HierarchicalMCTS: From State-Level to Plan-Level Search for Enhanced Zero-Shot LLM Reasoning

Anonymous ACL submission

Abstract

Large Language Models (LLMs) exhibit limitations in complex, multi-step reasoning tasks. This paper introduces a framework that enhances LLM problem-solving by incorporating explicit planning via a modified Monte Carlo Tree Search (HierarchicalMCTS). Our approach decouples planning from execution, using modified MCTS to hierarchically search the space of complete reasoning plans, guided by evaluation agents that assess logical consistency and feasibility. We also explore the use of smaller LLMs for planning and larger ones for execution to improve efficiency. Experiments on six reasoning benchmarks show that HierarchialMCTS planning significantly improves accuracy, achieving a 24.18% average improvement over zero-shot Chain-of-Thought methods. Notably, the smaller-larger LLM configuration maintains 90.70% of the full performance while reducing computational cost by 73%. These findings highlight the importance of explicit, search-based planning for LLMs and suggest a path towards more robust and efficient reasoning systems for complex problem-solving. Codes are anonymously available at https://anonymous.4open.science/r/HierarchicalMCTS-9C0D.

Planning Type	Method	Comment	Model	GSM8K
No Planning	**Zero-Shot CoT** (Kojima et al., 2022)	Zero-Shot	Qwen2.5-7B-it Llama3.1-8B-it	80.89 57.32
Graph Planning	**SWAP** (Xiong et al., 2024)	Fine-Tuned	*Llama3-8B-it* *Mistral-7B-It*	*78.10* *54.00*
Auto Regressive Planning	**Plan-and-Solve** (Kojima et al., 2022)	Zero-Shot	*GPT-3*	*56.40*
	Least-to-Most (Zhou et al., 2023)	Task-Specific Prompt	*GPT-3*	*62.39* (1-shot)
	Tree-of-Thought (Yao et al., 2023)	Task-Specific Prompt	*GPT-4*	*90.00*
	Meta Reasoning (Gao et al., 2024)	Zero-Shot	*GPT-4* *GPT-3.5*	*92.10* *78.10*
	Arrange & Execute (Qiu et al., 2024)	Fine-Tuned	Qwen2-7B-it *Llama3-8B-it*	*82.11* *77.03*
	RAP (Hao et al., 2023) (Vanila MCTS)	4 Shots	*Llama-33B* Qwen2.5-7B-it Llama3.1-8B-it	*48.80* 83.09 75.06
Hierarchical Planning	**HierarchicalMCTS (Ours)**	Zero-Shot	Qwen2.5-7B-it Llama3.1-8B-it	90.14 77.28

Table 1: **Accuracy(%) Comparison of Planning Methods on GSM8K.** Our HierarchicalMCTS framework demonstrates substantial improvements in problem-solving accuracy through systematic plan optimization. In zero-shot settings, our approach consistently outperforms existing methods, achieving **+4.635%** higher accuracy than vanilla MCTS while costing only **2.84%** of its computational resources (detailed in Table 6). This significant efficiency gain, combined with improved accuracy, validates the effectiveness of our hierarchical planning strategy. More benchmark results are further documented in Table 2. Results from original publications are denoted in *italics*.

1 Introduction

Despite remarkable advances in natural language understanding tasks through auto-regressive generation (Brown et al., 2020; Chowdhery et al., 2023), current Large Language Models (LLMs) face inherent limitations that significantly impact their reasoning capabilities. The auto-regressive nature of these models, where each step depends solely on previous outputs, leads to three critical challenges: (1) **error propagation**, where initial mistakes cascade and amplify through the reasoning chain, (2) **logical inconsistency**, where subsequent deductions may contradict earlier steps due to the lack of global context, and (3) **myopic planning**, where models focus only on immediate, local transitions without maintaining a comprehensive view of the solution space. While increasingly sophisticated LLMs have been developed to mitigate these challenges in auto-regressive generation, even state-of-the-art (SOTA) models like GPT-4 (OpenAI-Team, 2024) continue to exhibit these fundamental limitations, particularly when confronted with tasks demanding complex multi-step reasoning and maintenance of logical consistency across extended deductive chains.

While recent approaches have attempted to address these challenges through techniques like Chain-of-Thought (CoT) prompting (Wei et al., 2022) and various task decomposition methods (Patel et al., 2022; Zhou et al., 2023; Zebaze et al.,

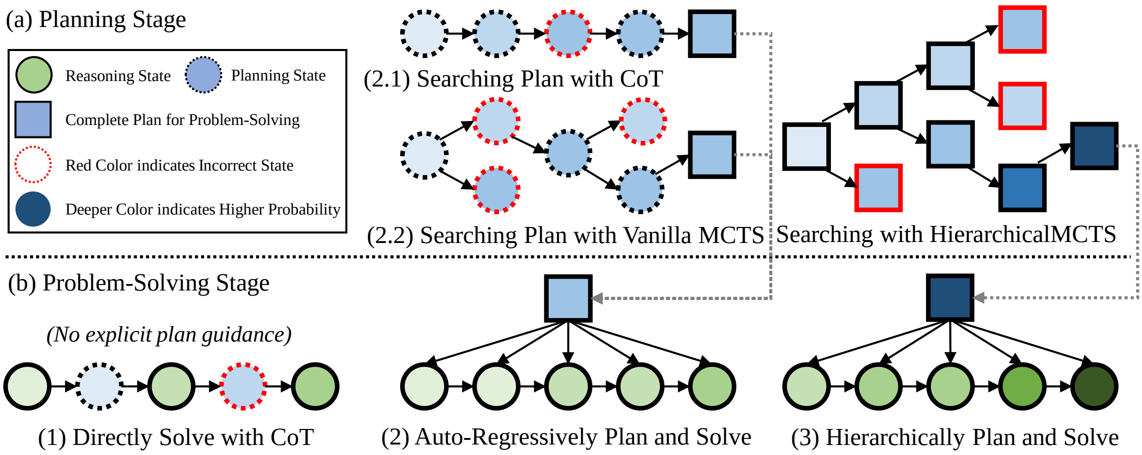

Figure 1: **A comparative analysis of three LLM problem-solving approaches:** (1) Direct CoT solving (Kojima et al., 2022), which interleaves planning and executing but inherently propagates errors through its reasoning chain; (2) Auto-regressive planning (Wang et al., 2023a; Hao et al., 2023), which separates planning from execution but generates potentially suboptimal plans due to its **state-by-state** search constraints of auto-regressive generation; and (3) Our HierarchicalMCTS framework, which systematically explores the complete plan space **plan-by-plan** through hierarchical search guided by LLM agent rewards. The darkness of shading indicates the joint probability of correct reasoning at each state. Experimental results demonstrate HierarchicalMCTS's superior performance (+24.18% vs. Zero-shot CoT; +4.635% vs. vanilla MCTS), achieved through more effective plan optimization via hierarchical exploration and global plan assessment.

2024), they remain constrained by the underlying auto-regressive architecture. Similarly, plan-and-solve frameworks (Wang et al., 2023a; Yao et al., 2023; Hao et al., 2023), though more structured, inherit these limitations as they still rely on step-by-step generation, as illustrated in Figure 1. The key issue lies in their inability to maintain global consistency and optimize across the entire solution space, leading to suboptimal plans and deteriorating performance in complex reasoning scenarios. This persistent challenge underscores **the critical need for external, systematic search methods that can overcome the internal limitations of auto-regressive generation**, thereby enabling the identification of globally optimal reasoning plans.

