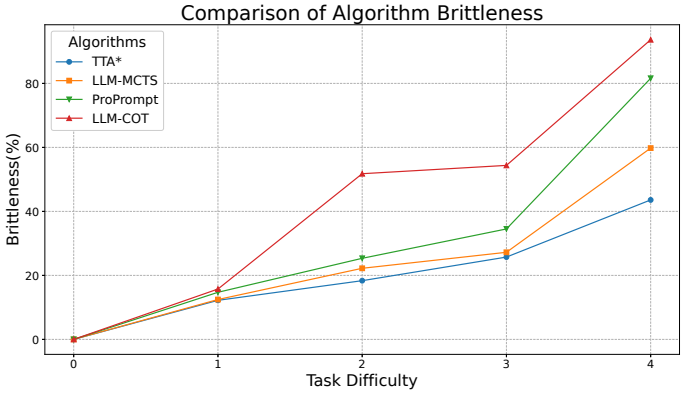

Fig. 3. A comparison of algorithm brittleness under increasing task difficulty. The brittleness metric is calculated based on the degradation of success rate relative to the base task.

TABLE II
ABLATION STUDY ON COMPONENTS

Method Variants	**Long-Term Base**		
	SR	**GCR**	**Exec.**
Hierarchical Architecture			
w/o Task Tree (Global Search)	44.89±0.48	64.22±0.42	72.83±0.42
Action Selection Strategy			
Majority Vote (Max)	69.11±0.65	81.26±0.53	95.37±0.40
Stochastic Sampling (Freq.)	67.26±0.46	79.89±0.31	93.60±0.64
Heuristic Function Composition			
Only Past Cost $g(n)$	71.33±0.36	82.70±0.23	95.60±0.14
Only Future Cost $h(n)$	72.59±0.58	83.00±0.18	96.20±0.22
TTA* (Full Method)	**75.26±0.55**	**84.74±0.70**	**95.70±0.49**

respectively, outperforming all baselines. As shown in Fig. 3, our method also shows lower brittleness as task difficulty increases. Its SR decreases more slowly than those of the baselines as task difficulty increases.

On the most challenging test set Novel Composition, our method achieves an SR of 52.00%. We attribute this result to the use of Task Trees, which introduce alternative branches at the subtask level. These branches allow the agent to backtrack to earlier subtask choices while preserving completed ancestor subtasks. At the action level, search is restricted to the current subtask instead of the full task.

C. Ablation Study

We perform ablation studies on the Difficult Base set to examine the contributions of different components. The results are summarized in the Table II.

When the Task Tree is removed and only A* search is retained, all three metrics decrease substantially. This result shows that high-level structured planning is important for complex long-horizon tasks. The Task Tree provides top-down guidance by decomposing a complex problem into a sequence of logically coherent subtasks, thereby narrowing the search space for low-level action planning.

For the evaluation-function ablation, we replace it with two alternative action-selection strategies: Max, which chooses the most frequent action in the LLM-generated sequences, and Frequency, which samples actions from their empirical distribution. The results show that our evaluation function performs better than these two alternatives.

Finally, the ablation on $g(n)$ and $h(n)$ shows that $h(n)$ contributes more to overall performance. A well-designed heuristic cost function can improve the success rate. This trend is also consistent with the ablation results reported in [25].

V. CONCLUSION

In this work, we presented TTA*, a hierarchical framework for long-horizon task planning with LLM-based agents. Our approach addresses limitations of existing planning methods through structured task decomposition. By organizing tasks into a tree, TTA* allows the agent to return to earlier subtask decisions when the current branch fails. This mechanism preserves completed ancestor subtasks while allowing alternative branches to be explored, which helps balance local revision and full-plan reconstruction. In addition, the framework uses action evaluation within each subtask to prioritize actions that are more consistent with environment constraints and successful reference trajectories.

Our evaluation in VirtualHome shows that TTA* achieves strong results across the evaluated settings. The method also shows better robustness as task complexity increases. However, several limitations remain in the current version. The current framework relies only on text input, which limits its applicability to real-world robotic settings. In addition, the framework does not yet use completed-task data for continual improvement. Inference latency also remains a practical limitation for interactive execution.

