How Far are LLMs from Real Search? A Comprehensive Study on Efficiency, Completeness, and Inherent Capabilities

Anonymous ACL submission

Abstract

Search plays a fundamental role in problem-solving across various domains, with most real-world decision-making problems being solvable through systematic search. Drawing inspiration from recent discussions on search and learning, we systematically explore the complementary relationship between search and Large Language Models (LLMs) from three perspectives. First, we analyze how learning can enhance search efficiency and propose **Search via Learning** (SEAL), a framework that leverages LLMs for effective and efficient search. Second, we further extend SEAL to SEAL-C to ensure rigorous completeness during search. Our evaluation across three real-world planning tasks demonstrates that SEAL achieves near-perfect accuracy while reducing search spaces by up to 99.1% compared to traditional approaches. Finally, we explore how far LLMs are from real search by investigating whether they can develop search capabilities independently. Our analysis reveals that while current LLMs struggle with efficient search in complex problems, incorporating systematic search strategies significantly enhances their problem-solving capabilities. These findings not only validate the effectiveness of our approach but also highlight the need for improving LLMs' search abilities for real-world applications.

1 Introduction

Search lies at the heart of problem-solving, offering a systematic approach to explore solution spaces and find optimal answers. From everyday choices to complex strategic planning, virtually all real-world decision-making processes can be formulated and solved through systematic search strategies. This insight aligns with the recent discussions on search and learning (Sutton, 2019; Snell et al., 2024), which emphasizes that the most effective problem-solving approaches combine systematic search with learning from experience.

Traditional search methods, such as brute-force searches, while theoretically complete, face challenges in systematically exploring large and complex search spaces. They are designed to ensure that all possible states are considered, but the vastness of such spaces often necessitates exhaustive exploration, making systematic traversal impractical. Moreover, these methods lack the intuitive problem-solving abilities that humans naturally employ, such as recognizing promising solutions early or quickly abandoning unproductive paths.

Recent advances in Large Language Models (LLMs) and Large Reasoning Models (LRMs) have opened new possibilities for more human-like search approaches. Some recent LRMs such as OpenAI o1 (OpenAI, 2024), QwQ-32B (Qwen, 2024b), and DeepSeek-R1 (Guo et al., 2025) have demonstrated remarkable performance by incorporating LLM-guided search strategies. These approaches leverage LLMs' extensive knowledge bases and reasoning capabilities to guide the search process (Wei et al., 2022; Wang et al., 2023; Snell et al., 2024; Yao et al., 2023a; Hao et al., 2023; Wang et al., 2024a), attempting to mirror human-like intuition in problem-solving.

Despite the great success of these LLM-based simulated searches, a critical limitation remains: they rely heavily on models' intrinsic knowledge rather than combining it with systematic search strategies. When confronted with complex problems requiring multi-step reasoning or extensive exploration, these models often struggle to maintain consistent performance (Wang et al., 2024b,c; Snell et al., 2024). Their reasoning can become unstable or incomplete, particularly when solutions require carefully exploring multiple solution paths or backtracking from dead ends—abilities that humans naturally employ in problem-solving.

Draw inspiration from recent discussions on search and learning (Sutton, 2019; Snell et al., 2024), which emphasizes the great power of combining search and learning over human-centric approaches, in this paper, we systematically explore the integration and complementation of search and LLMs from three crucial perspectives: efficiency, completeness, and inherent search capabilities.

Firstly, we explore how learning can benefit search. Specifically, we conduct a preliminary analysis to compare existing traditional and LLM-based search methods on a representative task, Game of 24, to investigate their problem-solving capabilities. Our experimental results reveal that the learning and reasoning capabilities of LLMs help solve simpler problems without extensive searching, reducing unnecessary state exploration and prioritizing promising states, which significantly shrink search spaces. Then, building upon these insights, we present SEAL, a framework that integrates learning into search algorithms to improve the search efficiency while maintaining completeness for enhancing the problem-solving capabilities of LLMs. We also introduce SEAL-C, a variant that rigorously ensures search completeness while preserving efficiency through learning-guided complete state decomposition and two-phase ranking.