To address this challenge, two fundamental research questions guide our investigation:

RQ1: *Can external systematic search methods **effectively** identify optimal plans while avoiding LLMs' auto-regressive limitations?*

RQ2: *Can smaller LLMs **efficiently** guide larger LLMs in execution using optimally searched plans?*

Approach In this work, we address these research questions by introducing HierarchicalMCTS, a novel framework that fundamentally **shifts the search paradigm from individual reasoning steps *(state-level)* to complete reasoning plans *(plan-level)*.** Our key innovation lies in utilizing a modified MCTS where each search node represents an entire reasoning plan, enabling *global plan optimization* before execution. This approach stands in stark contrast to existing state-by-state planning methods that typically rely on auto-regressive generation, making them susceptible to error accumulation and suboptimal solutions. By treating complete plans as atomic search units, we can systematically optimize plan quality—a crucial determinant of reasoning performance—through hierarchical exploration. The search process is guided by specialized evaluation agents that assess both the logical consistency and practical feasibility of these complete plans, providing structured refinement through quantitative scores and qualitative feedback. This dual evaluation mechanism ensures that only coherent and feasible plans are explored in the search space, leading to more reliable reasoning outcomes. To rigorously validate our approach, we conduct three comprehensive experiments: (1) a comparative analysis of HierarchicalMCTS against standard methods and SOTA baselines, (2) an investigation of the relationship between plan quality and reasoning outcomes, and (3) an efficiency analysis of various model configurations. These experiments empirically demonstrate not only the superior performance of our hierarchical planning approach but also its practical viability and robustness across diverse reasoning tasks.

Findings Our comprehensive evaluation across six reasoning benchmarks demonstrates that HierarchicalMCTS significantly advances LLM problem-solving capabilities in both effectiveness and efficiency. In terms of effectiveness, the framework achieves a remarkable 24.18% average accuracy improvement over zero-shot CoT prompting, with particularly strong performance in complex arithmetic tasks. This substantial gain stems from two key innovations: the decoupling of planning from execution, and the systematic exploration of complete reasoning plans through hierarchical search. The method's superiority is evidenced by its consistent outperformance of structured CoT plan-and-solve baselines across all six benchmarks, validating the advantages of global plan optimization over incremental reasoning. Regarding efficiency, our analysis reveals a striking discovery: deploying smaller LLMs (1.5B parameters) for planning in conjunction with larger models (>70B parameters) for execution maintains 90.70% of full performance while reducing computational costs by 73%. This finding has significant implications for practical deployment, demonstrating that sophisticated reasoning capabilities can be achieved in resource-constrained settings.

Key contributions include:

1. **Effectiveness:** A novel HierarchicalMCTS framework that systematically optimizes complete reasoning plans through specialized evaluation agents, achieving superior accuracy (+24.18%) over existing methods and addressing fundamental limitations in current LLM reasoning approaches.
2. **Efficiency:** An innovative hybrid architecture combining smaller models for planning with larger models for execution, reducing computational costs by 73% while maintaining 90.70% of full performance, enabling practical deployment in resource-constrained environments.
3. **Theoretical Foundation:** A rigorous analysis of planning-reasoning decomposition in LLMs, demonstrating how hierarchical plan optimization through MCTS leads to more robust reasoning outcomes compared to interleaved approaches.
4. **Empirical Validation:** Comprehensive experimental results across six diverse reasoning benchmarks demonstrating consistent improvements: 24.18% average accuracy increase over zero-shot CoT and superior performance to structured baselines in all benchmarks.

2 Related Work

Prompting Techniques. Prompt-based methods emerged as a powerful technique to enhance LLM reasoning capabilities by providing explicit instructions and examples that guide model behavior. Chain-of-Thought (CoT) prompting (Wei et al., 2022) pioneered this approach by eliciting step-by-step reasoning, enabling LLMs to break down complex problems into manageable steps. This success inspired various extensions including systematic task decomposition (Patel et al., 2022) and least-to-most prompting (Zhou et al., 2023) that further structure the reasoning process. However, these methods face fundamental limitations due to their reliance on auto-regressive generation: they lack a global view of the solution, leading to potential logical inconsistencies (Wang et al., 2023b), error propagation (Gero et al., 2023), and reasoning failures particularly in extended sequences (Wei et al., 2022; Google-Team, 2023).

Planning Approaches. Recent frameworks attempt to address these limitations by separating planning from execution. Plan-and-Solve (Wang et al., 2023a) introduced explicit problem decomposition, while subsequent work focused on plan quality improvement through various strategies: Least-to-Most (Zhou et al., 2023) via stepwise decomposition, Meta Reasoning (Gao et al., 2024) through dynamic meta-information selection, and Arrange & Execute (Qiu et al., 2024) using fine-tuned planning models. Despite these advances, their reliance on auto-regressive generation leads to suboptimal solutions and error accumulation due to inherent state-by-state search limitations.

Search-Based Planning. Search-based methods have emerged as a promising direction for overcoming auto-regressive limitations. Tree-of-Thoughts (Yao et al., 2023) pioneered systematic plan exploration, while RAP (Hao et al., 2023) introduced MCTS for stepwise plan optimization. SWAP (Xiong et al., 2024) further advanced this through graph-based planning. Besides, MCTS itself has demonstrated remarkable success in complex decision spaces, particularly in gaming (Silver et al., 2016), through its balanced exploration-exploitation framework (Chaslot et al., 2008). Recent applications to LLM planning (OpenAI, 2024; Wang et al., 2024) show promise but face significant challenges: vast generation spaces complicate effective sampling, while evaluation costs limit computational scalability.

Research Gaps and Our Contribution. Current approaches face two key limitations: planning methods remain constrained by sequential reasoning, while traditional MCTS struggles with vast action spaces and computational costs. Our work addresses these challenges through two key innovations: (1) hierarchical plan-level search with specialized evaluation agents, and (2) efficient model scaling that leverages smaller models for search and larger models for execution. This represents a fundamental shift from state-level to plan-level optimization, enhancing reasoning capabilities while maintaining computational efficiency.

3 HierarchicalMCTS

Current LLM architectures face fundamental limitations in complex reasoning tasks due to their auto-regressive nature, particularly when global consistency is required. We introduce HierarchicalMCTS, a framework that enhances LLM reasoning through systematic optimization of complete reasoning plans via modified Monte Carlo Tree Search. By decoupling planning from execution and implementing specialized evaluation agents, our approach addresses core limitations in auto-regressive generation.

3.1 From State-Level to Plan-Level Search

The LLM problem-solving process can be modeled probabilistically: given a problem X and context $C_{problem}$, the objective is to generate a solution Y. Traditional CoT methods directly model this as $P(Y|X, C_{problem})$, combining planning and reasoning into a single step. This conflation, coupled with auto-regressive generation, leads to error propagation where early mistakes cascade through the solution process.

Plan-and-Solve methods (Wang et al., 2023a) attempt to separate planning and execution by decomposing context C into problem description $C_{problem}$ and plan C_{plan}:

$$P(Y|X, C) = P(Y|X, C_{problem}, C_{plan}) \quad (1)$$

However, these methods remain constrained by auto-regressive plan generation, inheriting CoT's limitations in maintaining global consistency and optimal plan identification.

We propose that a high-quality plan C_{plan} enables solution Y to be conditionally independent of problem X, implying the plan's critical role in problem-solving. This leads to our two-stage approach:

1. **Planning** ($P(C_{plan}|C_{problem})$): Systematically search and optimize reasoning steps through HierarchicalMCTS to generate a comprehensive plan.
2. **Execution** ($P(Y|X, C_{problem}, C_{plan})$): Generate solution guided by the optimized plan.

