A^4: Tree-Based Adaptive Action Anticipation and Allocation for LLM Agent Evolution

Anonymous ACL submission

Abstract

The evolution of LLM-based agents is essential, as it enables sustained improvement in their problem-solving abilities. Prior work has focused on leveraging training data derived either from an agent's own behavior, such as refining decision sequences, or from stronger external agents, including expert or teacher models, to improve performance on agent tasks. However, when exploring candidate trajectories, these methods allocate their sampling budget uniformly across all decision steps, overlooking that individual actions contribute unequally to the final task outcome. To overcome this limitation, we propose A^4, a theoretically grounded framework that models the decision process as a tree and decomposes trajectory-level variance into per-step contributions, enabling adaptive allocation of the sampling budget toward high-uncertainty decision points. Extensive experiments across three representative interactive environments demonstrate that A^4 achieves 11.7% relative improvement over the strongest baseline. For reproducibility and future research, we release our code at https://anonymous.4open.science/r/A4-1674.

1 Introduction

Large Language Models (LLMs) have demonstrated impressive capabilities across a wide spectrum of tasks, including reasoning, coding, mathematics, summarization, and translation, yet they often struggle with interactive tasks that require long-horizon strategic reasoning (Luo et al., 2025; Guo et al., 2024). This gap has motivated the development of LLM-based agents, which aim to combine the language understanding of LLMs with planning and sequential decision-making abilities (Yao et al., 2022b). However, consistently enhancing these agents remains a fundamental challenge. Unlike standard LLM fine-tuning, upgrading an agentic system necessitates precise reward estimation for individual components, as the overall performance emerges from the complex interplay of reasoning, tool-use, and execution across diverse environments (Lin et al., 2025; Xi et al., 2025).

Prior work primarily focuses on training data curation. Some methods distill expert knowledge into student agents (Song et al., 2024a; Zeng et al., 2024; Chen et al., 2024), while others use rejection sampling (Lan et al., 2025) or trajectory rectification (Yuan et al., 2025; Lan et al., 2025) to obtain successful samples. To scale up exploration, search-based methods such as Monte Carlo Tree Search have been adopted to expand the candidate trajectory space. However, these approaches uniformly allocate sampling budgets across all decision steps without distinguishing their relative importance. In practice, interactive tasks often exhibit highly non-uniform decision landscapes: a wrong action at a pivotal step can render all subsequent actions irrelevant, while actions at non-critical steps have marginal impact on the outcome. Uniform sampling therefore under-explores critical decision points, causing potentially successful trajectories to be missed, limiting the diversity and quality of collected training data and ultimately constraining iterative self-training in complex agentic settings.

To address these limitations, we formulate the agentic process as a tree-structured computation to enable action-level importance estimation and adaptive resource allocation. Specifically, we represent the agent's decision-making as a tree where nodes are states and branches are potential solutions. This structure allows us to decompose the overall outcome variance into layer-wise components, identifying which decision steps contribute most to outcome uncertainty. We then employ an Adaptive Monte Carlo Tree Sampling (AMCTS) procedure that concentrates exploration on these critical steps, uncovering a richer and more diverse set of successful trajectories for self-training. By focusing sampling where it matters most, A^4 provides higher-quality training data for iterative pol-

icy improvement. Overall, the contributions of this paper can be summarized as follows:

- We propose A^4, an action-level evolutionary framework that models an agent's internal decision-making as a tree-structured process, enabling fine-grained decomposition of action-level importance beyond trajectory-level supervision.
- We introduce an AMCTS algorithm to directly anticipate action-level importance without relying on external expert models, providing stable and unbiased evolutionary signals for targeted agent optimization.
- Extensive experiments across three diverse iterative environments demonstrate that our theoretically grounded A^4 consistently outperforms baselines via providing more precise optimization signals.

2 Related Work

In this section, we organize the previous work according to the following progression: (1) methods that transfer capabilities from stronger expert models, (2) approaches that enhance correctness via multi-trajectory sampling and selection, and (3) techniques that rewrite or correct agent trajectories to provide improved training demonstrations.

Expert Distillation Strategy. These methods enhance LLM-based agent performance by training smaller models to imitate stronger experts (Lyu et al., 2025). Distillation is typically performed either through large-scale expert trajectory supervision, where the agent learns from sequences of high-quality actions, as in AgentBank (Song et al., 2024a) and AgentTuning (Zeng et al., 2024), or via behavior cloning of structured thought–action traces, which guides the agent using both intermediate reasoning steps and final actions (Kang et al., 2025; Xi et al., 2024; Chen et al., 2024). This lets smaller models acquire complex reasoning and decision-making patterns without requiring them to explore these strategies from scratch.

Sample-and-Select Strategy. A major class of work improves the agent by sampling multiple candidates and selecting higher-quality ones through consensus or evaluation. At inference time, Self-Consistency (Wang et al., 2022b) aggregates stochastic reasoning paths to reduce decision errors, while Tree-of-Thought (Yao et al., 2023) and MCTS-based planning (Zhao et al., 2023) conduct sampling-guided search over structured state spaces with intermediate scoring or rollout-based value estimation. Rejection Sampling Fine-Tuning (Lan et al., 2025) applies a similar filtering principle to retain successful expert trajectories for agent policy bootstrapping, emphasizing trajectory-level outcome correctness.

Trajectory Rewriting Strategy. These approaches explicitly correct or refine an agent's behavior at the trajectory level by transforming failed trajectories into corrected ones used for training. Representative approaches include iterative self-correction via verbal reflection (e.g., Reflexion (Shinn et al., 2023)), splicing correct continuations into failed trajectories using search (e.g., Agent-R (Yuan et al., 2025)), and utilizing teacher interventions (e.g., SCoRe (Lyu et al., 2025)) or valid sub-sequences from expert failures (Lan et al., 2025) to localize and fix errors.

3 Task Formulation

We consider a **single-agent** setting where one language model interacts with a **partially observable** environment, receiving noisy, incomplete observations of the true state, through a sequence of L discrete decision steps, and receiving a scalar reward $r(\tau) \in \mathbb{R}$ only at the terminal step.

Following prior work (Yuan et al., 2025; Song et al., 2024b), we formulate such tasks as a Partially Observable Markov Decision Process (POMDP), defined by $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, P, R)$. Here, $\mathcal{U}$ denotes the instruction space specifying task objectives, $\mathcal{S}$ the latent environment state space, $\mathcal{A}$ the action space, and $\mathcal{O}$ the observation space.

The transition function $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a stochastic kernel governing environment dynamics, while the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ provides sparse task-level feedback. In language-based agents, instructions, actions, and observations are in natural language.

Given an instruction $u \in \mathcal{U}$, a language agent parameterized by θ interacts with the environment by generating actions. At each step l, the agent observes textual feedback $o_l \in \mathcal{O}$ after executing action $a_l \in \mathcal{A}$, and maintains a history $\tau_{1:l} = (a_1, o_1, \ldots, a_l, o_l)$. The agent's policy π_θ produces the next action conditioned on the instruction and history: $a_{l+1} \sim \pi_\theta(\cdot \mid \tau_{1:l}, u)$. This loop continues until the task is completed or the maximum step number L is reached.

A complete interaction trajectory is denoted as $\tau = (a_1, o_1, \ldots, a_L, o_L)$, with probability $\pi_\theta(\tau \mid u) = \prod_{l=1}^{L} \pi_\theta(a_l \mid \tau_{1:l-1}, u)$, where L is the trajectory length, equivalently the depth of the search tree defined below. At the terminal step, the environment returns a reward $r(\tau) \in \mathbb{R}$, where $r(\tau) = R(s_L, a_L)$. In our benchmark tasks, $r(\tau) \in \{0, 1\}$ indicates task success ($r(\tau) = 1$) or failure ($r(\tau) = 0$), though the framework applies to general reward distributions.

