Multi-LLM Collaborative Search for Complex Problem Solving

Anonymous ACL submission

Abstract

Large language models (LLMs) often struggle with complex reasoning tasks due to their limitations in addressing the vast reasoning space and inherent ambiguities of natural language. We propose the Mixture-of-Search-Agents (MoSA) paradigm, a novel approach leveraging the collective expertise of multiple LLMs to enhance search-based reasoning. MoSA integrates diverse reasoning pathways by combining independent exploration with iterative refinement among LLMs, mitigating the limitations of single-model approaches. Using Monte Carlo Tree Search (MCTS) as a backbone, MoSA enables multiple agents to propose and aggregate reasoning steps, resulting in improved accuracy. Our comprehensive evaluation across four reasoning benchmarks demonstrates MoSA's consistent performance improvements over single-agent and other multi-agent baselines, particularly in complex mathematical and commonsense reasoning tasks. We release the anonymous code at https://anonymous.4open.science/r/MoSA-4C47/.

1 Introduction

Large language models (LLMs) face challenges with complex reasoning, even when augmented with linearized reasoning chains (e.g., Chain-of-Thought), due to the vast reasoning space inherent in the complexity and ambiguity of natural languages. A promising approach is step-wise search-based reasoning, which decomposes the reasoning problem into a traversal over a directed graph, where nodes and edges represent individual reasoning sub-steps distributed across the expansive reasoning space. Related methods have applied various search algorithms to LLMs, such as breadth-first search (BFS), depth-first search (DFS) (Yao et al., 2024; Besta et al., 2024), and best-first search (Hao et al., 2023; Zhang et al., 2024b; Qi et al., 2024).

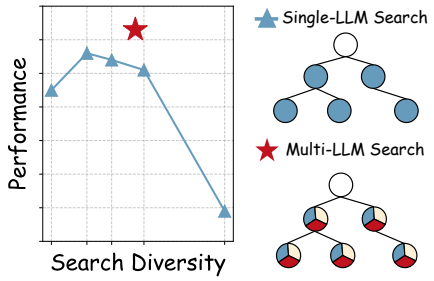

Figure 1: Reasoning performance on MATH-500 against search trajectory diversity. While the diversity of single-LLM search varies with different sampling temperatures, the multi-LLM search consistently achieves superior performance. More details are provided in § 5.

A successful search trial is featured with diverse yet effective explorations (Hao et al., 2023; Yao et al., 2024). A straightforward method to enhance diversity involves increasing the temperature, thereby making the probability distribution more uniform. This is typically combined with top-k and top-p sampling to balance diversity and quality. However, as shown in Figure 1, despite these sampling techniques, achieving a balance between diversity and quality remains challenging and necessitates careful tuning. Besides, even with near-optimal sampling parameters, a single LLM might still get trapped in local optima due to constraints inherent in its data and architecture.

To mitigate this limitation, an alternative solution is to aggregate the specialized strengths of multiple LLMs. Recent work (Wang et al., 2024b) has demonstrated that multiple LLMs can collaboratively enhance their instruction-following capabilities by post-editing each other's responses to the same instruction. Motivated by this progress, we explore leveraging the collective expertise of multiple LLMs for search-based reasoning, which, to the best of our knowledge, has not been previously tested. Figure 1 illustrates the reasoning accuracy on the MATH-500 dataset as a function of search diversity. The performance of search using a single

LLM initially improves with increased temperature but subsequently degrades, remaining consistently lower than that of multiple-LLM search.

In this work, we propose Mixture-of-Search-Agents (MoSA), an advanced paradigm for step-wise search-based reasoning that aggregates the complementary strengths of multiple LLMs, leveraging both independent and collaborative contributions to search for reasoning sub-steps more effectively. As illustrated in Figure 3, multiple LLMs propose diverse potential search directions at each reasoning step, either independently or through iterative refinement of each other's outputs. This hybrid approach ensures that the reasoning process is not constrained by the limitations or biases of any single model. For instance, one model may excel at identifying a promising initial direction, while another might build on it to refine or extend the reasoning path. By combining independence and collaboration, the framework avoids local optima while enhancing reasoning accuracy in the search.

We performed a comprehensive evaluation of MoSA across four reasoning benchmarks. The findings indicate that MoSA consistently outperforms its single-LLM counterpart in reasoning accuracy with an average improvement of 1.71%. Additionally, our results indicate a synergistic interaction between multi-agent collaboration and search-based reasoning. Further analysis and ablation studies reveal a key challenge for single-agent search-based reasoning: balancing diversity and quality varies across different benchmarks. We also confirm a positive correlation between reasoning performance and the number of distinct search agents, validating the efficacy of multi-agent search. Finally, experiments with an extended action set demonstrate the robustness of MoSA across diverse types of search actions.

2 Related Work

2.1 Reasoning with LLMs

The recent focus on large language models is partly due to their exceptional performance in solving complex reasoning tasks. A prominent example is Chain-of-Thought (CoT) reasoning (Wei et al., 2023). Recent advancements include self-consistency (Wang et al., 2023), problem decomposition (Zhou et al., 2023b), the use of tools (Gao et al., 2023; Chen et al., 2023), and search-based methods (Hao et al., 2023; Yao et al., 2024; Qi et al., 2024). Among these approaches, MoSA is most closely aligned with search-based reasoning methods.

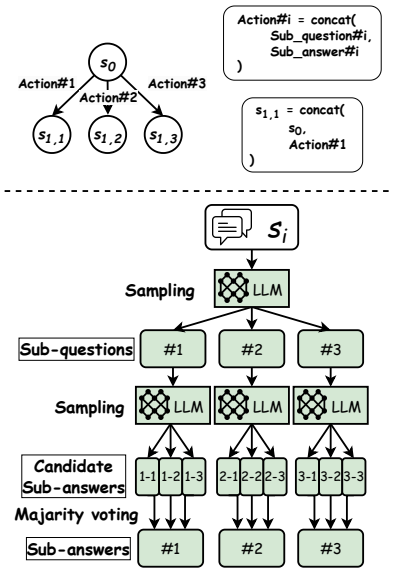

Figure 2: *Top*: An overview of the root node s_0 and its expanded child nodes. *Bottom*: The detailed framework for generating new actions (i.e., sampling sub-questions and sub-answers).