HierarchicalMCTS transforms traditional MCTS by operating on complete reasoning plans rather than individual steps. By treating plans as atomic search units and employing specialized evaluation agents for logical consistency and feasibility assessment, our approach enables systematic optimization toward globally optimal solutions while addressing error propagation and myopic planning limitations.

3.2 Plan-Level Search and Optimization

Within our probabilistic framework, the system begins with an LLM-generated base plan C_0 as the root node. For a given problem P, HierarchicalMCTS explores the solution space to discover an optimized plan C^* that maximizes the probability of correct solution generation Y, enabling global optimization before execution while efficiently navigating potential reasoning pathways.

Expansion. When reaching a leaf node (representing plan C_t), HierarchicalMCTS expands the search tree through structured plan refinement, generating complete variant plans rather than individual steps. The expansion process utilizes a plan refinement operator $Refine(C_t, \text{feedback}, M)$, which leverages both the LLM planner M and evaluation agent feedback to synthesize improved plans C_{t+1}. This process enables comprehensive optimization through iterative refinement - for instance, in mathematical problem-solving, an initial plan C_0 merely stating "Extract numerical values" might evolve through feedback into increasingly sophisticated strategies that incorporate relationship analysis, equation formulation, and systematic solution approaches. By maintaining focus on complete plan evolution rather than incremental modifications, this hierarchical refinement mechanism facilitates global optimization of reasoning strategies while preserving plan coherence and structural integrity throughout the search process.

Simulation and Reward. While traditional MCTS relies on stochastic rollouts for state evaluation, such an approach proves inadequate for assessing abstract reasoning plans. Our framework instead implements specialized evaluation agents, E, that provide both quantitative assessment and qualitative feedback. Two critical agents form the core of this evaluation system: a Logical Consistency Agent that examines plan coherence and identifies logical contradictions (e.g., "Step 3 contradicts Step 2"), and a Feasibility Agent that analyzes practical implementability and execution constraints (e.g., "Step 4 requires iterative solving but lacks termination conditions"). Both agents output normalized scores between 0 and 1, enabling systematic comparison and optimization of candidate plans while providing actionable insights for refinement.

These specialized agents generate both quantitative assessments ($score \in [0, 1]$) and qualitative feedback in the form of detailed critiques. To guide the MCTS search process, we employ a weighted reward function that synthesizes these evaluations: $\text{Reward}(C) = w_1 \cdot \text{LogicalConsistency}(C) + w_2 \cdot \text{Feasibility}(C)$ where weights w_1 and w_2 (with $w_1 + w_2 = 1$) reflect the relative importance of logical soundness and practical implementability. This dual-nature reward mechanism serves two critical functions: the numerical scores drive the quantitative optimization within MCTS, while the textual critiques inform the qualitative refinement of plans during the Expansion phase, ensuring a balanced approach to plan improvement.

Selection and Backpropagation. Node selection (choosing which plan to expand) uses the Upper Confidence Bound 1 algorithm (Auer, 2002):

$$\text{UCB1}(C) = Q(C) + C_{exp}\sqrt{\frac{\ln N_{\text{parent}}}{N_C}} \quad (2)$$

Here, $Q(node)$ is the average evaluation score of the plan at that node, $N(node)$ and $N(parent)$ denote visitation frequencies, and C is a constant that balances exploration and exploitation. This ensures that computational resources are focused on the most promising regions of the plan space. The reward signal, obtained from the evaluation agents, is then backpropagated up the MCTS tree, updating the value estimates of all nodes along the path from the root to the newly expanded node.

After evaluating a selected plan, the reward signal propagates upward through the tree, updating each node's value estimate via $Q(C) \leftarrow (Q(C) \cdot (N_C - 1) + \text{Reward}(C))/N_C$. This dynamic process ensures that promising plan variations receive increased attention while maintaining sufficient exploration of alternative approaches. The recursive nature of these updates gradually refines the search tree's value estimates, steering the algorithm toward optimal reasoning plans through iterative improvement and assessment.

The complete pseudocode for HierarchicalMCTS is provided in Appendix A.6.

4 Experiments

To rigorously evaluate our framework, we conduct two complementary experimental investigations: (1) a systematic evaluation of HierarchicalMCTS against standard baselines and state-of-the-art methods across diverse reasoning tasks, and (2) an in-depth analysis of efficiency trade-offs between different model configurations to identify optimal resource utilization strategies for practical deployment. Through these experiments, we aim to demonstrate both the effectiveness of hierarchical planning in enhancing LLM reasoning and its practical viability in resource-constrained settings.

4.1 Experimental Setup

Benchmark Selection. We evaluate our approach using a carefully curated set of benchmarks that assess two fundamental dimensions of LLM reasoning. For mathematical reasoning, we employ five complementary datasets: GSM8K (Cobbe et al., 2021) for multi-step problem solving, AddSub (Hosseini et al., 2014) for basic arithmetic operations, MultiArith (Roy and Roth, 2015) for complex numerical relationships, SVAMP (Patel et al., 2021) for structural variations, and SingleEq (Koncel-Kedziorski et al., 2015) for equation formulation. For commonsense reasoning, we include CommonsensQA (Talmor et al., 2019) to evaluate contextual understanding and knowledge application. This combination enables rigorous assessment of both structured mathematical thinking and flexible reasoning capabilities.

Model Selection. Our implementation uses two state-of-the-art language models: Qwen 2.5 (Yang et al., 2024) and Llama 3.1 (Llama-Team, 2024). All experiments are conducted on the SGLang platform (Zheng et al., 2024), with detailed protocols, code implementations, and configurations provided in Appendix A.1 to ensure reproducibility.

Baselines Selection. We compare our approach against three methodological categories: (1) Direct reasoning methods: Zero-shot CoT prompting (Ko-

Method Type	Benchmark Model	Addsub	CommonsensQA	GSM8K	MultiArith	SingleEq	SVAMP
Zero-Shot(ZS) CoT (Kojima et al., 2022)	Qwen2.5-7B-it	85.06	63.72	80.89	95.33	77.17	83.40
	Llama3.1-8B-it	28.61	63.80	57.32	38.17	39.76	27.00
	Avg.	56.84	63.76	69.10	66.75	58.47	55.20
CoT Plan (Wang et al., 2023a)	Qwen2.5-7B-it	87.59	78.62	88.84	98.33	93.70	91.90
	Llama3.1-8B-it	78.23	57.14	74.77	91.58	84.65	79.20
	Avg.	82.91	67.88	81.81	94.96	89.18	85.55
	Chagnes over ZS CoT	*26.07*	*4.12*	*12.71*	*28.21*	*30.71*	*30.35*
HierarchicalMCTS (Ours)	Qwen2.5-7B-it	88.10	79.20	90.14	98.67	92.91	92.90
	Llama3.1-8B-it	80.51	68.57	77.28	92.76	87.99	81.20
	Avg.	**84.31**	**73.88**	**83.71**	**95.72**	**90.45**	**87.05**
	Chagnes over ZS CoT	*27.47*	*10.12*	*14.61*	*28.97*	*31.98*	*31.85*

Table 2: **Accuracy(%) Comparison of Different Problem-solving Methods with LLM.** The Modified MCTS Plan consistently outperforms ZS CoT with an average improvement of 24.18%, and shows superior results compared to CoT Plan in all 6 benchmarks. Results suggest that decoupling planning from execution improves problem-solving accuracy and optimal plan searched by our approach can yield substantial performance gains. Best results are highlighted in bold.

jima et al., 2022) serves as our foundational baseline. (2) Plan-and-solve frameworks: Including vanilla CoT planning (Wang et al., 2023a), Meta Reasoning (Gao et al., 2024), and Arrange & Execute (Qiu et al., 2024). These methods separate planning from execution but remain constrained by auto-regressive generation. (3) Search-based methods: Including RAP (Hao et al., 2023), Tree-of-Thought (Yao et al., 2023), and SWAP (Xiong et al., 2024), which employ systematic exploration strategies. Our HierarchicalMCTS framework represents a novel extension of this category, focusing on effective and efficient plan optimization.