Monte Carlo Search Tree (MCT) Structure. To systematically explore the decision space, we organize the agent's trajectories into a tree $\mathcal{T} = (V, E)$ of depth L, where each node $s_i^{(l)}$ represents the agent's state at level l and edges are action-observation pairs (a_l, o_l). At each node, the agent samples $B_l^{(t)}$ candidate actions, yielding $N_{\text{traj}} = \prod_{l=1}^{L} B_l^{(t)}$ leaf trajectories. Each node value $Y_i^{(l)}$ is estimated as the mean reward over reachable leaves, and the incremental variance $\text{Lv}_l = \text{Var}_l - \text{Var}_{l-1}$ isolates uncertainty introduced at each decision step. This decomposition enables adaptive sampling allocation to the most critical layers, as detailed in Section 4.2.

4 Methods

In this section, we provide a detailed description of the A^4 framework. First, we formalize the decision-making process as a layer-wise search tree, establishing the mathematical notation for nodes, actions, and branching factors in Section 4.1. Building on this, Section 4.2 introduces the AMCTS algorithm. This component employs a layer-wise variance decomposition to identify critical decision steps and dynamically reallocates sampling budgets, a process we justify through a formal theorem. Finally, we introduce a trajectory-based self-training phase, where high-quality trajectories discovered by search are incorporated back into the policy through iterative self-training, forming a closed-loop mechanism for continual performance improvement. A concrete example of this method is illustrated in Figure 1. For reference, Table 5 consolidates all notation used throughout the paper.

4.1 The Layer-wise Search Tree

To analyze the propagation of uncertainty in interactive environments, we represent the agent's exploration space using the MCT structure formally defined in Section 3.

Definition 4.1 (Layer-wise Search Tree)**.** Following the MCT structure in Table 5, a layer-wise search tree $\mathcal{T}$ of depth L is a directed graph where each level $l \in \{1, \ldots, L\}$ corresponds to a discrete decision step. Each node $s_i^{(l)}$ at level l represents the agent's state, encapsulating history $\tau_{1:l} = (a_1, o_1, \ldots, a_l, o_l)$ from the root $s^{(0)}$. The expansion from a parent node at level $l-1$ to its children at level l is characterized by the branching factor $B_l^{(t)}$, the number of discrete actions $a_l \sim \pi_\theta$ sampled. Thus, total search capacity is determined by the leaf trajectory count $N_{\text{traj}} = \prod_{l=1}^{L} B_l^{(t)}$, and each completed trajectory τ is associated with terminal reward $r(\tau)$.

In the formulation provided in Definition 4.1, the branching factor $B_l^{(t)}$ serves as the primary control variable for computational resource allocation. While existing adaptive MCTS approaches—such as UATS (Song et al., 2026)—provide effective node-level selection strategies within a single search episode, our setting requires optimizing the sampling budget across *decision layers* over multiple self-training iterations. This motivates a layer-level allocation mechanism: we treat $B_l^{(t)}$ as a dynamic parameter that adapts to the observed variance of each layer, redistributing the sampling budget to decision depths where it matters most for downstream policy learning. A detailed comparison with related adaptive search methods is provided in Appendix C.1.

4.2 Adaptive Monte Carlo Tree Sampling

The efficiency of tree-based exploration is often limited by the uniform distribution of samples across layers, which leads to suboptimal sampling performance at non-critical steps. A natural idea is to use the layer-wise reward variance $\text{Var}(Y^{(l)})$ to quantify the importance of each decision step: layers exhibiting higher variance presumably correspond to more consequential decisions, and thus deserve larger branching factors.

Since node values are recursively defined as empirical means of their children, $\text{Var}(Y^{(l)})$ inevitably entangles the uncertainty propagated from ancestor levels with the intrinsic stochasticity at level l—a high value may simply reflect accumulated dispersion from earlier layers rather than critical decisions at the current depth. To formally characterize and disentangle these two components, we establish the following variance decomposition:

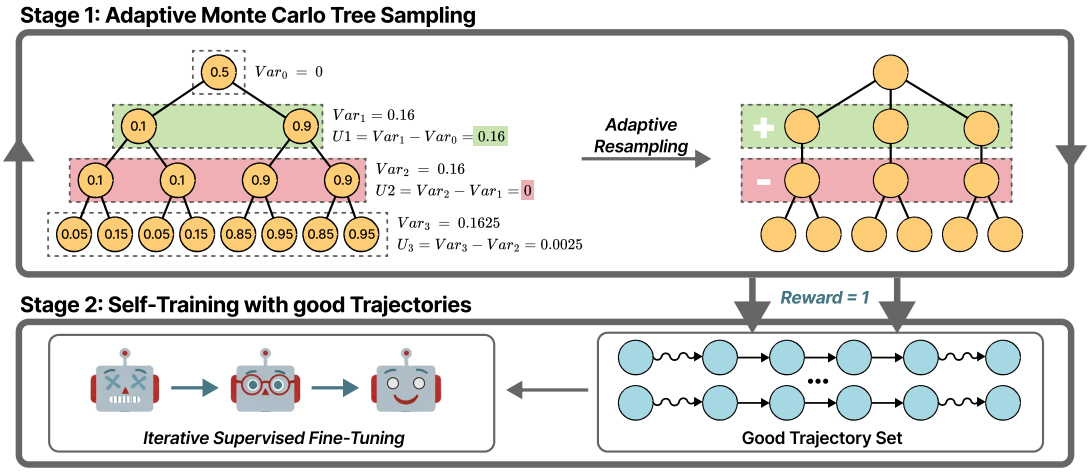

Figure 1: Overall workflow of A^4: In Stage 1, an initial Monte Carlo Tree is constructed without any hierarchical prior, where node values represent the conditional expectation of the final reward. The reward variance within each layer is computed, and the uncertainty of each layer is defined as the incremental variance Lv_l relative to the previous layer. Because larger Lv_l indicates greater sensitivity of the final outcome to decisions made at that depth, the sampling policy is adaptively updated to allocate more budget to layers with higher uncertainty. Consequently, the first layer (green box) receives increased sampling, whereas the second layer (red box) is assigned fewer samples. In Stage 2, the successful full trajectories obtained under the updated sampling strategy are collected and used for self-training, thereby improving the policy for subsequent iterations.

Theorem 4.1 (Separation of Layer-wise Variance)**.** *In an MCTS where each node value is defined as the empirical mean of its children,* $\mathrm{Var}(Y^{(l+1)})$ *can be decomposed into a parent-propagated term and a layer-intrinsic term, where the parent-propagated term equals* $\mathrm{Var}(Y^{(l)})$.

Proof Sketch. Since parent node values are averages of their children, the layer-wise mean is preserved, i.e., $\mathbb{E}[Y^{(l+1)}] = \mathbb{E}[Y^{(l)}]$. Writing $Y^{(l+1)} - \mathbb{E}[Y^{(l+1)}] = (Y^{(l+1)} - Y^{(l)}) + (Y^{(l)} - \mathbb{E}[Y^{(l)}])$ and averaging squared deviations separates $\mathrm{Var}(Y^{(l+1)})$ into a parent-propagated term and a residual term, where the parent-propagated term reduces to $\mathrm{Var}(Y^{(l)})$. See Appendix B.2 for the full derivation. □

Theorem 4.1 motivates the *incremental variance* $\mathrm{Lv}_l = \mathrm{Var}(Y^{(l)}) - \mathrm{Var}(Y^{(l-1)})$, which strips away the parent-propagated component and isolates the uncertainty genuinely introduced at level l. We adopt Lv_l as the importance signal and update the branching factors as $B_l^{(t+1)} = B_l^{(t)}[1 + \lambda(r_l - 1)]$, where $\lambda \in (0, 1]$ is a smoothing factor and $r_l = L \cdot \mathrm{Lv}_l / \sum_{l'} \mathrm{Lv}_{l'}$ is the normalized importance ratio satisfying $\sum_l r_l = L$. We show in Theorem B.1 that this update rule approximately preserves the total sampling budget across iterations, with relative deviation bounded by $O(\lambda^2)$.