Search-based Reasoning Search-based reasoning has demonstrated effectiveness, particularly for solving complex, multi-step problems (Hao et al., 2023; Yao et al., 2024; Chen et al., 2024c; Zhang et al., 2024b; Chen et al., 2024a; Qi et al., 2024; Zhang et al., 2024a; Zhou et al., 2023a; Koh et al., 2024; Ding et al., 2024). One of the recent state-of-the-art systems in this domain is rStar (Qi et al., 2024). rStar introduces two key innovations: (1) expanding the Monte Carlo Tree Search (MCTS) action space from one or two actions to five; and (2) employing a secondary LLM to verify the reasoning trajectories generated by the primary LLM through MCTS. In § 5, we empirically demonstrate that our method is complementary to the enriched action set of rStar.

2.2 LLM Ensemble

Ensembling, a widely used technique for leveraging the strengths of multiple models, remains highly effective in the era of LLMs. Jiang et al. (2023) proposed pairwise reranking of LLM outputs and fusing multiple responses using a trained generative model. Several studies have proposed training routing functions to match queries with appropriate LLMs (Lu et al., 2023; Shnitzer et al., 2023; Wang et al., 2024a). Others have proposed averaging the output distributions of multiple

LLMs (Huang et al., 2024).

Another line of research focuses on multi-agent collaboration, where multiple LLMs interact to discuss or debate specific topics (Du et al., 2023; Liang et al., 2024; Chan et al., 2023; Xu et al., 2023; Liu et al., 2024; He et al., 2023; Chen et al., 2024b; Zhang et al., 2024c). Common design variations in this paradigm include role assignments for LLMs (e.g., debaters and judges) and discussion mechanisms (e.g., symmetric versus asymmetric interactions).

3 Method

Search-based methods have been extensively used to tackle complex reasoning tasks, such as coding and mathematics, by breaking these problems into multiple search steps (Zhou et al., 2023b; Yao et al., 2024; Hao et al., 2023). Our proposed paradigm is readily applicable to various search algorithms, with the Monte Carlo Tree Search (MCTS) algorithm (Kocsis and Szepesvári, 2006; Coulom, 2007) adopted as the search backbone in this work. This section first introduces the baseline MCTS-based reasoning method with a single search agent (Hao et al., 2023; Qi et al., 2024) in § 3.1, followed by our method, which leverages the expertise of multiple LLMs as search agents in § 3.2.

3.1 Baseline Framework

Overview Given a problem x and a generator π^*, MCTS involves iteratively building a search tree starting from the root node x. We first define the state space $\mathcal{S}$ and the action space $\mathcal{A}$. In our case, each state $s_j \in \mathcal{S}$ captures the actions (i.e., reasoning steps) generated so far alongside a specific trajectory in the search tree, while each action $a_j \in \mathcal{A}$ represents the next reasoning step based on the current state and the type of action chosen. As shown in the upper part of Figure 2, given the selected node s_i (i.e., the reasoning steps generated so far), a step of *Expansion* essentially creates a set of child nodes. A child node is created by concatenating s_i with the new action, with that action being the next reasoning step generated by a search agent (e.g., an LLM) given s_i.

Action Space rStar (Qi et al., 2024) defined a comprehensive set of actions for MCTS-based LLM reasoning. The set of actions, $A = \{A1, A2, A3, A4, A5\}$, includes:

- $A1$: Propose a one-step thought;
- $A2$: Propose the remaining thought steps;
- $A3$: Propose the next sub-question along with its answer;
- $A4$: Answer the sub-question again;
- $A5$: Rephrase the question.

As for the main experiments in this paper, we adopt $A3$ as the primary action (i.e., RAP (Hao et al., 2023) in Table 1), composed of a *sub-question* and its corresponding *sub-answer*, i.e., $\text{action}_i \equiv \text{concat}(\text{sub_question}_i, \text{sub_answer}_i)$. For instance, an action can be "*### Sub-question 3: Does the sum of the previous two digits equal 8? ### Sub-answer 3: The two digits are 3 and 5. We have* $3 + 5 = 8$*, so the answer is yes.*". We consider the other actions along with their effects in an ablation analysis (§ 5). We present Algorithm 1 in the Appendix as a detailed illustration of generating new actions, i.e., combinations of sub-question & sub-answer. For a given state s_i, the algorithm traverses all possible actions, where the final sub-answer for each sub-question is determined by a heuristic function, e.g., majority voting.

Reward Function Following Hao et al. (2023); Qi et al. (2024), we consider a simple yet effective reward function: actions that frequently lead to correct final answers are assigned higher rewards. Specifically, $Q(s, a)$, the reward value for node s created by action a, receives a positive reward if a trajectory containing node s reaches a correct final answer, and no reward otherwise. Since the gold answer is not available during testing, the confidence given by *majority voting* is regarded as an approximation of the reward value.

MCTS Iterations Typically, each MCTS iteration involves four steps: *Selection*, *Expansion*, *Simulation*, and *Back-propagation*. To balance exploration and exploitation, we adopt the widely-used *Upper Confidence Bounds for Trees* (UCT) algorithm (Kocsis and Szepesvári, 2006) for *Selection*. Formally, a node s is selected to maximize:

$$\text{UCT}(s, a) = \frac{Q(s, a)}{N(s, a)} + c\sqrt{\frac{\ln N_{\text{parent}}(s)}{N(s, a)}} \quad (1)$$

where $N_{\text{parent}}(s)$ is the number of times the parent node of s has been visited, $N(s, a)$ is the number of times node s has been visited, and c is a constant. Once the node s is selected, an *Expansion* step is performed to add child nodes to s. After that, starting from a random child node, a *Simulation* is performed using the default rollout policy until a terminal node is obtained or a predefined maximum