Fair Comparison Consideration. To ensure fair comparison, we exclude task-specific approaches like PromptAgent (Wang et al., 2024) that rely on specialized architectures. Given the challenge of fairly transferring task-specific and few-shot methods across benchmarks, we focus our comparative analysis on GSM8k (Cobbe et al., 2021), a widely adopted benchmark in the field. For methods with public implementations, we report performance from original publications; for zero-shot methods (CoT and CoT Plan), we follow official implementations and parameters as specified in recent work (Kong et al., 2024; Kojima et al., 2022; Hao et al., 2024).

4.2 RQ1: Effectiveness of HierarchicalMCTS

Methods Comparison. Table 1 provides a detailed comparison on the challenging GSM8K benchmark, which evaluates mathematical reasoning capabilities. Our HierarchialMCTS planning approach achieved accuracies of 90.14% with Qwen2.5-7B-it and 77.28% with Llama3.1-8B-it, surpassing all comparable methods using similar LLMs. The results highlight key limitations of existing approaches. For instance, Least-to-Most prompting, which relies on LLMs' sequential reasoning, failed with both Qwen and Llama models due to infinite loops, indicating inherent constraints in LLMs' native reasoning capabilities and the need for controlled search frameworks. Our approach showed notable improvements over vanilla MCTS implemented in RAP, which achieved 83.09% with Qwen2.5-7B-it and 75.06% with Llama3.1-8B-it. This improvement stems from our specialized evaluation agents and structured plan refinement techniques. While direct comparisons with Least-to-Most prompting and ToT are limited by their use of larger models (GPT-3 and GPT-4), our method achieves comparable or superior results using smaller models, advancing the optimization between model size and performance.

Benchmark-Wide Comparison. Table 2 demonstrates the effectiveness of our HierarchicalMCTS planning approach across all benchmarks. The method achieved an average accuracy improvement of 24.18% over zero-shot CoT, showing statistically significant performance gains. Our approach outperformed the CoT plan-and-solve baseline in all six benchmarks, with particularly strong results in mathematical reasoning tasks (AddSub, MultiArith, SingleEq, SVAMP), where maintaining logical consistency across multiple calculation steps is crucial. Notably, even in commonsense reasoning tasks (CommonsensQA), we observed substantial improvements, likely due to our method's ability to systematically decompose knowledge application into coherent steps. This consistent pattern of improvement suggests that HierarchicalMCTS

Model	Max Depth								Number of Rollouts					
	1	3	5	7	10	20	50	100	1	3	5	7	10	20
Llama3.1-8B-it	74.22	75.97	76.12	76.42	76.80	76.42	77.18	77.41	79.08	76.88	76.04	76.57	76.19	77.48
Qwen2.5-7B-it	87.64	88.17	88.02	88.40	88.32	88.48	88.55	88.78	89.01	89.76	89.08	89.23	90.14	89.92

Table 3: **Performance Comparison of Language Models**

Model	Evaluator	Addsub	CommonsensQA	GSM8K	MultiArith	SingleEq	SVAMP
Qwen2.5-7B-it	Feasibility	88.1	71.3	89.5	97.7	91.5	92.2
Qwen2.5-7B-it	Logical Consistency	86.6	70.9	89.2	97.2	91.5	91.4
Qwen2.5-7B-it	Combined (Ours)	88.1	79.2	90.1	98.7	92.9	92.9

Table 4: **Performance Cmparison of Different Evaluation Agents for MCTS.**

planning is especially valuable for tasks requiring precise, multi-step reasoning processes, where its structured approach to plan optimization can effectively guide complex problem-solving.

These findings advance LLM research in several ways. First, they provide compelling evidence that separating planning from execution significantly improves LLM reasoning capabilities. The substantial performance gains achieved by our HierarchialMCTS planning approach demonstrate that a dedicated planning phase, guided by hierarchical search, enhances the accuracy and reliability of LLM-based problem-solving. Second, our results underscore the critical importance of plan quality. The success of MCTS stems from its ability to explore and refine possible plans, guided by specialized evaluation agents that assess both logical consistency and feasibility. This highlights opportunities for further research into plan generation and evaluation methods.

Ablation Study. Table 3 shows that increasing search tree depth improves performance up to a point, suggesting initial search steps are more critical. The number of rollouts also impacts performance, with diminishing returns as the number increases. For Qwen2.5-7B-it, increasing rollouts from 1 to 10 improves accuracy from 89.01% to 90.14%, but further increasing to 20 only yields a marginal improvement to 89.92%. Table 4 shows that combined evaluation agents (feasibility and logical consistency) yield the best results. For instance, on GSM8K, the combined agent achieves 90.1% accuracy, compared to 89.5% (feasibility) and 89.2% (logical consistency) alone.

Our findings demonstrate that separating planning from execution and using HierarchicalMCTS to search for optimal plans significantly enhances LLM reasoning. The substantial performance gains highlight the importance of plan quality and the effectiveness of MCTS, guided by specialized evaluation agents.

4.3 RQ2: Efficiency of HierarchicalMCTS

Building on the findings of Section 4.2, which demonstrated the significant benefits of MCTS planning for LLMs, we investigate the crucial tradeoff between computational efficiency and performance. A key question emerges: *Can strategically combining small and large LLMs enhance both the efficiency and effectiveness of MCTS-based planning?*

To address this question, we implemented a heterogeneous model approach within the MCTS framework, utilizing smaller LLMs (Qwen2.5-1.5B-it and Gemma-2-2b-it (Gemma-Team, 2024)) for plan generation and a larger LLM (Qwen2.5-72B-it) for plan evaluation or execution. This approach was systematically evaluated across the benchmarks detailed in Section 4.1, examining various model size combinations for each role in the planning process.

Table 5 reveals compelling patterns across different model configurations. The **synergistic relationship between small and large models** emerges as a key finding: using a smaller LLM for planning with a larger LLM for execution yielded substantial performance gains across all datasets. The Qwen2.5-1.5B-it (planner) + Qwen2.5-72B-it (evaluator+executor) configuration achieved an average improvement of 23.87% compared to using Qwen2.5-1.5B-it alone, validating this complementary approach.