AMCTS follows a two-pass protocol. First, a diagnostic pass collects rewards to estimate the Lv_l. Second, an **adaptive pass** executes search using the updated $\{B_l^{(t+1)}\}$, "widening" the tree at critical layers to maximize discovery of successful trajectories for the training buffer $\mathcal{D}_{\text{train}} = \{\tau \mid r(\tau) = 1\}$. The agent is updated on this set using a standard policy objective: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta; \mathcal{D}_{\text{train}})$, where η is the learning rate.

5 Experiments

In this section, we present experimental results on three interactive environments to demonstrate the effectiveness of A^4. Considering that distilling expert models requires substantial time and resource investment, we focus on conducting all following experiments via self-sampling, in scenarios where expert trajectories are unavailable.

5.1 Agentic Environments

Following previous work (Xi et al., 2024; Prasad et al., 2024; Yuan et al., 2025), we conduct experiments on three types of representative interactive environments: **WebShop** (Yao et al., 2022a), an interactive web environment for online shopping with 12k instructions. Agents perform search and click actions over one million real products from Amazon.com; **ScienceWorld** (Wang et al., 2022a), a scientific, text-based environment designed to evaluate agents' scientific reasoning abilities. It includes 30 types of scientific tasks at the standard elementary science curriculum level; and

TextCraft (Prasad et al., 2024), a text-based environment focusing on Minecraft item crafting trees.

5.2 Experiment Settings

We conduct our experiments on AgentGym (Xi et al., 2024), an interactive platform that includes diverse agent environments. We respectively sampled 300, 200, and 200 trajectories from the three tasks as sampling data. These tasks cover different forms of agent interaction, providing a comprehensive testbed for evaluating our method. For each task, we performed three iterations of iterative sampling, with the initial sampling count K set to 8, the sampling update parameter λ set to 0.2, and the maximum sampling depth set to 20. Correct trajectories were finally filtered based on environment scores. Each task in these three environments provides a task-specific target and a list of action commands generated by a tree structure. All three environments can be formally described as partially observable Markov decision processes, where observations are only available at the final step of each episode. Agents receive rewards according to the degree to which they achieve their respective task objectives, with final rewards ranging from 0 to 1. The test set sizes for the three tasks are 200, 200, and 100, respectively. At the same time, detailed training settings are provided in Appendix A.2.

5.3 Baselines

We compare our method against two categories of baselines: (1) **Agent-oriented LLMs**, which serve as strong off-the-shelf baselines, including AgentLM (7B and 13B) (Zeng et al., 2024) and Agent-FLAN (Chen et al., 2024); and (2) **Fine-tuning strategies** applied to a unified Llama-3.1-8B-Instruct (Dubey et al., 2024) backbone. This unified backbone enables a controlled comparison across different fine-tuning strategies. The latter category includes Best-of-N sampling, Rejection Sampling Fine-Tuning (RFT) (Yuan et al., 2023), ETO (Song et al., 2024b), standard MCTS combined with RFT, and Process Reward Model (PRM) (Lightman et al., 2024) guided fine-tuning. Detailed descriptions and configuration settings for all baseline methods are provided in Appendix A.1.

5.4 Evaluation Metrics

Following Yuan et al. (2025), we adopt different evaluation metrics tailored to each environment. Specifically, for SciWorld and WebShop, the average final reward achieved by the agent is used as the primary metric, reflecting overall task performance across episodes. In contrast, for TextCraft, we measure performance using the success rate, which captures the proportion of episodes in which the agent successfully completes the given task. For consistency across all three environments, we cap the maximum number of interaction rounds per episode at 20, ensuring a fair comparison of agent efficiency and effectiveness. Detailed evaluation settings are provided in Appendix A.2.

5.5 Main Result

Table 1 presents the performance of various methods across three interactive environments. Among agent-oriented LLMs, AgentLM benefits from scale but struggles with multi-step reasoning, while Agent-FLAN achieves moderate improvements via instruction-tuned reasoning. The base Llama-3.1-8B-Instruct model performs poorly, highlighting the limitations of standard pretraining data for complex interactive tasks. Best-of-N slightly improves WebShop but degrades SciWorld and TextCraft due to selection of erroneous trajectories. RFT leverages successful trajectories but is limited by trajectory-level rewards, whereas ETO uses pairwise rewards to partially mitigate this issue. MCTS + RFT expands exploration via tree search but uniform branching limits its focus on critical actions. PRM provides finer-grained step-level credit assignment, yet still underperforms our method as it lacks adaptive budget allocation across decision steps. We additionally include Ours (budget), an ablation that uses AMCTS but constrains training data to match MCTS + RFT, to verify that gains stem from sampling quality rather than data quantity (details in Appendix C.2). Our full method achieves the best overall performance, as action-level adaptive sampling focuses on critical decisions and generates higher-quality trajectories, providing stable, fine-grained feedback. Gains on SciWorld are smaller because self-sampled trajectories are constrained by the base model, which struggles with the environment's reasoning complexity.To better understand the computational overhead introduced by adaptive sampling, we further compare the resource consumption of AMCTS and standard MCTS on the WebShop environment. Both methods are evaluated on 300 tasks using a maximum sampling depth of 20 and an initial sampling factor of 8, with identical hardware configurations. Table 2 shows that AMCTS improves trajectory quality with limited extra cost.

Table 1: Performance Comparison on Three Interactive Agent Environments.

Method	WebShop	SciWorld	TextCraft	Average
Agent-oriented LLMs				
AgentLM-7B (Zeng et al., 2024)	36.50	2.75	0.00	13.08
AgentLM-13B (Zeng et al., 2024)	39.50	10.68	4.00	18.06
Agent-FLAN (Chen et al., 2024)	40.35	**28.64**	16.00	28.33
Fine-tuning strategies				
Llama-3.1-8B-Instruct (Dubey et al., 2024)	17.64	8.85	14.47	13.65
w/ Best-of-N	26.92	6.64	5.00	12.85
w/ RFT	40.42	11.03	19.00	23.48
w/ ETO (Song et al., 2024b)	41.37	10.06	31.00	27.51
w/ MCTS + RTF	40.30	9.80	17.00	22.36
w/ MCTS + RTF + PRM	39.84	8.37	20.00	22.73
w/ Ours (budget)	42.13	11.56	23.00	25.56
w/ Ours	**45.44**	13.83	**37.00**	**31.65**

Table 2: Computational cost and estimated carbon emissions of AMCTS and MCTS on 300 WebShop tasks over three iterations, with a maximum tree depth of 20 and an initial sampling factor of 8. Both methods run the same number of iterations; MCTS uses uniform branching throughout, while AMCTS adaptively reallocates the sampling budget after each iteration. Carbon emissions are estimated assuming 2 $\times$ NVIDIA H200 GPUs (700 W each) and a carbon intensity of 0.475 kg CO_2/kWh.

	MCTS	AMCTS
Hardware	2 $\times$ NVIDIA H200	2 $\times$ NVIDIA H200
GPU Hour	6.28 h	6.58 h (+4.77%)
Total Sampling Steps	30,331	31,593 (+4.16%)
Carbon Emission (kg CO_2 eq)	4.18	4.38 (+4.78%)

6 Further Analysis

In this section, we conduct a comprehensive evaluation of our approach across multiple dimensions. We analyze the effectiveness of multi-task versus single-task training, investigate the influence of model scale on performance, and examine the sensitivity of the adaptive coefficient λ in balancing sampling stability and adaptability. We further justify the use of variance-based action importance for efficient trajectory exploration and track the evolution of high-quality trajectory collection across iterations. These analyses collectively highlight the advantages of our method in enhancing model performance efficiency in interactive environments.