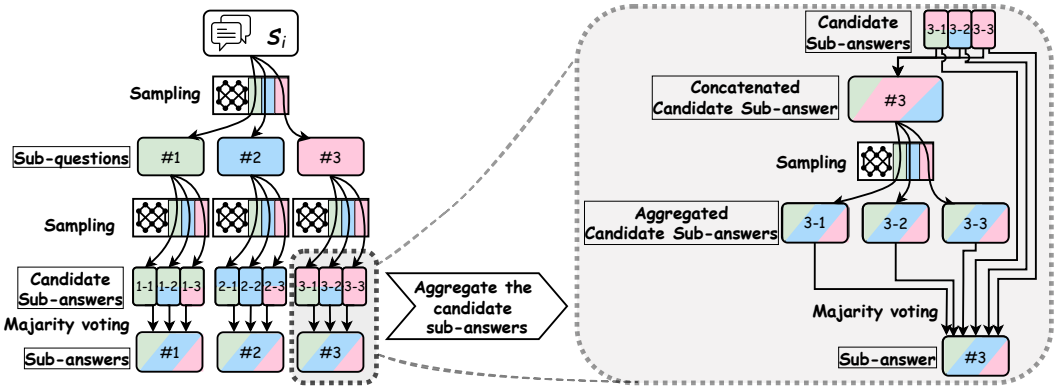

Figure 3: Generate three new actions using MoSA. *Left*: Use MoSA to propose sub-questions and sub-answers. *Right*: Use MoSA to aggregate candidate sub-answers.

depth is reached. The outcome of the simulation determines the reward, which is then propagated back up the tree during the *Back-propagation* step. Upon multiple iterations, we consider each leaf node as a solution. In this work, we focus on *Expansion*, which aims to effectively expand the search space.

Sampling Diversity Applying stochastic sampling techniques in LLM generation is essential for introducing diversity to MCTS. As presented in the lower part of Figure 2, given the selected state s_0, the sub-questions and the sub-answer candidates are all stochastically sampled using temperature scaling, top-k sampling and nucleus sampling (Holtzman et al., 2020). In § 5, we empirically alter search diversity by manipulating generation temperature for single-LLM search.

3.2 Mixture-of-Search-Agents

Conventional Monte Carlo Tree Search (MCTS) methods utilizing a single model face two significant limitations: (1) Encouraging search diversity while maintaining generation quality is challenging (Zhang et al., 2020), necessitating meticulous tuning of sampling parameters to balance the trade-off between these aspects; (2) using heuristic metrics like majority voting to determine the final sub-answer can be less accurate when the model favors incorrect search directions. To this end, we explore a simple yet effective alternative, Mixture-of-Search-Agents (MoSA), which employs multiple agents to perform search algorithms like MCTS and utilizes a neural function to refine the candidate step-wise outputs. Firstly, leveraging the distinct distributions from different models intrinsically yields better generation diversity, alleviating the necessity for sampling parameters optimization. Additionally, incorporating a neural function enhances the robustness of answer aggregation.

Figure 3 illustrates how our method generates three new actions starting from the current node s_i. Unlike vanilla single-model search, MoSA employs multiple agents (denoted by distinct colors) to explore diverse actions, such as sub-questions and sub-answers. In the remainder of this section, we illustrate two roles performed by MoSA when generating new actions in MCTS. Specifically, we will start with the straightforward improvement, MoSA *as Proposers*, where multiple agents are involved for sampling actions; then we will introduce the more intricate MoSA *as Aggregators*, which extends the heuristic majority voting method to an aggregating phase where multiple LLMs read and refine the answers given by all.

MoSA as Proposers to Diversify Actions The left side of Figure 3 shows MoSA *as Proposers*, where the single search agent adopted by the baseline MCTS method (as in Figure 2) is replaced by MoSA. MoSA leverages multiple LLMs to enhance action diversity by fulfilling two sub-roles: multi-agent proposers that generate *sub-questions* and *sub-answers*.

Generating a new search action begins with sampling a sub-question from the current state s_i. The sub-question proposing phase essentially controls the directions of the current search step because whatever follows within this step is constrained by the scope of that sub-question. Because of this, we consider maintaining the independence among sub-questions, ensuring that the initial search direction indicated by each sub-question is independent of others. As shown in the upper-left part of Figure 3, this effectively diversifies the sampled sub-

questions as the same s_i is colored with distinct characteristics after going through different LLMs.

After the initial search directions are created, the target is to comprehensively explore each search direction. To achieve this, each sub-question is answered by various LLMs, generating a diverse set of candidate sub-answers. These candidates are then aggregated to reach a finalized sub-answer. A simple yet effective aggregating method is majority voting, leveraging the principle of self-consistency (Wang et al., 2023).

MoSA as Aggregators for Collaborative Refinement We introduce a neural function, termed "aggregator", to mitigate the limitation of majority voting for selecting the final answer. An aggregator leverages the innate capability of the LLM to critique, compare and aggregate multiple answers into a final answer. Specifically, we prompt each LLM to consolidate all responses into an aggregated answer (see Appendix C for detailed prompts), resulting in a new set of aggregated answers as illustrated in the right section of Figure 3. The underlying intuition is that this aggregation process enhances the likelihood of producing correct answers by facilitating comparisons among different responses, thereby increasing the overall success rates for correct answers under majority voting. We present an example below to illustrate this intuition.

In the previous section, we consider majority-voting after obtaining candidate sub-answers from diverse proposers. Let us consider a sub-question that requires 3 sub-answers to be generated and an MoSA component consisting of 3 distinct LLMs. We simply assume that each LLM proposes one sub-answer. If there are k LLMs that are proficient at this sub-question and the other $3-k$ are not, then it is likely that we would have k *good* sub-answer candidates and $3-k$ *bad* candidates[1]. With majority voting, a *bad* finalized answer is likely if $k \geq 2$.

Now we turn to use MoSA to aggregate the candidate sub-answers and then include the aggregated sub-answers into majority-voting. The inputs for all three aggregator LLMs are the same, which concatenates the sub-question and all the three candidate sub-answers. We hypothesize that a *bad* aggregator that receives at least a *good* sub-answer could yield a sub-answer that is at least better than its original *bad* sub-answer. Such a hypothesis has been empirically verified in the case of instruction following by Wang et al. (2024b), who showed that many LLMs can generate higher-quality responses by building upon outputs from other LLMs. Thus, if the two *bad* aggregator LLMs can learn from the *good* sub-answer and generate *good* aggregated sub-answers, then we will have 4 *good* sub-answers and 2 *bad* ones, which lead to a *good* finalized sub-answer.