This heterogeneous approach also offers substantial **efficiency gains**. Using smaller LLMs for planning and evaluation with a larger LLM for execution (Qwen2.5-1.5B + Qwen2.5-1.5B + Qwen2.5-72B) reduced GPU time to 27% of that required when using large LLMs throughout, while main-

Planner Model	Evaluator Model	Benchmark Executor Model	Addsub	CommonsensQA	GSM8K	MultiArith	SingleEq	SVAMP	GPU Sec.	Eff. Ratio
Qwen2.5-1.5B-it	Qwen2.5-1.5B-it	Qwen2.5-1.5B-it	75.70	58.72	64.29	86.50	83.66	72.10	2914.4	2.322
Gemma2-2B-it	Gemma2-2B-it	Gemma2-2B-it	81.52	17.16	52.67	86.83	86.02	69.40	2893.6	2.064
Qwen2.5-1.5B-it	Qwen2.5-1.5B-it	Qwen2.5-72B-it	88.86	78.49	86.96	96.83	95.28	90.70	3481.6	**2.727**
	Qwen2.5-72B-it	Qwen2.5-1.5B-it	86.58	68.36	81.35	92.50	88.78	81.40	7735.2	1.061
	Qwen2.5-72B-it	Qwen2.5-72B-it	90.63	80.71	92.80	98.67	94.88	92.00	8844.0	1.156
Gemma2-2B-it	Gemma2-2B-it	Qwen2.5-72B-it	91.39	77.72	88.48	97.67	95.67	92.30	3335.2	**2.983**
	Qwen2.5-72B-it	Gemma2-2B-it	90.89	34.64	79.83	95.00	92.13	84.00	7793.6	1.039
	Qwen2.5-72B-it	Qwen2.5-72B-it	92.41	78.54	92.42	98.33	95.87	93.00	8899.2	1.138
Qwen2.5-72B-it	Qwen2.5-72B-it	Qwen2.5-72B-it	91.14	83.95	94.62	98.67	95.08	93.40	12888.0	0.821

Table 5: **Performance Comparison of Different LLMs for Planning and Execution.** Results shows that using smaller models for planning and larger models for execution with HierarchicalMCTS enhances efficiency. 'GPU Sec.' represents the total GPU time (in seconds) needed to complete the six benchmarks. 'Eff. Ratio' is calculated as the ratio of GPU seconds to the average accuracy of the six benchmarks, given by $\frac{\text{GPU Sec.}}{\text{Avg. Acc.} \times 100}$. The highest ratio, indicating better efficiency, is highlighted in bold.

Method	Model	GSM8K	GPU Sec.	Eff. Ratio
Vanila MCTS (RAP) (Hao et al., 2023)	Qwen2.5-7B-it	83.09	25612.0	0.324
	Llama3.1-8B-it	75.06	23296.0	0.322
HierarchicalMCTS (Ours)	Qwen2.5-7B-it	**90.14**	**546.4**	**16.497**
	Llama3.1-8B-it	**77.28**	**566.4**	**13.644**

Table 6: **Performance and Efficiency Comparison of Different MCTS Planning Methods.** Our Modified MCTS Plan outperforms Vanila MCTS (RAP) in its official implementation with only 2.84% of the GPU seconds required to complete GSM8K benchmark.

taining 90.70% of full performance (vs. 93.40%). This configuration also outperformed using only smaller LLMs (90.70% vs. 72.10%) with just a 19.46% increase in computational cost. The competitive performance of Gemma-2-2b-it, despite its smaller size, indicates that model architecture and training methodology significantly influence effectiveness beyond parameter count.

These findings demonstrate that strategic combinations of small and large LLMs can significantly enhance MCTS-based planning efficiency while maintaining high performance. This approach offers clear advantages in computational efficiency and resource optimization, particularly valuable for resource-constrained applications. Future research should explore techniques to optimize model combinations and develop methods to maintain planning quality while further reducing computational overhead.

5 Conclusion

This paper introduces HierarchicalMCTS, a framework that fundamentally advances LLM problem-solving by integrating hierarchical MCTS with specialized evaluation agents for systematic plan optimization. Our comprehensive experimental evaluation demonstrates significant improvements across diverse reasoning benchmarks, achieving an average accuracy gain of 24.18% over zero-shot CoT methods. The framework shows particular efficacy in complex arithmetic and commonsense reasoning tasks, where maintaining logical consistency across extended deductive chains is crucial.

Our investigation yields three insights that advance the field's understanding of LLM reasoning. First, explicit search-based planning outperforms implicit reasoning approaches, as evidenced by superior performance across all benchmarks. This finding challenges the assumption that increasingly LLM architectures alone can overcome fundamental limitations in complex reasoning tasks. Second, our results establish a strong correlation between plan quality and reasoning accuracy, validating the effectiveness of hierarchical optimization in generating robust solution strategies. Third, our hybrid architecture, which deploys smaller LLMs for planning and larger ones for execution, maintains 90.70% of full performance while reducing GPU time by 73%. This breakthrough in efficiency demonstrates a viable pathway for practical deployment in resource-constrained environments.

These findings open several promising research directions at the intersection of classical AI and modern language models. The success of combining systematic search techniques with LLMs suggests opportunities for integrating other traditional AI methods, particularly in areas requiring structured reasoning. Future work could explore adaptive evaluation strategies that dynamically adjust to problem complexity, automated plan repair mechanisms for handling execution failures, and extensions to more diverse reasoning scenarios. Due to space limitations, we place discussion in Appendix A.2. More broadly, our results advance the development of AI systems capable of reliable complex problem-solving through principled, hierarchical reasoning approaches.

Ethical Considerations

This research enhances LLM problem-solving via MCTS planning. While our benchmark datasets pose minimal direct ethical concerns, the potential impact of improved AI problem-solving necessitates broader ethical consideration.

Integrating MCTS introduces unique challenges. Unlike CoT methods, MCTS explores a wider range of plans, some potentially ethical yet logically sound. For instance, directly plan and executing a toxic action could be rejected by LLM with alignment to ethical principles. However, if the search objective is to find such a plan, MCTS may escape the LLM's ethical constraints. In this context, Mitigation strategies warrant investigation. Incorporating fairness constraints into MCTS, adversarial training for agents, and human-in-the-loop plan review could enhance ethical outcomes.

Limitations

While our HierarchialMCTS planning approach demonstrates significant improvements in LLM problem-solving capabilities, it is important to acknowledge several limitations of our current work.

Firstly, the computational cost of MCTS, particularly for larger language models, remains a significant challenge. Although we have shown that using smaller models for planning can mitigate this issue to some extent, further research is needed to optimize the efficiency of MCTS in the context of LLMs. Future work could explore pruning techniques or more sophisticated heuristics to reduce the search space without compromising plan quality.

Secondly, our study primarily focused on a specific set of benchmark datasets. While these datasets cover a range of problem types, they may not fully represent the diversity of real-world problems that LLMs might encounter. Expanding our evaluation to a broader set of tasks and domains would provide a more comprehensive assessment of our method's generalizability and robustness.

Despite these limitations, we believe that our work represents a significant step forward in enhancing the problem-solving capabilities of LLMs. By explicitly addressing these challenges, we hope to inspire further research that will lead to even more powerful and reliable AI systems capable of tackling complex real-world problems.

References

Peter Auer. 2002. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Peizhong Gao, Ao Xie, Shaoguang Mao, Wenshan Wu, Yan Xia, Haipeng Mi, and Furu Wei. 2024. Meta reasoning for large language models. *Preprint*, arXiv:2406.11698.

Gemma-Team. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Zelalem Gero, Chandan Singh, Hao Cheng, Tristan Naumann, Michel Galley, Jianfeng Gao, and Hoifung Poon. 2023. Self-verification improves few-shot clinical information extraction. In *ICML 3rd Workshop on Interpretable Machine Learning in Healthcare (IMLH)*.

Google-Team. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. 2024. LLM reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

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

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, Doha, Qatar. Association for Computational Linguistics.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. 2024. Better zero-shot reasoning with role-play prompting. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4099–4113.

Llama-Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

OpenAI. 2024. Openai o1 system card.

OpenAI-Team. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094.

Pruthvi Patel, Swaroop Mishra, Mihir Parmar, and Chitta Baral. 2022. Is a question decomposition unit all we need? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4553–4569.