6.1 Multi-task Training *vs.* Single-task Training

To validate the role of A^4 in multi-task cooperative evolution, we compare multi-task training with single-task training. In the multi-task setting, trajectories from different environments are jointly aggregated to optimize a single policy, whereas single-task training uses task-specific data independently collected by AMCTS. As shown in the left part of Figure 2, multi-task training generally provides a more effective framework for language agents in interactive environments, though its benefits vary across tasks. WebShop mainly emphasizes intent understanding and product selection from natural language descriptions, with limited reliance on complex interactions. Thus, single-task training already achieves strong performance, and heterogeneous multi-task data brings no additional gain, causing a marginal drop from 45.82 to 45.44. In contrast, SciWorld and TextCraft share interaction dynamics and sequential decision-making structures. Mixing data across them helps the agent learn transferable behavioral patterns and strategic reasoning skills, improving generalization, exploration efficiency, long-horizon dependency handling, and overall performance compared with isolated single-task training.

6.2 Impact of Model Scale

To further demonstrate that our approach generalizes across model architectures and scales, we conduct additional experiments on the WebShop task using the Qwen3 family. As shown in the right part of Figure 2, A^4 consistently outperforms the origin

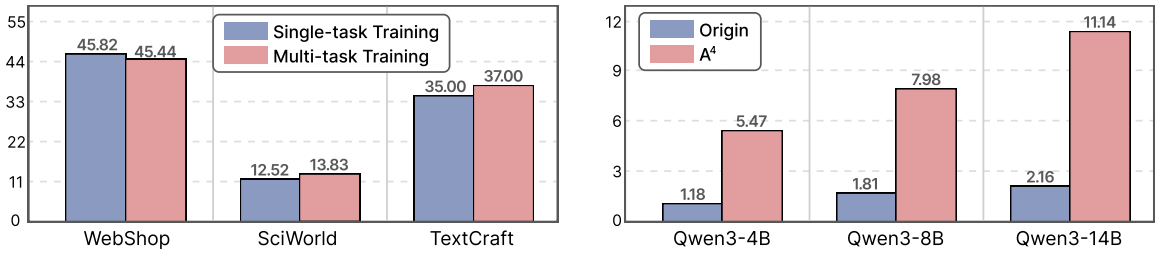

Figure 2: The left part shows the performance comparison on three iterative environments between single-task and multi-task training, where the multi-task setting jointly trains a single model on mixed data collected from all tasks. The right part shows the performance comparison of A^4 on WebShop using Qwen models at different scales.

Table 3: Effect of the adaptive coefficient λ on sampling behavior and performance.

λ	WebShop		SciWorld		TextCraft	
	Good Traj. Num.	Results	Good Traj. Num.	Results	Good Traj. Num.	Results
0.0	2906	41.12	813	11.24	1087	35.00
0.2	3531	45.44	1016	13.83	1231	37.00
0.5	3213	43.68	775	11.36	1074	35.00

baseline across Qwen3 models of different sizes. Notably, the performance gain becomes more pronounced as the model size increases, suggesting that larger models are better able to leverage the benefits of our algorithm. These results indicate that our method is not tied to a specific backbone architecture and remains effective across different model families and parameter scales, demonstrating strong robustness and scalability.

6.3 Sensitivity Analysis of the Adaptive Coefficient λ

We further analyze the effect of the hyperparameter λ on performance and trajectory quality across WebShop, TextCraft, and SciWorld. The results highlight that λ plays a key role in balancing sampling stability and adaptability. At $\lambda = 0$, the method reduces to a standard combination of MCTS and RFT, where hierarchical differences between iterations are ignored. As a result, trajectory refinement is limited, leading to fewer high-quality trajectories and lower environment scores. Increasing λ to 0.2 allows the sampling policy to be updated moderately after each iteration. This enables the algorithm to progressively incorporate high-quality trajectories while maintaining stability, which results in a notable increase in both trajectory quality and environment scores across all tasks. Further increasing λ to 0.5, however, causes overly aggressive updates, introducing instability in the sampling process. This large deviation from the assumption in B.1 that λ should remain small leads to a decline in both trajectory quality and final model performance. Overall, these results demonstrate that selecting an appropriate λ is crucial for effective hierarchical sampling, with moderate values striking the best balance between stability and adaptability.

6.4 Why Variance-based Action Importance

Consider the task shown in Figure 3, where the model is given the instruction to identify dual-band streaming media players that feature quad-core processors and are priced below $350. In the initial step, which involves searching for products, the trajectories retrieved by the model are largely similar, indicating that this action has a relatively small impact on the final outcome. This low variability suggests that excessive sampling at this stage would be inefficient, as most trajectories provide redundant information. Consequently, AMCTS assigns fewer samples to this step compared to standard tree sampling approaches, conserving computational resources for more critical decision points. In the subsequent steps, such as clicking on a specific product, evaluating its specifications, and making the final purchase decision, the trajectories exhibit significant variability and exert a strong influence on the overall result. Errors or suboptimal choices in these steps have a high likelihood of leading to failed task completion. By quantifying the layer-wise contribution of each action through variance analysis, AMCTS dynamically allocates more sampling resources to these high-impact steps, ensuring

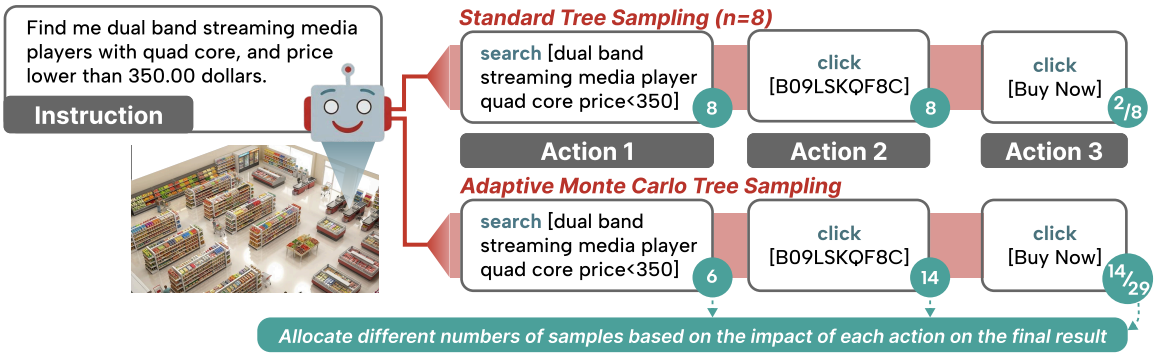

Figure 3: Case study of variance-based dynamic action sampling versus conventional tree search.

Table 4: Number of good trajectories and corresponding performance across iterations.

Iteration	WebShop		SciWorld		TextCraft	
	Good Traj. Num.	Results	Good Traj. Num.	Results	Good Traj. Num.	Results
# Iter 1	2824	40.30	806	8.85	1020	17.00
# Iter 2	3050	41.00	881	9.56	1141	23.00
# Iter 3	3531	45.44	1016	13.83	1231	37.00

that diverse candidate trajectories are explored thoroughly. This adaptive allocation enables the model to discover a wider range of successful action sequences, producing high-quality trajectories that reflect meaningful differences in decision-making. Overall, this mechanism demonstrates how AM-CTS can effectively balance exploration and exploitation by focusing on actions that matter most, leading to more robust and efficient agent behavior in complex, multi-step environments.