We evaluate SEAL and SEAL-C on three planning tasks: Game of 24, Mini Crosswords (Yao et al., 2023a) and Blocksworld (Valmeekam et al., 2022) using five representative LLMs, GPT-4o (Hurst et al., 2024), GPT-4o-mini (Achiam et al., 2023), Qwen2.5-72B-Instruct (Qwen, 2024a), QwQ-32B-Preview (Qwen, 2024b) and DeepSeek-R1 (Guo et al., 2025). Our experimental results show that SEAL reaches almost perfect pass rates across almost all settings, while reducing search space by up to 99.1% compared to traditional brute-force searches. And SEAL-C is also proved to ensure rigorous completeness efficiently. These validate the effectiveness of SEAL in enabling complete and efficient search via learning.

To this end, we investigate how far LLMs are from real search by investigating a reverse but natural problem: *how search can benefit LLMs and whether LLMs can learn to search by themselves*. Specifically, we prompt LLMs to conduct searches solely relying on intrinsic knowledge or guided by SEAL's search strategies, respectively. Our analysis yields two significant insights. First, while search capabilities are crucial for LLMs' problem-solving effectiveness, current LLMs/LRMs exhibit inefficient search behaviors, requiring extensive sampling to achieve satisfactory performance. Second, incorporating SEAL's search strategies into LLM prompts demonstrably enhances their problem-solving capabilities. These findings not only underscore the critical role of search in enhancing LLMs' reasoning and learning capabilities but also validate the effectiveness of SEAL's search strategies, motivating us to improve LLMs' self-search capabilities in the future works. Our **main contributions** are:

- (**Analysis**) Inspired by the principles of search and learning, we conduct a systematic exploration of how learning benefits search, demonstrating that LLMs can reduce unnecessary state exploration and improve efficiency by prioritizing promising states. Additionally, we explore how search benefits LLMs, revealing the importance of teaching LLMs to efficiently search for solving complex problems.
- (**Methodology**) We propose SEAL, a framework that integrates learning into search for efficiency and completeness, and SEAL-C, a variant to ensure rigorous completeness.
- (**Experiments**) We evaluate SEAL and SEAL-C across diverse real-world tasks, demonstrating their effectiveness in achieving efficient and complete search. We also reveal that existing LLMs cannot perform efficient and effective search regarding complex tasks, which is a desired fundamental ability for LLMs to be applied to real-world decision making tasks.

2 Related Works

LLM-based Search Methods. Recent advancements in test-time compute scaling have sparked growing interest in methods that enable LLMs to simulate search processes and "think longer" instead of directly generating answers in one pass. Several works (Wang et al., 2023; Hao et al., 2023; Feng et al., 2023; Yao et al., 2023a; Zhao et al., 2024; Besta et al., 2024; Wang et al., 2024a; Snell et al., 2024) adopt such approaches to enhance LLMs' problem-solving capabilities. Despite their innovation, these methods rely solely on LLMs' intrinsic knowledge, often leading to unstable performance due to limitations in LLMs' reasoning capabilities. In contrast, our proposed SEAL integrates LLMs with traditional search strategies, ensuring both completeness and efficiency in solving decision-making tasks. More LLM-based search methods are reviewed in Appendix A.1.

Traditional Search Methods. Inspired by the recent discussion about search and learning (Sutton, 2019), which underscores the enduring value of general-purpose strategies that scale with computational power, we investigate how to better integrate search with learning to leverage the strengths of both paradigms. Our SEAL and SEAL-C draw inspiration from traditional search while incorporating LLM-guided reasoning to reduce search space, thereby significantly enhancing efficiency without sacrificing completeness. More details of the related works are in Appendix A.