[1]For clarity in illustrating our motivation, we simplify the correctness of candidate sub-answers into two groups: *good* and *bad*. This abstraction helps explain the role of aggregators in improving answer correctness, though actual correctness exists on a spectrum depending on task complexity and evaluation criteria.

4 Experiments

4.1 Baselines

Few-shot Chain-of-Thought (CoT) feeds the LLM with a few demonstrations followed by the input question. Since we are using instruction-tuned LLMs, we format the demonstrations as multi-turn dialogues. In each turn, the human asks a question and then the assistant answers it.

Self-Consistency@*n* (Wang et al., 2023) also adopts the few-shot CoT prompting scheme, but it samples n independent answers per instance. The final answer is then given by majority voting over the n candidate answers. Except for the conventional single-LLM self-consistency experiments, we also evaluate self-consistency with multiple different LLMs. Such a multi-LLM self-consistency setting can be regarded as a simplified version of Wang et al. (2024b), which collects direct answers from various agents and aggregates them with majority voting.

Reasoning-via-Planning (RAP) (Hao et al., 2023) is a representative LLM-based reasoning method using MCTS. We use it as the foundation to apply MoSA. In each search step, RAP generates one or more sub-questions along with their sub-answers. The original RAP paper adopted different reward functions for different types of tasks. In this work, we use the simple self-consistency score as the reward value, which has been shown to be competitive with those manually designed ones in Appendix A.1 of Qi et al. (2024). Note that the total number of LLM forward calls of a single-LLM method are approximately the same as its multi-LLM counterpart, e.g., RAP + Llama *versus* RAP + MoSA in Table 1.

rStar (Qi et al., 2024) is one of the recent SoTA MCTS-based LLM reasoning methods. The au-

Method		Multi?	Search?	GSM8K	SVAMP	MATH	STG	Avg.
Few-shot CoT	○ Llama	×	×	84.00	86.80	41.60	67.39	69.95
	○ Ministral	×	×	82.41	89.20	40.00	70.60	70.55
	○ Qwen	×	×	84.00	88.60	24.20	66.67	65.87
	○ GLM	×	×	83.85	89.70	40.00	71.32	71.22
w/ Llama	○ Self-Consistency@4	×	×	88.02	89.70	43.80	69.43	72.74
	○ Self-Consistency@32	×	×	90.37	92.40	44.80	70.89	74.62
	○ Self-Consistency@128	×	×	90.98	93.30	52.20	71.32	76.95
	○ Self-Consistency@256	×	×	90.90	92.90	53.20	71.03	77.01
w/ All Four LLMs	○ Self-Consistency@4	√	×	90.45	92.20	46.20	70.45	74.82
	○ Self-Consistency@32	√	×	90.75	93.20	52.60	71.76	77.08
	○ Self-Consistency@128	√	×	91.21	93.70	53.80	72.78	77.87
	○ Self-Consistency@256	√	×	90.98	93.50	54.20	71.47	77.54
RAP *w/* Proposer(s)	○ Llama	×	√	90.52	91.60	53.00	75.40	77.63
	○ Ministral	×	√	88.32	93.10	46.80	71.47	74.92
	○ Qwen	×	√	87.64	92.00	49.80	72.35	75.45
	○ GLM	×	√	88.17	91.10	48.20	75.11	75.65
	• MoSA	√	√	91.13	94.50	54.60	75.69	78.98
RAP *w/* Pro. & Agg.	○ Llama	×	√	90.05	92.50	54.80	75.69	78.26
	○ Ministral	×	√	90.14	93.90	47.40	71.91	75.84
	○ Qwen	×	√	89.84	93.90	50.80	72.93	76.87
	○ GLM	×	√	89.54	91.30	49.60	75.84	76.57
	• MoSA	√	√	**91.96**	**94.90**	**56.60**	**76.42**	**79.97**

Table 1: Main results. "Multi?" refers to whether or not using multi-LLM. "STG" represents StrategyQA. "*w/* Pro. & Agg." refers to "with Proposer(s) and Aggregator(s)". **The highest number** on each dataset is marked in bold while the secondary high is underlined.

thors proposed a comprehensive set of search actions, which we have introduced in § 3.1. We adopt their innovative set of actions to evaluate the effects brought by the scope of action set on MoSA in § 5.

4.2 Experimental Settings

Benchmarks We perform evaluation on four reasoning benchmarks covering different scopes, including three mathematical reasoning datasets (GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023)) and one commonsense reasoning dataset (StrategyQA (Geva et al., 2021)).

Models We adopt four open-sourced instruction-following LLMs to formulate the LLM pool of MoSA: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Qwen-2-7B-Instruct (Yang et al., 2024), Ministral-8B-Instruct-2410 (Mistral, 2024), and GLM-4-9B-Chat (GLM et al., 2024). The number of LLMs could also be made larger or smaller, depending on customized choices. Our later experiments will show that benchmark performances are positively correlated with the number of distinct LLMs.

Implementation Details For few-shot CoT and RAP baselines, we report the results of all four LLMs. For single-LLM Self-Consistency@n, we adopt **Llama-3.1-8B-Instruct** due to its competitiveness and robustness across various benchmarks. For all experiments regarding sampling from multiple LLMs, we try to maintain a pseudo uniform distribution for the `SelectLLM` function in Algorithm 1. That is, if 7 completions need to be sampled and there are 4 distinct LLMs, we manually assign each LLM to sample one completion and then uniformly sample 3 LLMs out of 4 without replacement to finish the remaining 3 completions. We carried out our experiments using A100 GPUs, employing between 1 and 8 GPUs depending on the computational requirements. Hyper-parameter settings are listed in Appendix § B. We also discuss the some limitations of MoSA in Appendix § 6.

4.3 Main Results

We report the main results on the four benchmarks in Table 1. Below we highlight our key findings.

{RAP + MoSA} Outperforming {RAP + Any Single LLM} {RAP + MoSA} consistently yields superior performances across all datasets (GSM8K, SVAMP, MATH-500, StrategyQA) against {RAP + Llama / Ministral / Qwen / GLM}. The strongest MoSA along with RAP *w/* Proposers & Aggregators reaches an average performance (Avg.) of 79.97%. On the challenging MATH-500 benchmark, it obtains exceptional improvements over all single-LLM RAP baselines, outperforming the best baseline for +1.8% accuracy points.