6.5 Evolution of Good Trajectory Collection Across Iterations

Table 4 presents the iterative performance of our algorithm across three distinct environments: webshop, sciworld, and textcraft. For each iteration, we report both the environment score and the number of high-quality trajectories generated. A clear trend emerges across all tasks: as iterations progress, both the environment scores and the number of successful trajectories consistently increase. In webshop, the environment score rises from 40.30 in the first iteration to 45.44 in the third, accompanied by an increase in high-quality trajectories from 2824 to 3531. Similar improvements are observed in sciworld and textcraft, with notable gains in both metrics by the third iteration. These results indicate that the algorithm effectively leverages information from previous sampling steps to assess the relative importance of different actions. Actions that exhibit greater influence on the outcome are progressively allocated more sampling resources, while less critical steps receive proportionally fewer samples. This adaptive allocation allows the model to explore a broader range of promising trajectories, leading to higher-quality paths and improved overall performance. Consequently, the observed iterative gains demonstrate that our approach not only identifies high-impact actions but also systematically enhances decision-making efficiency, resulting in a more thorough and strategic exploration of the decision space over time.

7 Conclusion

We introduce A^4, an action-level evolutionary framework for training LLM-based agents in long-horizon interactive environments. By modeling decision-making as a tree-structured process, A^4 moves beyond trajectory-level supervision and provides finer-grained feedback on intermediate actions. A^4 employs Adaptive Monte Carlo Tree Sampling, which uses layer-wise variance to guide adaptive sampling across decision steps. This allows sampling resources to focus on actions that have a greater impact on the final outcome, improving the quality of collected trajectories under a fixed budget. Experiments on multiple interactive environments show that A^4 consistently improves performance across tasks and model scales. Overall, this work demonstrates that variance-based, action-level sampling provides an effective direction for improving data collection and training in interactive LLM agents.

Limitations

While our work demonstrates the effectiveness of A^4 in improving trajectory quality and performance, it has some limitations. First, due to computational and resource constraints, we only consider scenarios where expert trajectories are unavailable, leaving open the question of how A^4 might perform with high-quality expert demonstrations. Second, our experiments are conducted in three interactive environments, which, although diverse, may not cover all task types or domains. Third, although AMCTS is a general sampling strategy that could theoretically be applied to any decomposable task, in this work, we only evaluate it in single-agent, multi-stage settings and have not validated it in multi-agent scenarios.

Ethical Considerations

This work aims to improve the training efficiency of LLM-based agents by adaptively allocating exploration budgets during trajectory collection. While our method does not introduce new data sources or human-facing deployment settings, more efficient agent training may amplify both beneficial and harmful behaviors learned from existing environments. In particular, agents optimized through self-sampling may inherit biases from benchmark tasks, reward functions, or model-generated trajectories. We therefore emphasize that A^4 should be applied with careful reward design, task-specific safety constraints, and human oversight when used in real-world interactive scenarios. Our experiments are conducted only on standard research benchmarks, and no private user data or personally identifiable information is used.

References

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9354–9366. Association for Computational Linguistics.

Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ning Gao, Xiuhui Zhang, Xingyu Jiang, Mukang You, Mohan Zhang, and Yue Deng. 2026. RF-Agent: Automated Reward Function Design via Language Agent Tree Search. *arXiv preprint arXiv:2602.23876*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Minki Kang, Jongwon Jeong, Seanie Lee, Jaewoong Cho, and Sung Ju Hwang. 2025. Distilling llm agent into small models with retrieval and code tools. *arXiv preprint arXiv:2505.17612*.

Kezhao Lai, Yutao Lai, and Hai-Lin Liu. 2026. Beyond the Node: Clade-level Selection for Efficient MCTS in Automatic Heuristic Design. *arXiv preprint arXiv:2602.00549*.

Li-Cheng Lan, Andrew Bai, Minhao Cheng, Cho-Jui Hsieh, and Tianyi Zhou. 2025. Exploring expert failures improves llm agent tuning. *arXiv preprint arXiv:2504.13145*.

Joshua Ong Jun Leang, Yu Zhao, Mihaela CÄ Stoian, Wenda Li, Shay B Cohen, and Eleonora Giunchiglia. 2026. Can I Have Your Order? Monte-Carlo Tree Search for Slot Filling Ordering in Diffusion Language Models. *arXiv preprint arXiv:2602.12586*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Saunders. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Muhan Lin, Shuyang Shi, Yue Guo, Vaishnav Tadiparthi, Behdad Chalaki, Ehsan Moradi Pari, Simon Stepputtis, Woojun Kim, Joseph Campbell, and Katia Sycara. 2025. Speaking the language of teamwork: Llm-guided credit assignment in multi-agent reinforcement learning. *arXiv preprint arXiv:2502.03723*.

Tao Liu, Jiafan Lu, Bohan Yu, Pengcheng Wu, Liu Haixin, Guoyu Xu, Li Xiangheng, Lixiao Li, Jiaming Hou, Zhao Shijun, and 1 others. 2026. IESR: Efficient MCTS-Based Modular Reasoning for Text-to-SQL with Large Language Models. *arXiv preprint arXiv:2602.05385*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Hao Lu, Haoyuan Huang, Yulin Zhou, Chen Li, and Ningxin Zhu. 2026. Empirical-MCTS: Continuous Agent Evolution via Dual-Experience Monte Carlo Tree Search. *arXiv preprint arXiv:2602.04248*.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, and 1 others. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.

Zhizhao Luo, Zhaojing Luo, Meihui Zhang, and Rui Mao. 2026. TabTracer: Monte Carlo Tree Search for Complex Table Reasoning with Large Language Models. *arXiv preprint arXiv:2602.14089*.

Yuanjie Lyu, Chengyu Wang, Jun Huang, and Tong Xu. 2025. From correction to mastery: Reinforced distillation of large language model agents. *arXiv preprint arXiv:2509.14257*.

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*. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024a. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141. Association for Computational Linguistics.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024b. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600. Association for Computational Linguistics.

Zeen Song, Zihao Ma, Wenwen Qiang, Changwen Zheng, and Gang Hua. 2026. Adaptive Uncertainty-Aware Tree Search for Robust Reasoning. *arXiv preprint arXiv:2602.06493*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022a. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298. Association for Computational Linguistics.

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

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, and 1 others. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Yajie Yang, Wenxiang Chen, Zhihao Zhang, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, and 1 others. 2025. AgentPRM: Process Reward Models for LLM Agents via Step-Wise Promise and Progress. *arXiv preprint arXiv:2511.08325*.

Xubang Xiong, Raymond Chi-Wing Wong, and Yuanfeng Song. 2026. Monte Carlo Tree Search with Reasoning Path Refinement for Small Language Models in Conversational Text-to-NoSQL. *arXiv preprint arXiv:2602.12574*.

Shunyu Yao, Howard Chen, John Yang, and Karthik R Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*.

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. 2022b. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077. Association for Computational Linguistics.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. *Advances in neural information processing systems*, 36:31967–31987.

A Experiment Details

A.1 Details of Baseline Methods

We compare A^4 against two broad categories of baselines: **agent-oriented LLMs** and **fine-tuning strategies** based on a common backbone.

Agent-oriented LLMs. These models are specifically designed or tuned for agentic behaviors. They serve as strong off-the-shelf agent baselines without requiring task-specific adaptation:

- **AgentLM-7B and AgentLM-13B** (Zeng et al., 2024): Trained via large-scale agent tuning on multi-task interactive data.
- **Agent-FLAN** (Chen et al., 2024): Further enhances instruction-following and decision-making abilities through agent-centric instruction tuning.

Fine-Tuning Strategies. For a fair comparison of optimization approaches, we adopt Llama-3.1-8B-Instruct (Dubey et al., 2024) as a unified backbone and apply the following strategies:

- **Best-of-N**: Selects the highest-reward trajectory among N sampled rollouts from the base agent; we set $N = 10$ in all experiments.
- **RFT (Rejection Sampling Fine-Tuning)** (Yuan et al., 2023): Augments the training data with successful trajectories generated by the agent itself and fine-tunes the model on the expanded dataset.
- **ETO** (Song et al., 2024b): First performs supervised fine-tuning via behavioral cloning on expert demonstrations, followed by preference optimization using contrastive successful and failed trajectories.
- **MCTS + RFT**: Applies standard Monte Carlo Tree Search with uniform branching factors to collect trajectories, then trains on the successful ones via RFT.
- **PRM** (Lightman et al., 2024): Applies a Process Reward Model to provide finer-grained credit assignment at the step level. We use GPT-4o as the PRM to score each intermediate step based on environment feedback and select higher-quality trajectories for fine-tuning.