3 How Learning can Benefit Search

Search and learning represent two fundamental approaches to problem-solving. Traditional search methods offer systematic exploration with guaranteed completeness, while learning-based approaches leverage pattern recognition to identify promising solutions quickly. In this section, we conduct a preliminary analysis to systematically explore existing search algorithms and the synergies between learning and search. Building on these insights, we introduce SEAL, a framework that integrates learning into search algorithms to reduce unnecessary exploration, prioritize promising paths, and ultimately maintain reliable and efficient performance as problems grow in complexity.

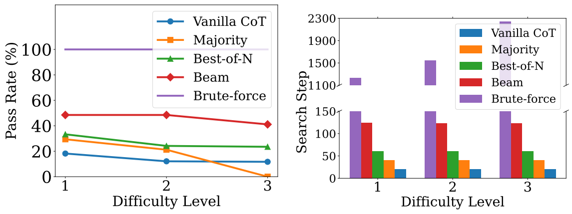
3.1 Experimental Setup

Task Setup. To investigate the impact of learning on search, we use *the Game of 24* as a representative task. This task can be solved using traditional search algorithms and is also widely adopted for evaluating LLMs' planning abilities. Details about the task are provided in Appendix G.1. Following the setting in Yao et al. (2023a), we select 100 problems indexed as $900 - 999$ for our experiments.
Baselines. For our preliminary analysis, we evaluate three LLM-based simulated search algorithms: MAJORITY VOTE (Wang et al., 2023), BEST-OF-N (Snell et al., 2024), and BEAM SEARCH (Yao et al., 2023a). Additionally, we include VANILLA COT (Chain-of-Thought) (Wei et al., 2022) as a reference. To evaluate search efficiency, we also consider two traditional brute-force search methods, Depth-First Search (DFS) and Breadth-First Search (BFS), along with their pruning variants, DFS-PRUNE and BFS-PRUNE. These variants improve efficiency by avoiding exploration of previously visited states. Finally, an EXHAUSTIVE SEARCH is included as a baseline for comprehensive comparison. Detailed implementation details for all baselines are provided in Appendix G.2.
Evaluation Metrics We evaluate performance using two primary metrics: (i) pass rates (PR) across games per difficulty level, measuring solution quality, and (ii) search steps (SS), measuring exploration efficiency by counting traversed states in the search space $\mathcal{S}$. Complete metric definitions are available in Appendix G.4.

3.2 Analysis: How Learning can Benefit Search

To better understand how learning can enhance search, we divide the 100 problems from the Game of 24 into three difficulty levels based on human success rates[1]. Other experimental settings follow those in Sec. 3.1. Our experimental results, presented in Fig. 1, reveal several key findings regarding the performance of existing search methods. Full analyses are in Appendix B.1.
Obs. 1: LLMs Perform Better for Simpler Problems. As in Fig. 1(a), VANILLA COT achieve pass rates of 18.2%, 12.1%, and 11.7% for problems with difficulty levels 1, 2, and 3, respectively. This pattern suggests that LLMs excel at direct problem-solving for simpler cases, indicating their potential for single-step solutions rather than requiring iterative approaches. More examples are in Fig. 5.
Obs. 2: Learning-Based Pruning Has Precision-Coverage Trade-offs. According to Fig. 1(a) and 1(b), BEAM SEARCH achieves the superior performance in pass rate than other LLM-based search methods, and significantly saves search steps than DFS. To deeply analyze this observation, an example of BEAM SEARCH in Fig. 6 shows that it only relies on LLMs to generate a few of possible next steps instead of decomposing all possible states in brute-force searches in each time of generating next intermediate steps. However, it is possible that some valid states are overlooked during the intermediate step generation, leading to failure in solving tasks and reflecting a trade-off between efficiency and completeness in search.
Obs. 3: Learning Provides Adaptive Search Guidance. Fig. 7 is an example of BEAM SEARCH employing an LLM verifier to assess intermediate states' progress toward the goal, prioritizing exploration of valid states and avoiding further exploration on other states. Moreover, as in Fig. 1(a), the LLM-based methods using more search steps generally perform better. This is because these

[1] https://www.4nums.com/game/difficulties/



(a) Pass Rate (b) Search Step

Figure 1: PR (%) and SS of existing searches across various problem difficulties using GPT-4o-mini.