Yuli Qiu, Jiashu Yao, Heyan Huang, and Yuhang Guo. 2024. Optimizing chain-of-thought reasoning: Tackling arranging bottleneck via plan augmentation. *Preprint*, arXiv:2410.16812.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752.

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

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Siheng Xiong, Ali Payani, Yuan Yang, and Faramarz Fekri. 2024. Deliberate reasoning for llms as structure-aware planning with accurate world model. *Preprint*, arXiv:2410.03136.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

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. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

Armel Randy Zebaze, Benoît Sagot, and Rachel Bawden. 2024. Tree of problems: Improving structured problem solving with compositionality. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18028–18047, Miami, Florida, USA. Association for Computational Linguistics.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. *Preprint*, arXiv:2312.07104.

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

A Appendix

A.1 Implementation Details

A.1.1 Experimental Details and Hyperparameter Settings

This appendix provides essential details regarding the experimental setup. Table 7 outlines the default hyperparameter values used in our experiments, specifically those reported in Tables 2 and 5. These values were selected based on preliminary experiments aimed at balancing computational cost with the quality of generated plans: Exploration Weight ($C = 1.0$) controls the balance between exploration and exploitation; Maximum Depth (10) limits the search tree's depth; and Number of Rollouts (8) determines the number of simulations per node expansion.

Hyperparameter	Default Value
Exploration Weight (C)	1.0
Q Initialization	0
Maximum Depth	10
Number of Rollouts	8

Table 7: Default Hyperparameter Values for our HierarchicalMCTS

A.1.2 Code Implementation

To ensure reproducibility and facilitate fair comparisons, we have made our code and demo available on an anonymous GitHub repository. All experiments were conducted using the official Docker container `lmsysorg/sglang` from the SGLang platform (Zheng et al., 2024) with 8 NVIDIA H800 GPUs.

For benchmarking, we utilized the official implementations of the methods proposed by Wei et al. (2022), Wang et al. (2023a), and Hao et al. (2023). We remastered the CoT and Plan-and-Solve approaches using the authors' code to ensure accurate replication of their results. This approach guarantees that our comparisons are conducted on a level playing field, thereby strengthening the validity of our findings.

A.2 Discussion

A.2.1 Differences between Planning and Reasoning

This section elaborates on the conceptual differences between planning and reasoning, a distinction that is fundamental to this research.

Planning is the process of formulating a high-level strategy or sequence of steps to achieve a goal. It is forward-looking, focusing on the overall approach and considering multiple possible paths before selecting the most promising one. In the context of LLMs, planning involves generating a structured outline of the reasoning process.

Reasoning, in contrast, is the process of executing the individual steps outlined in the plan. It involves performing calculations, making deductions, and drawing inferences based on the given information and the chosen plan. Reasoning is primarily concerned with the accuracy and logical consistency of each step within the predetermined framework.

Several key aspects highlight the differences between planning and reasoning.

Temporal Relationship Planning occurs before execution, setting the stage for reasoning. Reasoning follows planning, involving the actual execution of the planned steps.

Error Propagation Characteristics Errors in planning impact the overall strategy, potentially leading to systemic failures. However, such errors are often easier to detect and correct before execution begins. In contrast, errors in reasoning occur at individual calculation or deduction steps. These mistakes can compound through the reasoning chain, making them harder to detect until reaching incorrect conclusions.

Cognitive Load Planning requires holistic understanding and strategic thinking. It focuses on the relationships between steps and considers multiple possible approaches to achieve the goal. Reasoning, on the other hand, demands precise execution of logical operations. It concentrates on the accuracy of individual steps, following the predetermined path set by the plan.

Adaptability Planning is more flexible and can be modified based on initial results. It allows for alternative approaches and can incorporate feedback from preliminary attempts. Reasoning is more rigid, as it must follow established logical rules and maintain consistency within the chosen approach. It is also limited by the constraints of the selected method.

Our HierarchicalMCTS leverages these differences by explicitly separating planning from execution. By using our approach to hierarchically explore and optimize plans, we address the challenges of error propagation and logical inconsistency often encountered in LLM reasoning. Understanding these distinctions is crucial for developing more robust and reliable LLM-based reasoning systems.

A.2.2 Hierarchical Planning vs. Auto-Regressive Planning

This section provides a comparative analysis between our proposed HierarchicalMCTS planning and existing auto-regressive planning methods. We will examine the theoretical frameworks underpinning each approach, highlighting both their mechanistic differences and their distinct optimization objectives.

Auto-regressive planning, mirroring the token-by-token generation process of LLMs, implicitly treats planning as a Markov Decision Process. Here, each planning step represents a state, and generating the next step is viewed as an action. A critical limitation is the reliance on the Markov property, where each step depends solely on its immediate predecessor. This method tends to make locally optimal choices through a greedy approach at each step, neglecting the overall plan's optimality. This step-by-step search is analogous to a greedy search of possible planning states, a fundamental limitation that leads to error propagation and sub-optimal plans.

The most prominent consequence of this greedy, state-level approach is the compounding of errors. Since each planning step relies only on the previous one, errors occurring early in the plan are propagated through subsequent steps, making it difficult to converge on a correct solution. Moreover, this inherently sequential nature prevents the model from backtracking to correct prior errors. Therefore, despite being easy to implement in existing LLM architectures, this greedy, state-level method is ultimately inadequate for complex planning tasks.

In contrast, HierarchicalMCTS reframes planning as a global optimization problem. Rather than viewing planning as a series of individual steps, it explicitly explores the space of **complete plans** as unified entities, enabling the evaluation of plan quality as a whole. By utilizing a modified MCTS, guided by specialized evaluation agents that serve as reward functions, HierarchicalMCTS explores the plan space in a best-first manner, promoting the discovery of globally optimal strategies. This is in stark contrast to the greedy, step-by-step approach of auto-regressive planning.

Furthermore, unlike the sequential nature of auto-regressive planning, MCTS allows for backtracking and error correction by exploring multiple planning alternatives. This enables the model to adapt and refine plans prior to execution. For example, while an auto-regressive planner might incorrectly derive "2+2=4, then 4*3=12" when asked to "compute 2+2*3", a HierarchicalMCTS planner would explore alternative plans, as review agent will feasibility check the plan and reject the incorrect one. By reframing the planning problem as global optimization, HierarchicalMCTS provides a more robust planning strategy and a theoretical shift in perspective compared to auto-regressive methods.

A.2.3 Vanilla MCTS vs HierarchicalMCTS

This section provides a critical comparison between vanilla MCTS and our proposed HierarchicalMCTS, highlighting the fundamental limitations of vanilla MCTS when applied to LLM planning and underscoring the targeted solutions offered by our approach.

While prior work has explored vanilla MCTS for LLM planning (Hao et al., 2023; Wang et al., 2024), closer examination reveals crucial limitations inherited from its state-level, auto-regressive nature. These limitations prevent vanilla MCTS from effectively addressing the unique challenges in LLM reasoning, highlighting the necessity of HierarchicalMCTS. Specifically, three main limitations plague vanilla MCTS in the context of LLMs:

Searching objective First, vanilla MCTS typically searches for states of the plan directly, thereby mirroring the step-by-step nature of auto-regressive methods. This makes it equally susceptible to error propagation, local optima, and lack of a global perspective by only considering the current state, not the entire plan. This is a fundamental limitation as it does not overcome the limitations of state-level planning.