A.2 Training and Evaluation Details

Hardware and Environment Setup. All experiments are conducted using eight H200-141GB GPUs. Services for different environments are deployed on separate ports of the same server to avoid conflicts and ensure smooth parallel execution.

Model Configuration. For the SFT phase, we employ the Llama-3.1-8B-instruct model. Mixed-precision training with dynamic loss scaling is used to improve memory efficiency and training speed. Gradient clipping is applied with a maximum norm of 1.0. Multi-task learning is supported by the framework, but all current experiments are conducted in a single-task setting.

Data Preparation. The training dataset is derived from the processed Alpaca English dataset. Sequences are packed to a maximum length of 16,384 tokens. We use a batch size of 1 per GPU with gradient accumulation over 4 steps, resulting in an effective batch size of 4. Data loading is performed with 4 worker threads per GPU. Sequence parallelism is supported through a dedicated sampler, with a standard fallback provided for non-parallel configurations.

Optimization and Training Schedule. We optimize the model using the AdamW optimizer (Loshchilov and Hutter, 2017) with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, zero weight decay, and gradient clipping at 1.0. The learning rate is initialized at 2×10^{-5}, with a linear warm-up over the first 3% of the total training epochs, followed by a cosine annealing schedule. The SFT training is conducted for 4 epochs. Gradient accumulation and mixed-precision training ensure efficient scaling across multiple GPUs.

Experimental Platform and Interactive Environments. We conduct our experiments on AgentGym (Xi et al., 2024), an interactive platform that includes diverse agent environments. Each task in these environments provides a task-specific target and a list of action commands generated by a tree structure. All three environments can be formally described as partially observable Markov decision processes, where observations are only available at the final step of each episode. Agents receive rewards according to the degree to which they achieve their respective task objectives, with final rewards ranging from 0 to 1.

Sampling Details. For the three tasks, we sampled 300, 200, and 200 trajectories as initial sampling data. Iterative sampling was performed for three iterations per task, with the initial sampling count $K = 8$, the sampling update parameter $\lambda = 0.2$, and the maximum sampling depth set to 20. Correct trajectories were filtered based on environment scores. The maximum number of tokens used for sampling is set to 8,192 (8k), and the sampling temperature is set to 1.0.

Evaluation Protocol. The test set sizes for the three tasks are 200, 200, and 100, respectively. Following Yuan et al. (2025), we adopt different evaluation metrics tailored to each environment. For SciWorld and WebShop, the primary metric is the average final reward achieved by the agent, reflecting overall task performance across episodes. For TextCraft, performance is measured using the success rate, capturing the proportion of episodes in which the agent successfully completes the task. For consistency across all three environments, the maximum number of interaction rounds per episode is capped at 20, and the evaluation maximum token length is set to 8,192 (8k).

Reproducibility. All experiments are repeated with three random seeds (42, 137, 2024) and we report the averaged results. The CuDNN benchmark is disabled for reproducibility. The training loop is managed by the TrainLoop class from Xtuner, which automatically handles epoch iterations, scheduler updates, and optimizer steps.

B Proofs

B.1 Notations

All notation used in the proofs below follows the definitions established in Table 5 (Section 3). In particular, $Y_i^{(l)}$ denotes the value of node $s_i^{(l)}$ and $Y_{i,j}^{(l+1)}$ denotes the value of the j-th child of node $s_i^{(l)}$.

B.2 Proof of Theorem 4.1

Proof. Let $\mu = \mathbb{E}[Y^{(l+1)}] = \mathbb{E}[Y^{(l)}]$. To evaluate the variance transition between layers, we decompose the total deviation into the local residual $\delta_{i,j}$ and the parent node deviation ϵ_i:

$$\delta_{i,j} = Y_{i,j}^{(l+1)} - Y_i^{(l)},$$
$$\epsilon_i = Y_i^{(l)} - \mu.$$

Algorithm 1 Adaptive Monte Carlo Tree Sampling

Require: Initial branching factors $B_1^{(0)}, \ldots, B_L^{(0)}$; Smoothing factor λ; Maximum iteration T
Ensure: Training set $\mathcal{D}_{\text{train}}$

```
D_train ← {}
for t = 0 to T − 1 do
    T ← {s^(0)} {Initialize search tree with root node}
    for l = 1 to L do
        for each node s_i^(l−1) in level l−1 of T do
            Sample B_l^(t) actions to expand s_i^(l−1)
        end for
    end for
    for each leaf trajectory τ in T do
        Evaluate terminal reward r(τ)
    end for
    for l = L to 0 do
        for each node s_i^(l) in level l of T do
            Y_i^(l) ← Mean{r(τ) | τ reachable from s_i^(l)}
        end for
    end for
    Var_0 ← 0
    for l = 1 to L do
        Var_l ← Variance{Y_i^(l) | s_i^(l) ∈ level l}
        Lv_l ← Var_l − Var_{l−1} {Incremental variance at step l}
    end for
    for l = 1 to L do
        L̄v ← (1/L) Σ_{k=1}^{L} Lv_k
        r_l ← Lv_l / L̄v
        B_l^(t+1) ← ⌊B_l^(t) [1 + λ(r_l − 1)]⌉
    end for
    D_train ← D_train ∪ {τ | r(τ) = 1}
end for
return D_train
```

The total deviation of a child node is $Y_{i,j}^{(l+1)} - \mu = \delta_{i,j} + \epsilon_i$. By the definition of empirical variance for layer $l+1$:

$$\mathrm{Var}(Y^{(l+1)}) = \mathrm{Var}(\delta) + \mathrm{Var}(\epsilon) + 2\mathrm{Cov}(\delta, \epsilon),$$

Specifically, the variance components are defined as follows: the branching variance is given by $\mathrm{Var}(\delta) = \frac{1}{n_l B_{l+1}^{(t)}} \sum_{i,j} \delta_{i,j}^2$, which captures the local exploration uncertainty; the node-level estimation bias is expressed as $\mathrm{Var}(\epsilon) = \frac{1}{n_l} \sum_i \epsilon_i^2$; and the interaction between these terms is quantified by

Table 5: Summary of notation used throughout this paper.

Component	Symbol	Definition
POMDP and Agent		
Spaces	$\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}$	Instruction, latent state, action, and observation spaces.
Transition	P	Stochastic transition kernel: $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$.
Reward func.	R	Reward function: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
Policy	π_θ	Agent policy parameterized by θ.
Depth	L	Trajectory length (equivalently, tree depth).
Step index	l	Decision step (layer) index, $l \in \{1, \ldots, L\}$.
Trajectory	τ	Complete trajectory $(a_1, o_1, \ldots, a_L, o_L)$.
Partial traj.	$\tau_{1:l}$	Action-observation pairs from step 1 to l.
Terminal reward	$r(\tau)$	Scalar reward for trajectory τ, $r(\tau) \in \mathbb{R}$.
Monte Carlo Search Tree (MCT)		
Search tree	$\mathcal{T} = (V, E)$	Layer-wise search tree of depth L.
Node	$s_i^{(l)}$	The agent's state at level l, encapsulating history $\tau_{1:l}$. Root: $s^{(0)}$.
Edge	(a_l, o_l)	Action-observation pair connecting level $l-1$ to level l.
Branching	$B_l^{(t)}$	Number of candidate actions sampled per node at level l at iteration t. Uniform within each level for tractability and symmetry.
Node count	n_l	Total nodes at level l: $n_l = \prod_{k=1}^{l} B_k^{(t)}$.
Leaf count	N_{traj}	Total complete trajectories: $N_{\text{traj}} = \prod_{l=1}^{L} B_l^{(t)}$.
Variance Decomposition and AMCTS		
Node value	$Y_i^{(l)}$	Expected return of the agent at state $s_i^{(l)}$, estimated as the mean reward over all leaf trajectories reachable from $s_i^{(l)}$.
Child value	$Y_{i,j}^{(l+1)}$	Value of the j-th child of node $s_i^{(l)}$.
Variance	Var_l	Empirical variance of node values $\{Y_i^{(l)}\}$ at level l.
Incremental var.	Lv_l	$\mathrm{Lv}_l = \mathrm{Var}_l - \mathrm{Var}_{l-1}$, the uncertainty introduced at step l.
Importance	r_l	$r_l = \mathrm{Lv}_l \,/\, \overline{\mathrm{Lv}}$, where $\overline{\mathrm{Lv}} = \frac{1}{L}\sum_k \mathrm{Lv}_k$.
Smoothing	λ	Adaptation rate of branching factor update.
Sampling budget	$N^{(t)}$	Total sampling budget at iteration t: $N^{(t)} = \prod_l B_l^{(t)}$.
Training set	$\mathcal{D}_{\text{train}}$	Successful trajectories for self-training: $\{\tau \mid r(\tau) = 1\}$.
Learning rate	η	Learning rate for policy optimization.