Synergistic Effect between Multi-Agent Collaboration and Search-based Reasoning MoSA integrates two research paradigms: multi-agent col-

Method		Action Set	GSM8K	SVAMP	MATH	STG	Avg.
RAP *w/* Proposer(s)	○ Llama	$A\{2,3\}$	90.52	91.60	53.00	<u>75.40</u>	77.63
	● MoSA		91.13	<u>94.50</u>	54.60	**75.69**	78.98
rStar *w/* Proposer(s)	○ Llama	$A\{1,2,3,4,5\}$	<u>91.36</u>	93.30	<u>59.00</u>	74.96	<u>79.66</u>
	● MoSA		**91.96**	**95.60**	**63.20**	75.11	**81.47**

Table 2: The results of MoSA combined with rStar (Qi et al., 2024), a recent SoTA MCTS-based reasoning method that extends the set of actions. The definitions of $A\{1,2,3,4,5\}$ are in § 3.1. **The highest number** on each dataset is marked in bold while <u>the secondary high</u> is underlined.

laboration and search-based reasoning. When applied independently, each achieves moderate improvements, but their combination yields significantly enhanced results due to synergy effects. (**1**) Transitioning from single-agent to multi-agent: Across all four benchmarks, transitioning from a single LLM to multiple LLMs with the best non-search baseline (Self-consistency) results in an average absolute improvement of +0.53%. By contrast, transitioning from single-agent search (e.g., RAP *w/* Proposers + Llama) to multi-agent search (RAP *w/* Proposers + MoSA) yields a larger average absolute improvement of +1.35%. Augmenting with aggregators further increases the improvement from single-agent search (RAP *w/* Pro. & Agg. + Llama) to multi-agent search (RAP *w/* Pro. & Agg. + MoSA), achieving +1.71%. (**2**) Transitioning from non-search to search-based reasoning: Using a single LLM, the performance gap between non-search (Self-consistency@256) and search (RAP) is +0.62%. This gap widens to +1.44% when employing multiple LLMs, showcasing the synergy between multi-agent collaboration and search. These results highlight that combining multi-agent collaboration with search-based reasoning yields significantly greater performance gains than applying either approach in isolation.

Boosting Search-based Reasoning with MoSA as Aggregators While vanilla RAP performs well, the inclusion of aggregators, particularly with MoSA as Aggregators, significantly enhances performance. For instance, augmenting RAP with a single-LLM aggregator yields an average improvement of +0.63%. This improvement increases to +0.99% when MoSA as Proposers is further enhanced with MoSA as Aggregators.

Search-Based Methods Excel in Complex Reasoning Tasks The best accuracy numbers on GSM8K and SVAMP, both exceeding 90%, suggest these datasets are relatively easier. In contrast, MATH-500 and StrategyQA, with best scores around 55% and 80%, respectively, are more challenging. Notably, search-based methods demonstrate a clear advantage on these more complex datasets, underscoring their effectiveness in tackling intricate reasoning tasks. Take StrategyQA as an example, the best accuracy number with non-search methods (Self-consistency) is 72.78%, which is significantly lower than the best search counterpart (RAP) accuracy (75.69%).

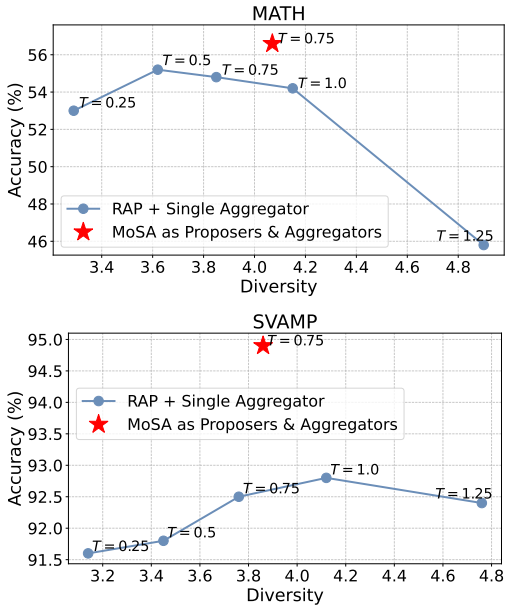

Figure 4: Diversity *versus* accuracy. T = Temperature. We use Llama as the single aggregator since it performs the best among the four LLMs.

5 Analysis

We perform a comprehensive analysis on MoSA in this section. Specifically, we scale the diversity of the single-LLM search baseline and compare it with MoSA. After that, we vary the number of distinct LLMs in MoSA. We also combine MoSA with the rich set of actions proposed by Qi et al. (2024). Finally, we evaluate variations of MoSA by ablating the numbers of proposers and aggregators.

Diversity *versus* Performance For single-LLM search, a common technique to increase generation diversity is to manipulate with decoding hyper-parameters, e.g., the sampling temperature.

#Proposer(s)	#Aggregator(s)	GSM8K	SVAMP	MATH	StrategyQA	Avg.
Single	None	90.52	91.60	53.00	75.40	77.63
Single	Single	90.05	92.50	54.80	75.69	78.26
Single	Multi	91.05	91.90	55.60	76.42	78.74
Multi	None	91.13	94.50	54.60	75.69	78.98
Multi	Single	91.66	94.20	56.00	76.13	79.50
Multi	Multi	**91.96**	**94.90**	**56.60**	**76.42**	**79.97**

Table 3: Ablation analysis to isolate the effects of MoSA as proposers and as aggregators, respectively, for search-based reasoning. By multi, we are referring to the default setting in MoSA, i.e., the four distinct LLMs. **The highest number** on each dataset is marked in bold while the secondary high is underlined.