Unbounded action space Second, unlike traditional Reinforcement Learning (RL) where action spaces are constrained, vanilla MCTS for LLMs faces an unbounded action space, where the LLM can generate any possible next state. This uncontrollable search space makes it extremely difficult for vanilla MCTS to converge to an optimal plan due to the sparse reward signal and infinite branching.

Exploration vs Exploitation Third, while MCTS is designed to explore the search space, balancing exploration and exploitation in vanilla MCTS often leads to premature cutoffs, especially when planning for LLMs. This is because it searches for the most promising next step, not the most promising overall plan, leading to premature convergence to local optima, limiting the exploration of superior plans.

In contrast, HierarchicalMCTS is explicitly designed to overcome these limitations in a Evolutionary Algorithms way. By changing the search objective from states of the plan to **complete plans**, we transform the search from a state-by-state decision-making process to a global plan optimization process. This shift removes the limitations inherent to auto-regressive and vanilla MCTS planners.

This critical shift allows the model to assess the entire reasoning trajectory before execution, enabling a global optimization of complete plans, and allowing more powerful search by reframing the objective from "most promising next step" to "most promising plan." For instance, while vanilla MCTS might incrementally build a plan "2+2=4; then..." for "2+2*3", HierarchicalMCTS explores complete plans, such as "calculate multiplication, calculate addition" and "calculate addition then multiplication," selecting the one with the highest logical consistency and feasibility, thus bypassing local optima. By redesigning the search space and objective, HierarchicalMCTS unlocks the full potential of MCTS in LLM planning, providing a more effective solution for complex reasoning tasks.

Method Type	Benchmark / Model	Last Letters	Object Tracking
Zero-Shot(ZS) CoT (Kojima et al., 2022)	Qwen2.5-7B-it	21.00	74.80
	Llama3.1-8B-it	26.40	49.33
	Avg.	19.20	62.07
CoT Plan (Wang et al., 2023a)	Qwen2.5-7B-it	55.20	79.33
	Llama3.1-8B-it	15.40	57.94
	Avg.	**35.30**	**68.64**
	Chagnes over ZS CoT	*16.10*	*6.57*
HierarchicalMCTS (Ours)	Qwen2.5-7B-it	56.60	79.33
	Llama3.1-8B-it	12.80	55.43
	Avg.	34.70	67.38
	Chagnes over ZS CoT	*15.50*	*5.31*

Table 8: **Accuracy(%) Comparison on Sequential Reasoning Benchmarks.** Our evaluation compares HierarchicalMCTS against baseline methods on sequential reasoning tasks. Results demonstrate that our MCTS-enhanced Chain-of-Thought approach achieves comparable performance to CoT Plan, particularly in tasks requiring structured planning and systematic reasoning.

A.3 Extended Results

To provide a more comprehensive assessment of our framework's capabilities, we conducted additional experiments on sequential reasoning benchmarks. As shown in Table 8, we evaluated HierarchicalMCTS against baseline methods on tasks requiring structured tracking of information over multiple steps. While maintaining comparable performance to CoT Plan, these results reveal interesting patterns in the efficacy of hierarchical planning for different reasoning modalities. The relatively smaller gains on sequential tasks compared to mathematical reasoning suggest that the benefits of plan-level optimization may vary based on task structure and cognitive demands.

A.4 Count Characters: Example

We present two examples of the Count Characters task, illustrating the application of CoT, Plan-and-Solve, and MCTS planning methods. Figure 6 demonstrates the CoT, Plan-and-Solve approach, and MCTS planning process. These examples provide a detailed step-by-step breakdown of the problem-solving process, highlighting the differences between the two methods.

A.5 Prompt Templates for Task Execution, Evaluation Agents

Task Execution Prompt The Task Execution prompt instructs the LLM to execute a given plan to solve a problem. The LLM follows the plan step-by-step and outputs the final answer formatted within a box using the `\boxed{}` command.

Feasibility Evaluation Prompt The Feasibility Agent evaluates the feasibility of a given plan by checking the logical consistency of each step. The agent provides feedback on the plan's feasibility and assigns a score between 0 and 100 based on the number of logical inconsistencies found. We use regex to extract the score from the agent's response and normalize it to a scale of 0 to 1. Prompt templates for the Feasibility Evaluation Agent are shown in Figure 3.

Logical Consistency Evaluation Prompt Logical Consistency Agent evaluates the logical consistency of a given plan by checking the correctness of each step. The agent provides feedback on the plan's logical consistency and assigns a score between 0 and 100 based on the number of logical inconsistencies found. We use same way to extract and normalize to process the socre. Prompt templates for the Logical Consistency Evaluation Agent are shown in Figure 4.

Task Execution Prompt

System: You are a highly capable AI assistant tasked with solving problems by meticulously following a provided plan.
User:
Problem
{question text}
Plan
{plan text}
Task
1. Execute the plan to solve the given problem.
2. Format your final answer within a box: \boxed{Your final answer}

Figure 2: Task Execution Prompt

Feasibility Evaluation Prompt

System: You are a powerful agent tasked with validating the feasibility of plans. Given a question and corresponding plan, evaluate the plan's feasibility step by step. You should provide a score between 0 and 100, where 100 indicates that the plan is completely feasible and 0 means that the plan is completely infeasible. Your score should be placed in a box: \boxed{Your score}.
User:
Question
{question text}

Plan
{plan text}

Your Task
Please evaluate the feasibility of the plan based on the question step by step. You should provide a score between 0 and 100, where 100 indicates that the plan is completely feasible and 0 means that the plan is completely infeasible. Your score should be placed in a box: \boxed{Your score}. Now, Let's verify the feasibility of the plan step by step.

Figure 3: Feasibility Evaluation Prompt

Logical Consistency Evaluation Prompt

System: You are a powerful agent tasked with validating the logical consistency of plans. Given a question and corresponding plan, evaluate the plan's logical consistency step by step. You should provide a score between 0 and 100, where 100 indicates that the plan is completely logical and 0 means that the plan is completely inconsistent. Your score should be placed in a box: \boxed{Your score}.
User:
Question
{question text}

Plan
{plan text}

Your Task
Please evaluate the logical consistency of the plan based on the question step by step. After your evaluation, provide a score between 100 and 0, where 100 indicates that the plan is completely logical and 0 means that the plan is completely inconsistent. Your score should be placed in a box: \boxed{Your score}. Now, Let's verify the logical consistency of the plan step by step.

Figure 4: Logical Consistency Evaluation Prompt

A.6 Pseudocode for HierarchicalMCTS

Algorithm 1 HierarchicalMCTS

```
Require: Problem P, initial plan C_0, LLM-based planner M, evaluators set E, LLM executor L
Ensure: Optimal plan C*, Solution Y
root ← Node(C_0, 0, 0)                              ▷ Initialize root node with initial plan, Q=0, N=0
leaf_queue ← [root]                                 ▷ Initialize leaf queue with the root node
while leaf_queue is not empty do
    path ← SelectPromissingPath(root)               ▷ Get the path from root to current leaf
    leaf ← Dequeue(leaf_queue, path)                ▷ Get the next leaf node from the queue by path
    if not IsTerminal(leaf) then ▷ Terminate if the plan is optimal (all eval agents give full socre) or reaches the max depth
        Expand(leaf)                                ▷ Add child nodes to current leaf
        for each child in leaf.children do
            scores, feedback ← Evaluate(child, P, E)    ▷ Use LLM Agnet evaluator to review current plan
            Backpropagate(path, scores)             ▷ Update Q and N along the path
            Enqueue(leaf_queue, child)              ▷ Add the new child nodes to the queue
        end for
    end if
end while                                           ▷ Stop seaching if all plans in search tree has been fully explored
C* ← SelectBestPlan(root)                           ▷ Select the plan with highest value
Y ← ExecutePlan(P, C*, L)
return C*, Y

function EXPAND(node)
    modified_plan ← ModifyPlan(node.plan, node.feedback, M)   ▷ Use LLM to modify plan based on the feedback
    new_node ← Node(modified_plan, node.problem)    ▷ Create a new node based on modified plan
    node[children] ← {new_node, new_node, new_node} ▷ Add the new nodes as children of current node
end function

function EVALUATE(plan, P, E)
    scores ← {}                                     ▷ Initialize an empty dictionary to store evaluation scores
    feedback ← []                                   ▷ Initialize an empty list to store feedback text
    for each evaluator in E do
        score, feedback ← evaluator.evaluate(P, C)  ▷ Get evaluation score and text, as detailed in Appendix A.5
        scores[evaluator.class_name] ← score        ▷ Store the score with evaluator name
        feedback.append(feedback)                   ▷ Store the feedback text
    end for
    return scores, feedback                         ▷ Return all scores and feedback
end function
```