the covariance $\mathrm{Cov}(\delta, \epsilon) = \frac{1}{n_l B_{l+1}^{(t)}} \sum_i \epsilon_i (\sum_j \delta_{i,j})$. Since ϵ_i is constant for all children of node i, we can factor it out:

$$\sum_{i,j} \delta_{i,j}\epsilon_i = \sum_i \epsilon_i \left(\sum_j \delta_{i,j} \right) = 0,$$

where $\sum_j \delta_{i,j} = 0$ follows directly from $Y_i^{(l)}$ being the empirical mean of its children $\{Y_{i,j}^{(l+1)}\}_{j=1}^{B_{l+1}^{(t)}}$. For the second term, as ϵ_i^2 is independent of the child index j:

$$\frac{1}{n_l B_{l+1}^{(t)}} \sum_{i=1}^{n_l} \sum_{j=1}^{B_{l+1}^{(t)}} \epsilon_i^2 = \frac{1}{n_l} \sum_{i=1}^{n_l} \epsilon_i^2 = \mathrm{Var}(Y^{(l)}).$$

Combining these, we obtain the additive decomposition:

$$\mathrm{Var}(Y^{(l+1)}) = \mathrm{Var}(Y^{(l)}) + \mathrm{Lv}_{l+1},$$

where $\mathrm{Lv}_{l+1} = \frac{1}{n_l B_{l+1}^{(t)}} \sum_{i,j} \delta_{i,j}^2$ represents the average within-node variance, which completes the proof. □

B.3 Budget Invariance

Theorem B.1 (Budget Invariance)**.** *Under the update rule $B_l^{(t+1)} = B_l^{(t)}[1 + \lambda(r_l - 1)]$ with $\sum_l r_l = L$, the relative change in total sampling budget satisfies $\frac{N^{(t+1)} - N^{(t)}}{N^{(t)}} = O(\lambda^2)$.*

Proof. We analyze the stability of the total sampling budget $N^{(t)} = \prod_{l=1}^{L} B_l^{(t)}$ under the update rule $B_l^{(t+1)} = B_l^{(t)}[1 + \lambda(r_l - 1)]$. The ratio of the total budget between consecutive iterations is:

$$\frac{N^{(t+1)}}{N^{(t)}} = \prod_{l=1}^{L} \left[1 + \lambda(r_l - 1)\right].$$

Applying the natural logarithm to the product:

$$\ln \frac{N^{(t+1)}}{N^{(t)}} = \sum_{l=1}^{L} \ln \left[1 + \lambda(r_l - 1)\right].$$

Using the Taylor expansion $\ln(1 + x) = x - \frac{x^2}{2} +$

$O(x^3)$ for a small smoothing factor $\lambda \in (0, 1]$:

$$\ln \frac{N^{(t+1)}}{N^{(t)}} = \sum_{l=1}^{L} \left[\lambda(r_l - 1) - \frac{\lambda^2}{2}(r_l - 1)^2 + O(\lambda^3) \right]$$
$$= \lambda \sum_{l=1}^{L}(r_l - 1) - \frac{\lambda^2}{2} \sum_{l=1}^{L}(r_l - 1)^2 + O(\lambda^3).$$

Under the normalization constraint $\frac{1}{L}\sum_{l=1}^{L} r_l = 1$, the first-order term vanishes:

$$\sum_{l=1}^{L}(r_l - 1) = \sum_{l=1}^{L} r_l - L = L - L = 0.$$

Consequently, the log-ratio is dominated by the second-order term:

$$\ln \frac{N^{(t+1)}}{N^{(t)}} = -\frac{\lambda^2}{2} \sum_{l=1}^{L}(r_l - 1)^2 + O(\lambda^3) = O(\lambda^2).$$

Exponentiating both sides and using $e^x = 1 + x + O(x^2)$, the relative change in the sampling budget is:

$$\frac{N^{(t+1)} - N^{(t)}}{N^{(t)}} = \exp\left(O(\lambda^2)\right) - 1 = O(\lambda^2).$$

This demonstrates that the adaptive branching redistribution preserves the total sampling budget up to a second-order approximation in λ. □

C Adaptive Monte Carlo Tree Sampling

The AMCTS algorithm is illustrated in Algorithm 1. This algorithm effectively combines Monte Carlo Tree Search with adaptive strategies to explore large state spaces. In addition, AMCTS iteratively updates both the policy and the evaluation function, improving search efficiency and overall decision quality.

C.1 Comparison with Other Methods

Table 6 provides a systematic comparison of A^4 with representative methods that incorporate tree search or uncertainty-aware exploration. We highlight the key distinctions along five dimensions that are critical for effective self-training in interactive agent tasks.

Uncertainty Modeling. Most existing methods incorporate some form of uncertainty estimation, typically through UCB-based confidence bounds (Coulom, 2006) or model-predicted value uncertainty (Song et al., 2026; Gao et al., 2026). However, these methods model uncertainty at a single node or trajectory level. In contrast, A^4 performs *layer-wise variance decomposition* (Theorem 4.1), which disentangles the uncertainty contribution of each decision step from the cumulative variance. This decomposition is particularly important in the self-training setting, where the agent must identify which decision steps are most informative for improving its own policy, without access to external reward models or expert demonstrations.

Hierarchical Design and Dynamic Budget. While methods such as Clade-AHD (Lai et al., 2026) and Stage-MCTS (Xiong et al., 2026) adopt hierarchical structures, they either operate within local clades or fixed stage boundaries without adapting the sampling budget across levels. UATS (Song et al., 2026) adjusts exploration dynamically but only at the single-node level. Empirical-MCTS (Lu et al., 2026) partially supports dynamic budgets through experience replay but does not provide formal guarantees on budget preservation. A^4 uniquely combines a global hierarchical view with *provably budget-invariant* adaptive allocation (Theorem B.1), ensuring that increased exploration at critical steps does not inflate the total computational cost — a key practical requirement for iterative self-training where the sampling procedure is repeated across multiple iterations.

Action-level Analysis. Several methods provide action-level feedback, including RF-Agent (Gao et al., 2026) (via reward function design), TabTracer (Luo et al., 2026) (via trace-based reasoning), and IESR (Liu et al., 2026) (via modular SQL decomposition). However, these action-level analyses are task-specific and not designed for general-purpose self-training. A^4's variance-based importance measure ($r_l = \mathrm{Lv}_l/\overline{\mathrm{Lv}}$) is *task-agnostic*: it automatically identifies critical decision steps from the reward signal alone, requiring no domain-specific reward shaping or external verifiers.