We modify the temperature of the RAP + Single-LLM as Aggregator baseline on two datasets, with $T = \{0.25, 0.5, 0.75, 1.0, 1.25\}$. Diversity is assessed by calculating the $\{1, 2, 3, 4\}$-gram Vendi Score (Friedman and Dieng, 2023) across search trajectories. Figure 4 illustrates that while the reasoning accuracy of RAP initially improves with increasing diversity, it subsequently declines. More importantly, the two benchmarks favor different temperature values. This suggests that attaining an optimal balance between diversity and reasoning performance requires careful tuning, as balancing diversity and quality can be challenging (Zhang et al., 2020). In contrast, RAP + MoSA with the default sampling parameters consistently holds an advantageous position.

Ablation of LLM Collaboration To evaluate the impact of varying the number of different LLMs in MoSA, we conduct an analysis using 1 to 4 LLMs across four benchmarks, prioritizing them in the following order: Llama, GLM, Qwen, Ministral. All four variants utilize approximately the same number of LLM forward calls, ensuring that the only variable is the number of distinct LLMs involved. Figure 5 shows that increasing the number of different LLMs generally correlates with higher reasoning accuracy, except for a slight decrease in performance when the number of LLMs increases from 3 to 4 on MATH-500. This trend indicates that the diverse expertise contributed by different LLMs significantly enhances search-based reasoning performance.

Support for Extended Action Set rStar (Qi et al., 2024) proposes using a comprehensive set of actions in MCTS-based LLM reasoning. Since enriching the action set is orthogonal to our method, we hypothesize that MoSA is compatible with the enlarged action set. The results in Table 2 support our hypothesis. For example, rStar combined with MoSA boosts the reasoning accuracy on MATH-500 from 59.00% to 63.20%. We also found that

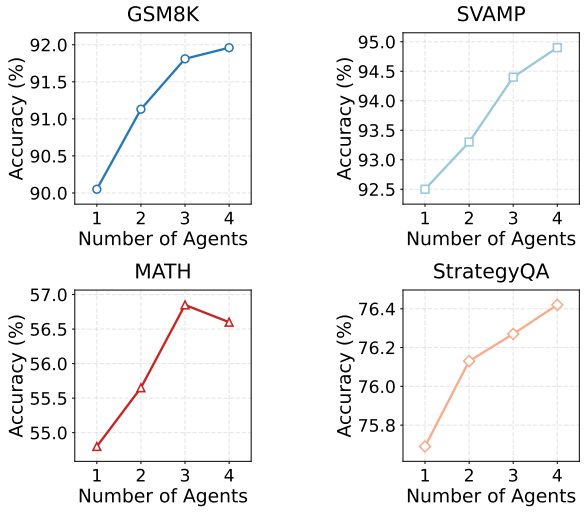

Figure 5: Reasoning accuracy with different number of distinct LLMs as search agents.

enriching the action set is not always beneficial. On StrategyQA, the expanded action set yielded inferior performance; however, we note that our MoSA still demonstrates improvements.

Ablation of Proposers & Aggregators We consider to isolate the effects of MoSA as Proposers and MoSA as Aggregators by ablating the number of distinct LLMs for those two roles. As shown in Table 3, changing the number of distinct proposers to be single yields a larger decrease comparing with ablating the number of aggregators (-1.23% *versus* -0.47%), suggesting that MoSA brings more benefits as proposers.

6 Conclusion

In this work, we investigated a novel paradigm called MoSA. MoSA combines independent exploration and iterative refinement among multiple LLMs to enhance reasoning diversity and accuracy. Experiments across benchmarks demonstrate its consistent advantages over single-LLM and multi-agent baselines, especially in complex tasks. This work underscores the potential of multi-agent collaboration in advancing search-based reasoning.

Limitations

Limitation#1: Computational Overhead

The main limitation of MoSA (and other MCTS-augmented method) is the computational overhead, introduced by the increasing number of forward calls. Below we discuss the inference cost of MoSA and some of the baselines. We will also discuss the real-world practicality of MoSA.

Inference Cost Across all our experiments, we ensure comparable computational costs between MoSA and the most important baseline (i.e., RAP). Specifically:

- RAP / MoSA without aggregation typically requires approximately 200–300 forward calls per query.
- RAP / MoSA with aggregation increases to approximately 500 forward calls.
- Self-consistency@256 requires a similar number of calls with RAP / MoSA without aggregators.

Real-world Practicality In practical deployments, multiple models can be efficiently served concurrently using dedicated inference servers or cloud infrastructure services such as TogetherAI. Such services readily offer diverse LLM inference APIs, making MoSA straightforwardly scalable with minimal operational complexity.

Performance Gain vs. Overhead Given that MoSA ensures a balanced computational distribution across LLMs, the marginal overhead compared to self-consistency methods or single-LLM search (e.g., RAP) is acceptable, and thus the achieved performance improvements (1.71% on average and up to 1.8% on complex tasks) justify the computational investment.

Limitation#2: Prompt Engineering

The improved performances of MoSA cannot be obtained without proper design of the prompt templates (as shown in Appendix § C). Since we did not run experiments with different sets of prompts, it is very likely that the reported results in this paper might vary upon using a different set of prompt templates.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *Preprint*, arXiv:2308.07201.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024a. Alphamath almost zero: Process supervision without process. *Preprint*, arXiv:2405.03553.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. 2024b. ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7066–7085, Bangkok, Thailand. Association for Computational Linguistics.

Qiqi Chen, Xinpeng Wang, Philipp Mondorf, Michael A. Hedderich, and Barbara Plank. 2024c. Understanding when tree of thoughts succeeds: Larger models excel in generation, not discrimination. *Preprint*, arXiv:2410.17820.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

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.

Rémi Coulom. 2007. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, pages 72–83, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2024. Everything of thoughts: Defying the law of penrose triangle for thought generation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1638–1662, Bangkok, Thailand. Association for Computational Linguistics.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *Preprint*, arXiv:2305.14325.

Dan Friedman and Adji Bousso Dieng. 2023. The vendi score: A diversity evaluation metric for machine learning. *Transactions on Machine Learning Research*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, and 37 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *Preprint*, arXiv:2406.12793.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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.