Count Characters: Example 1

Problem: how many R’s are there in the word “Strawberry”?

—

CoT Prompt: Let’s think step by step.

CoT Result:
Here’s how to count the ’R’s in "strawberry":
1. **Identify the ’R’s:** Look for the letter ’R’ in the word.
2. **Count:** There are **two** ’R’s in the word "strawberry".

—

Planning Prompt:
To count the R’s in "strawberry." we can iterate throuoh each letter of the word, Initially, we’ll set a counter to zero, for each leter encountered, we’ll check if itis an "R" character. If it is, we’ll increment our counter, This proess will continwe until we’ve examined all letters in the word, allowing us to determine the final count of "R’s present.

Planning Result:
Following the plan:
1. **Initialize counter:** ‘r_count = 0‘.
2. **Iterate through the word:** Check each letter of "strawberry" for ’R’.
3. **Increment counter:** If the letter is ’R’, increment the counter (‘r_count = r_count + 1‘).
4. **Final Count:** The word "strawberry" contains **3** ’R’s.

—

MCTS Planning:
1. **Define the task:** Our goal is to count the number of "R’s present within the word "strawberry."
2. **Iterate through the word:** We will examine each letter of "strawberry" one at a time.
3. **Check for "R":** If the letter is an "R", increment a counter to keep track of our count.
4. **Increment counter:** If the letter is an "R", increment a counter to keep track of our count.
5. **Complete iteration:** We will continue this process until we have examined every letter.
6. **Return the count:** After processing all letters, the counter will hold the final number of "R’s.

MCTS Planning Result:
Here’s the breakdown:
1. **Define the task:** We need to count how many "R’s are in "strawberry".
2. **Iterate through the word:** Let’s examine each letter:
* **s** - Not an R, counter remains 0.
* **t** - Not an R, counter remains 0.
* **r** - **It’s an R! Increment the counter to 1.**
* **a** - Not an R, counter remains 1.
* **w** - Not an R, counter remains 1.
* **b** - Not an R, counter remains 1.
* **e** - Not an R, counter remains 1.
* **r** - **It’s an R! Increment the counter to 2.**
* **r** - **It’s an R! Increment the counter to 3.**
* **y** - Not an R, counter remains 3.
3. **Complete iteration:** After examining all letters, the counter holds **3** R’s.
4. **Result:** There are **3** R’s in the word "strawberry".

Figure 5: Character counting example

Count Characters: Example 2

Problem: how many R's are there in the word "Pneumonoultramicroscopicsilicovolcanoconiosis"?

—

CoT Prompt: Let's think step by step.

CoT Result:

1. **Identify the letter:** We're looking for the letter 'R'.
2. **Scan the word:** Carefully read through the word "Pneumonoultramicroscopicsilicovolcanoconiosis".
3. **Count the occurrences:** Count each instance of the letter 'R'. There is only one 'R'.

Therefore, there is only **one** 'R' in the word "Pneumonoultramicroscopicsilicovolcanoconiosis".

—

Planning Prompt:

To count the R's in "Pneumonoultramicroscopicsilicovolcanoconiosis." we can iterate throuoh each letter of the word, Initially, we'll set a counter to zero, for each leter encountered, we'll check if itis an "R" character. If it is, we'll increment our counter, This proess will continwe until we've examined all letters in the word, allowing us to determine the final count of "R's present.

Planning Result:

Following the plan:

1. **Initialization:** Set a counter to zero (e.g., 'r_count = 0').
2. **Iteration:** Iterate through each letter of the word "Pneumonoultramicroscopicsilicovolcanoconiosis".
3. **Comparison:** For each letter, check if it is equal to "R".
4. **Increment:** If the letter is "R", increment the counter ('r_count = r_count + 1').
5. **Final Count:** After iterating through all letters, the value of 'r_count' will be the total number of "R"s.

Let's apply this to the word:

The word "Pneumonoultramicroscopicsilicovolcanoconiosis" contains **one** "R".

—

MCTS Planning Prompt:

1. **Define the task:** Our goal is to count the number of "R's present within the word "strawberry."
2. **Iterate through the word:** We will examine each letter of "strawberry" one at a time.
3. **Check for "R":** If the letter is an "R", increment a counter to keep track of our count.
4. **Increment counter:** If the letter is an "R", increment a counter to keep track of our count.
5. **Complete iteration:** We will continue this process until we have examined every letter in the word.
6. **Return the count:** After processing all letters, the counter will hold the final number of "R's in the word.

MCTS Planning Result:

Okay, let's adapt your plan to count the R's in "Pneumonoultramicroscopicsilicovolcanoconiosis":

Define the task: Our goal is to count the number of "R's" present within the word "Pneumonoultramicroscopicsilicovolcanoconiosis".

Iterate through the word: We will examine each letter of "Pneumonoultramicroscopicsilicovolcanoconiosis" one at a time.

Let's start with a counter set to 0.

* **P** - Not an R, counter remains 0.
* **n** - Not an R, counter remains 0.
* **e** - Not an R, counter remains 0.
* **u** - Not an R, counter remains 0.
* **m** - Not an R, counter remains 0.
* **o** - Not an R, counter remains 0.
* **n** - Not an R, counter remains 0.
* **o** - Not an R, counter remains 0.
* **u** - Not an R, counter remains 0.
* **l** - Not an R, counter remains 0.
* **t** - Not an R, counter remains 0.
* **r** - **It's an R! Increment the counter to 1.**
* **a** - Not an R, counter remains 1.
* **m** - Not an R, counter remains 1.
* **i** - Not an R, counter remains 1.
* **c** - Not an R, counter remains 1.
* **r** - **It's an R! Increment the counter to 2.**
* **o** - Not an R, counter remains 2.
* **s** - Not an R, counter remains 2.
* **i** - Not an R, counter remains 2.
* **l** - Not an R, counter remains 2.
* **i** - Not an R, counter remains 2.
* **c** - Not an R, counter remains 2.
* **o** - Not an R, counter remains 2.
* **v** - Not an R, counter remains 2.
* **o** - Not an R, counter remains 2.

SKIP FOR SPACE

Complete iteration: We have now processed all letters, and the counter holds the final number of "R's" in the word.

Result: There are **2** R's in the word "Pneumonoultramicroscopicsilicovolcanoconiosis".

Figure 6: Count Characters Example of "R" in the word "Pneumonoultramicroscopicsilicovolcanoconiosis" with MCTS Planning.