Scope and Self-training Advantages. Perhaps the most significant distinction is in scope. Most compared methods operate at the level of single nodes, local clades, or individual stages, which limits their ability to capture cross-layer dependencies in multi-step decision-making. A^4 operates at a *multi-level, global* scope, jointly optimizing

Table 6: Comparison of different methods across uncertainty modeling, hierarchical design, budget allocation, and action-level analysis. △ refers to partially supported.

	Uncertainty Modeling	Hierarchical	Dynamic Budget	Variance Decomposition	Action-level Analysis	Scope
(Coulom, 2006)	✗	✗	✗	✗	✗	Single Node
Clade-AHD (Lai et al., 2026)	✓	✓	✓	✗	✗	Local Clade
RF-Agent (Gao et al., 2026)	✓	△	✗	✗	✓	Local + Global
UATS (Song et al., 2026)	✓	✓	✓	✗	✗	Single Node
TabTracer (Luo et al., 2026)	✗	✗	✗	✗	✓	Single Node
Stage-MCTS (Xiong et al., 2026)	✗	✓	✗	✗	✗	Intra-stage
Empirical-MCTS (Lu et al., 2026)	✓	✓	△	✗	✗	Local + Global Memory
McDiffuSE (Leang et al., 2026)	✓	✗	✗	✗	✓	Sequence-level
IESR (Liu et al., 2026)	✓	✓	△	✗	✓	Trajectory-level
A^4 **(Ours)**	✓	✓	✓	✓	✓	Multi-level / Global

sampling allocation across all decision steps. This global perspective is essential for self-training: by concentrating exploration on the most uncertain decision points, A^4 discovers a more diverse set of successful trajectories (as evidenced by the trajectory count improvements in Table 4), which in turn provides richer and more informative training data for iterative policy refinement.

C.2 Budget-Controlled Ablation Details

To disentangle the effect of *sampling quality* from *training quantity*, we introduce the "Ours (budget)" ablation. In this setting, we apply AMCTS to collect trajectories with adaptive branching, but then *downsample* the collected successful trajectories to match the exact number used by the MCTS + RFT baseline. Specifically, if MCTS + RFT yields M successful trajectories for training, we randomly sample M trajectories from the larger pool discovered by AMCTS. This ensures that both methods train on the same amount of data, and any performance difference can be attributed solely to the quality and diversity of the selected trajectories.

As shown in Table 1, Ours (budget) consistently outperforms MCTS + RFT across all three environments (42.13 vs. 40.30 on WebShop, 11.56 vs. 9.80 on SciWorld, 23.00 vs. 17.00 on TextCraft), confirming that the adaptive sampling strategy discovers higher-quality trajectories even when the training budget is held constant. The full A^4 method, which additionally benefits from a larger pool of diverse trajectories, achieves further gains, demonstrating that both sampling quality and quantity contribute to the final performance.

D Prompt Template

Prompt for WebShop

You are a strict task-oriented web shopping agent. IMPORTANT RULES (must follow strictly):
1. The user instruction is a set of HARD constraints. Treat all attributes (category, gender, usage, material, sleeve length, color, size, price, etc.) as AND conditions. A product is acceptable ONLY IF it satisfies ALL specified conditions.
2. NEVER click "Buy Now" unless you have verified that the product:

- Matches the correct product category (e.g., shorts vs tops vs jackets)
- Matches the correct gender (men vs women)
- Matches required functional attributes (e.g., activewear, quick-drying, moisture-wicking)
- Matches color EXACTLY as specified
- Matches size EXACTLY as specified
- Has a price within the specified limit

3. If ANY required condition is missing, unclear, or violated:

- Do NOT buy the product
- Go back, continue searching, or browse other results

4. Do NOT assume or guess missing attributes. If the product page does not clearly confirm a required attribute, treat it as NOT matching.
5. Use "Thought" to explicitly check each required condition one by one before buying. Example:

- Category: ✓
- Gender: ✓
- Color: × → Do NOT buy

6. "Buy Now" means FINAL DECISION. It should only be clicked when you are confident the product fully satisfies the instruction. Partial matches are NOT acceptable.
7. Ignore irrelevant signals such as rating availability or personal suspicion unless they violate the instruction.
Your goal is to maximize instruction compliance, not to finish quickly. It is better to keep searching than to buy an incorrect product.

Prompt for WebShop

You are an agent playing a constrained Minecraft crafting environment.IMPORTANT RULES YOU MUST FOLLOW STRICTLY:
1. Closed-world assumption:

- You may ONLY use the crafting commands explicitly provided by the user.
- You must NEVER invent, infer, modify, or generalize new crafting commands.
- If a command is not listed, it does NOT exist.

2. Ingredient availability:

- An item can be used ONLY IF it is currently in the inventory.
- If the environment or inventory explicitly reports that an item is unavailable, you must treat it as impossible to obtain unless a listed crafting command can produce it.

3. No circular or self-dependent crafting:

- You must NOT attempt to craft an item using itself as an ingredient.
- You must NOT attempt crafting actions that require ingredients you already know are unavailable.

4. Feasibility check before acting:

- Before taking an action, reason whether the goal is achievable given:

  (a) current inventory

  (b) available crafting commands

- If the goal item requires ingredients that cannot be obtained by any allowed command or get action, you must stop trying to craft.

5. Failure handling:

- If repeated observations confirm that required ingredients cannot be found or crafted, you should stop issuing crafting/get actions.
- In such cases, respond with a Thought explaining why the task is impossible under current constraints, and an Action indicating no-op or termination.

6. Output format:

- Always respond strictly using: Thought: (your reasoning)

  Action: (one valid action, or a termination/no-op action if impossible)

Your goal is to behave like a rational planner, not a guesser. Do not repeat failed actions unless the world state has changed.

Prompt for SciWorld

You are an agent operating in a strictly defined text-based simulation called Science World.
You MUST follow the rules below exactly. Violating them will cause failure.
1. Closed-world environment:

- The environment description (from "look around", "look at", "look in") is the ONLY source of truth.
- Do NOT assume the existence of any object, device, location, or property that is not explicitly mentioned.
- If an object is not listed in the current observation, you must assume it does NOT exist or is not accessible.

2. Action grammar is strict:

- You may ONLY use actions from the provided action list.
- Each action must exactly match one of the allowed formats.
- OBJ and LOC names must exactly match names that appear verbatim in the latest observation.
- Do NOT paraphrase, invent, or semantically modify action names or object names.

3. Objects vs locations:

- Locations (LOC) are rooms such as "bathroom", "workshop", "greenhouse", "hallway".
- Objects (OBJ) are items inside rooms (e.g., toilet, wire, thermometer, box).
- You may ONLY use "go to LOC" with a valid room name, never with an object.

4. Task instructions are procedural and mandatory:

- If the task says "First, focus on X. Next, focus on Y", you MUST execute these steps in order using "focus on OBJ".
- Do NOT skip, reorder, or reinterpret task steps.
- Do NOT invent additional goals (e.g., heating, boiling, powering) unless explicitly required.

5. Electrical and physical interactions:

- Only attempt "connect", "use", "activate", etc. if the action syntax is explicitly supported.
- If an action fails due to incorrect format, STOP and re-check the action list before trying again.
- Do NOT repeatedly try semantically identical actions after failure.

6. Failure awareness and recovery:

- System messages like "No known action matches that input" or "Its not clear how to..." are hard constraints.
- After such feedback, you must reconsider your understanding of:

  (a) available actions

  (b) valid objects

  (c) valid locations

- Revert to safe actions such as "look around", "task", or "inventory" if uncertain.

7. Reasoning style:

- Think like an executor, not a storyteller.
- Plan minimally: identify the next REQUIRED step, then execute exactly one valid action.

Response format (MANDATORY): Thought: (brief reasoning strictly grounded in the observation and task) Action: (one single valid action)