Future work will incorporate visual input to support deployment in real-world robotic settings. We also plan to introduce memory modules that enable the agent to reuse past experience and acquire new skills over time. To reduce inference overhead, we will further explore smaller models for low-level planning.

REFERENCES

[1] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[2] H. Wei, Z. Zhang, S. He, T. Xia, S. Pan, and F. Liu, "Plangen-llms: A modern survey of llm planning capabilities," *arXiv preprint arXiv:2502.11221*, 2025.

[3] Y. Wu, J. Zhang, N. Hu, L. Tang, G. Qi, J. Shao, J. Ren, and W. Song, "Mldt: Multi-level decomposition for complex long-horizon robotic task planning with open-source large language model," in *International Conference on Database Systems for Advanced Applications*. Springer, 2024, pp. 251–267.

[4] P. Cao, T. Men, W. Liu, J. Zhang, X. Li, X. Lin, D. Sui, Y. Cao, K. Liu, and J. Zhao, "Large language models for planning: A comprehensive and systematic survey," *arXiv preprint arXiv:2505.19683*, 2025.

[5] N. Dainese, M. Merler, M. Alakuijala, and P. Marttinen, "Generating code world models with large language models guided by monte carlo tree search," *Advances in Neural Information Processing Systems*, vol. 37, pp. 60 429–60 474, 2024.

[6] S. Wang, Z. Fei, Q. Cheng, S. Zhang, P. Cai, J. Fu, and X. Qiu, "World modeling makes a better planner: Dual preference optimization for embodied task planning," *arXiv preprint arXiv:2503.10480*, 2025.

[7] J. Zhang, L. Tang, Y. Song, Q. Meng, H. Qian, J. Shao, W. Song, S. Zhu, and J. Gu, "Fltrnn: Faithful long-horizon task planning for robotics with large language models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 6680–6686.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[9] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[10] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," *arXiv preprint arXiv:2209.11302*, 2022.

[11] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2998–3009.

[12] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 11 809–11 822, 2023.

[13] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 31 967–31 987, 2023.

[14] S. Hao, Y. Gu, H. Ma, J. Hong, Z. Wang, D. Wang, and Z. Hu, "Reasoning with language model is planning with world model," *arXiv preprint arXiv:2305.14992*, 2023.

[15] M. Hu, Y. Mu, X. Yu, M. Ding, S. Wu, W. Shao, Q. Chen, B. Wang, Y. Qiao, and P. Luo, "Tree-planner: Efficient close-loop task planning with large language models," 2024.

[16] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[17] A. Prasad, A. Koller, M. Hartmann, P. Clark, A. Sabharwal, M. Bansal, and T. Khot, "Adapt: As-needed decomposition and planning with language models," *arXiv preprint arXiv:2311.05772*, 2023.

[18] J. Wang, E. Shi, H. Hu, C. Ma, Y. Liu, X. Wang, Y. Yao, X. Liu, B. Ge, and S. Zhang, "Large language models for robotics: Opportunities, challenges, and perspectives," *Journal of Automation and Intelligence*, vol. 4, no. 1, pp. 52–64, 2025.

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[20] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8634–8652, 2023.

[21] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang *et al.*, "Self-refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 534–46 594, 2023.

[22] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv preprint arXiv:2207.05608*, 2022.

[23] Z. Chen, D. Chen, R. Sun, W. Liu, and C. Gan, "Scaling autonomous agents via automatic reward modeling and planning," *arXiv preprint arXiv:2502.12130*, 2025.

[24] J. Y. Koh, S. McAleer, D. Fried, and R. Salakhutdinov, "Tree search for language model agents," *arXiv preprint arXiv:2407.01476*, 2024.

[25] Y. Zhuang, X. Chen, T. Yu, S. Mitra, V. Bursztyn, R. A. Rossi, S. Sarkhel, and C. Zhang, "Toolchain*: Efficient action space navigation in large language models with a* search," *arXiv preprint arXiv:2310.13227*, 2023.

[26] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "Virtualhome: Simulating household activities via programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8494–8502.

[27] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[28] X. Puig, T. Shu, S. Li, Z. Wang, Y.-H. Liao, J. B. Tenenbaum, S. Fidler, and A. Torralba, "Watch-and-help: A challenge for social perception and human-ai collaboration," *arXiv preprint arXiv:2010.09890*, 2020.