Zhitao He, Pengfei Cao, Yubo Chen, Kang Liu, Ruopeng Li, Mengshu Sun, and Jun Zhao. 2023. LEGO: A multi-agent collaborative framework with role-playing and iterative feedback for causality explanation generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9142–9163, Singapore. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Yichong Huang, Xiaocheng Feng, Baohang Li, Yang Xiang, Hui Wang, Bing Qin, and Ting Liu. 2024. Ensemble learning for heterogeneous large language models with deep parallel collaboration. *Preprint*, arXiv:2404.12715.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14165–14178, Toronto, Canada. Association for Computational Linguistics.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024. Tree search for language model agents. *Preprint*, arXiv:2407.01476.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17889–17904, Miami, Florida, USA. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. Routing to the expert: Efficient reward-guided ensemble of large language models. *Preprint*, arXiv:2311.08692.

AI team Mistral. 2024. Un ministral, des ministraux | mistral ai | frontier ai in your hands. `https://mistral.ai/news/ministraux/`.

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, Online. Association for Computational Linguistics.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. 2023. Large language

model routing with benchmark datasets. *Preprint*, arXiv:2309.15789.

Hongyi Wang, Felipe Maia Polo, Yuekai Sun, Souvik Kundu, Eric Xing, and Mikhail Yurochkin. 2024a. Fusing models with complementary expertise. In *The Twelfth International Conference on Learning Representations*.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024b. Mixture-of-agents enhances large language model capabilities. *Preprint*, arXiv:2406.04692.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. 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 Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Zhenran Xu, Senbao Shi, Baotian Hu, Jindi Yu, Dongfang Li, Min Zhang, and Yuxiang Wu. 2023. Towards reasoning in large language models via multi-agent peer review collaboration. *Preprint*, arXiv:2311.08152.

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, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *Preprint*, arXiv:2406.03816.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024b. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *Preprint*, arXiv:2406.07394.

Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan. 2020. Trading off diversity and quality in natural language generation. *CoRR*, abs/2004.10450.

Jintian Zhang, Xin Xu, Ningyu Zhang, Ruibo Liu, Bryan Hooi, and Shumin Deng. 2024c. Exploring collaboration mechanisms for LLM agents: A social psychology view. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14544–14607, Bangkok, Thailand. Association for Computational Linguistics.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023a. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

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. 2023b. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Main Algorithm of MoSA

We present a detailed illustration of generating new actions, i.e., combinations of sub-question & sub-answer in Algorithm 1.

B Additional Experimental Settings

B.1 Hyper-parameters

The default sampling parameters for LLM generation are {temperature=0.75, top_k=40, top_p=0.95}. Across all MCTS experiments, we set the number of rollouts to 8, the number of sub-questions per node to 4, the number of candidate sub-answers per sub-question to 4, the maximum depth allowed to 5.

B.2 Dataset Statistics

Since we make use of the rStar code base [2] to implement MoSA, we directly adopt the data files released in their git repository. There are 1,319 instances in GSM8K, 1,000 instances in SVAMP, 500 instances in MATH-500, and 687 instances in StrategyQA.

C Additional Implementation Details for Aggregators

In this section, we will show the basic instruction and several in-context learning demonstrations for aggregators.

[2] https://github.com/zhentingqi/rStar/

Algorithm 1 `GenerateActions`: A function for generating actions, i.e., a sub-question along with a sub-answer, given the current state. The implementation of this function using conventional MCTS and MoSA mainly differs in two aspects: (1) In conventional MCTS methods, the number of search agents $m = 1$, while for MoSA, $m > 1$. (2) The `FinalizeSubAnswer` function employs heuristic majority voting for single-model search, while employing an additional neural aggregation function for MoSA (see the right section of Figure 3).

Require: Selected node s_i; Number of sub-questions n_q; Number of candidate sub-answers per sub-question n_a; A set of LLMs $\pi^{\text{mix}} = \{\pi_1, \pi_2, ..., \pi_m\}$
Ensure: A set of new actions $\texttt{new_actions} = \{\texttt{action}_1, \texttt{action}_2, ..., \texttt{action}_{n_q}\}$

```
Initialize new_actions ← ∅ % Prepare the set of new actions
for i = 1 to n_q do
    π^sub_q ← SelectLLM(π^mix) % Select an LLM for generating sub-question
    sub_question_i ← GenerateSubQuestion(π^sub_q, s_i) % Generate the i-th sub-question
    Initialize candidate_sub_answers ← ∅ % Store candidate sub-answers for sub-question i
    for j = 1 to n_a do
        π^sub_a ← SelectLLM(π^mix) % Select an LLM for generating a sub-answer
        candidate_sub_answer_j ← GenerateSubAnswer(π^sub_a, s_i, sub_question_i) % Generate the j-th candidate
        sub-answer
        candidate_sub_answers.add(candidate_sub_answer_j) % Store the candidate sub-answer
    end for
    sub_answer_i ← FinalizeSubAnswer(candidate_sub_answers) % Aggregate or vote on candidate sub-answers

    action_i ← concat(sub_question_i, sub_answer_i) % Form the final action by concatenation
    new_actions.add(action_i) % Add the action to the set of new actions
end for
return new_actions
```

Basic Instruction for Aggregators

TASK:
You are an intelligent and supportive AI assistant. You will receive a collection of responses from various AI assistants regarding a query. Your goal is to synthesize these responses into a single, high-quality response. You should first write down your thoughts as "### Thoughts: <THOUGHTS>.", and then write down the synthesized response in the format of "### Synthesized Response: <SYNTHESIZED RESPONSE>.". If "The answer is: <ANSWER>" appears in the set of responses, you should also write down "The answer is: <ANSWER>"; otherwise, do not write it.
1. Critically assess the information in the responses, as some may be biased, incorrect, or nonsensical.
2. Your synthesized response should not merely echo the provided answers but should present a refined, accurate, and thorough response to the prompt.
3.Ensure that your response is well-organized, coherent, and meets the highest standards of accuracy and reliability.

An Example ICL Demonstration of MATH-500 for Aggregators: User Query

User:
[Here is the user query that was fed into those AI assistants]:
Given a question, please decompose it into sub-questions. For each sub-question, please answer it in a complete sentence, ending with "The answer is <a numeric answer >". When the original question is answerable, please start the subquestion with "Now we can answer the question: <original question>".
Question 5: Find the product CD of the integers C and D for which

$$\frac{C}{x-3}+\frac{D}{x+8}=\frac{4x-23}{x^2+5x-24}$$

for all real values of x except -8 and 3.
Question 5.1: What is the equation obtained by multiplying both sides of the given equation by $(x-3)(x+8)$?
Answer 5.1: $(x-3)(x+8)\cdot\frac{C}{x-3}+(x-3)(x+8)\cdot\frac{D}{x+8}=(x-3)(x+8)\cdot\frac{4x-23}{x^2+5x-24}$
This simplifies to $C(x+8)+D(x-3)=4x-23$.
The answer is $C(x+8)+D(x-3)=4x-23$.
Question 5.2: What are the values of C and D that satisfy the equation $C(x+8)+D(x-3)=4x-23$ for all x except -8 and 3?
Answer 5.2:
[Here are the responses from various AI assistants]:
Response 1:
To find the values of C and D, we can substitute specific values of x into the equation $C(x+8)+D(x-3)=4x-23$ to obtain a system of equations.
Question 5.2.1: What happens if we substitute $x=3$ into the equation $C(x+8)+D(x-3)=4x-23$?
Answer 5.2.1: Substituting $x=3$ gives $C(3+8)+D(3-3)=4\cdot 3-23$, which simplifies to $11C=-11$. The answer is $11C=-11$.
Question 5.2.2: What happens if we substitute $x=-8$ into the equation $C(x+8)+D(x-3)=4x-23$?
Answer 5.2.2: Substituting $x=-8$ gives $C(-8+8)+D(-8-3)=4\cdot(-8)-23$, which simplifies to $-11D=-59$. The answer is $-11D=-59$.
Question 5.2.3: Now we can answer the question: What are the values of C and D?
Answer 5.2.3: Solving the system of equations $11C=-11$ and $-11D=-59$ gives $C=-1$ and $D=7/11\cdot 11=7$. The answer is -1 and 7 respectively.
Response 2:
To find the values of C and D, we can compare the coefficients of the corresponding terms on both sides of the equation.
For the x^2 terms, we have $C+D=0$.
For the x terms, we have $8C-3D=4$.
Solving these equations simultaneously, we find that $C=3$ and $D=-3$.
Therefore, the product $CD=3\cdot(-3)=\boxed{-9}$.
Response 3:
To find C and D, we can substitute $x=-8$ and $x=3$ into the equation $C(x+8)+D(x-3)=4x-23$.
For $x=-8$:

$$C(-8+8)+D(-8-3)=4(-8)-23$$
$$-11D=-55$$
$$D=5$$

For $x=3$:

$$C(3+8)+D(3-3)=4(3)-23$$
$$11C=-13$$
$$C=-\frac{13}{11}$$

Now we can answer the question: What is the product CD of the integers C and D?

$$CD=\left(-\frac{13}{11}\right)(5)=-\frac{65}{11}$$

The answer is $-\frac{65}{11}$.
Response 4:
To find C and D, we can set $x=3$ and $x=-8$ to get two equations:
1. $C(3+8)+D(3-3)=4(3)-23\Rightarrow 11C=-11\Rightarrow C=-1$
2. $C(-8+8)+D(-8-3)=4(-8)-23\Rightarrow -11D=-55\Rightarrow D=5$
The answer is $C=-1$ and $D=5$.

An Example ICL Demonstration of MATH-500 for Aggregators: Assistant Response

Assistant:
Thoughts:
1. The responses generally attempt to solve the equation $C(x+8)+D(x-3)=4x-23$ by substituting specific values for x, namely $x=3$ and $x=-8$.
2. The correct method involves creating a system of linear equations by substituting these values and solving for C and D.
3. There is a discrepancy in the values of C and D obtained in the responses:
- Some responses correctly find $C=-1$ and $D=5$.
- Another response incorrectly finds $C=-13/11$ and $D=5$.
- Another response finds $C=3$ and $D=-3$, which is not consistent with the given equation.
4. The correct values should satisfy both equations derived from the substitutions.
Synthesized Response:
The equation $C(x+8)+D(x-3)=4x-23$ can be solved by substituting specific values for x:
1. Substituting $x=3$:

$$C(3+8)+D(3-3)=4(3)-23$$
$$11C=-11$$
$$C=-1$$

2. Substituting $x=-8$:

$$C(-8+8)+D(-8-3)=4(-8)-23$$
$$-11D=-55$$
$$D=5$$

Thus, the values of C and D are $C=-1$ and $D=5$. The product CD is:

$$CD=(-1)\cdot 5=-5$$

The answer is: -5

An Example ICL Demonstration of StrategyQA for Aggregators: User Query

User:
[Here is the user query that was fed into those AI assistants]:
Given a question, please decompose it into sub-questions. For each sub-question, please answer it in a complete sentence. When the original question is answerable, please start the sub-question with "Now we can answer the question: <original question>".
Question 5: Johnny Carson: American talk show host and comedian. Could Johnny Carson's children fill out a water polo team?
Question 5.1: How many children did Johnny Carson have?
Answer 5.1: Johnny Carson had three children: two daughters, Catherine and Christie, and a son, Jeff.
Question 5.2: How many players are needed to form a water polo team?
Answer 5.2:
[Here are the responses from various AI assistants]:
Response 1:
A standard water polo team consists of 13 players.
Response 2:
A water polo team consists of seven players, including one goalkeeper and six field players.
Response 3:
A water polo team consists of 13 players: 1 goalkeeper and 12 field players (7 field players and 5 substitutes).
Response 4:
A standard water polo team consists of 13 players, including 7 field players and 6 substitutes.

An Example ICL Demonstration of StrategyQA for Aggregators: Assistant Response

Assistant:
Thoughts:
1. The responses contain valuable information about the composition of a water polo team.
2. There is a consensus that a water polo team consists of 13 players in total. However, there is some variation in how these 13 players are categorized.
3. The key point to clarify is how many players are in the water at one time and how many are substitutes.
4. It appears that in the water during play, there are 7 players (including the goalkeeper), and the remaining players are substitutes.
Synthesized Response:
The synthesized response is:
To form a water polo team, a total of 13 players are required. This team includes 7 players who are in the water during play, comprising one goalkeeper and six field players. The remaining 6 players serve as substitutes.