Table 1: PR (%) and SS on the Game of 24 using Qwen2.5-72B-Instruct. For LLM-based methods, we additionally present the reduction ratio of SS compared to BRUTE-FORCE (DFS).

Method	PR (%)	SS
EXHAUSTIVE SEARCH	100	12,928
BRUTE-FORCE (DFS)	100	1,623
BRUTE-FORCE (BFS)	100	3,429
BRUTE-FORCE (DFS-PRUNE)	100	1,385
BRUTE-FORCE (BFS-PRUNE)	100	1,306
VANILLA COT	17	1 (↓ 99.9%)
MAJORITY VOTE	23	10 (↓ 99.3%)
BEST-OF-N	27	20 (↓ 98.7%)
BEAM SEARCH	35	124 (↓ 92.4%)

methods allocate more search budget in verifying results via LLMs, further demonstrating LLMs' capability as dynamic evaluators to leverage learned knowledge to identify promising states and guide the search process.

These findings suggest that learning can substantially improve search efficiency when properly integrated into the search process. However, they also highlight the need for careful mechanism design to balance the benefits of learning-based pruning with the completeness guarantees of traditional search.

3.3 Analysis: Existing Search Algorithms

While existing search algorithms show significant promise, they still encounter notable challenges. To better illustrate their limitations, we conduct an analysis following the setup in Sec.3.1 by applying existing search algorithms to the Game of 24 task. The results are reported in Table 1, revealing the following insights: **(i)** Traditional brute-force algorithms achieve perfect 100% accuracy but require up to 3,429 search steps, demonstrating their inefficiency in searching for answers. **(ii)** While LLM-based methods significantly reduce the search space (at least 92.4% fewer steps than DFS), their pass rates peak at only 35%, highlighting their instability compared to traditional searches. Full analyses are in Appendix B.3

These findings underscore the necessity of a framework that combines the completeness of systematic search with the efficiency of learning-based approaches. This motivates the development of a more effective search-and-learning methodology to address these limitations.

3.4 SEAL: Search via Learning with LLMs

Notations. We focus on solving planning problem following the essence of solving real-world decision-making problems via search: starting from an initial state, envisioning possible actions, and systematically working toward a goal state. Specifically, a planning problem is formally defined as a tuple $P = \langle \mathcal{S}, s^{init}, s^{goal}, \mathcal{A}, f \rangle$. Here, $\mathcal{S}$ represents a finite and discrete set of states describing the world (i.e., state space). $s^{\text{init}}, s^{\text{goal}} \in \mathcal{S}$ denote the initial and goal world states, respectively. $\mathcal{A} = \{a_1, a_2, \dots, \}$ represents the set of possible actions, and $f(s, a_i) = s'$ is a transition function mapping a state and action to a resulting state. A solution to problem P is a sequence of actions $\langle a_1, a_2, \dots \rangle$ that transforms s^{init} into s^{goal}.

Building upon the above insights, we introduce SEAL, a framework that systematically integrates learning capabilities into search processes to emulate human problem-solving strategies. The goal of SEAL is on enhancing this natural problem-solving process by combining the systematic nature of search with the learning capabilities of LLMs. SEAL consists of four components to enhance different aspects of search processes through learning:

Direct Solution Generation. Motivated by the insight from Obs. 1 that learning excels at solving simpler problems directly, we begin each step by attempting a direct solution, akin to how humans first try to solve problems in one step:

$$r^{cur} = M(p_{\text{solve}}(s^{cur})) \quad (1)$$

where M represents the backbone LLM and p_{solve} denotes the solution generation prompt. A verifier f validates the generated solution. More details of p_{solve} are in Appendix D.1. This mechanism leverages LLMs' strength in handling simpler cases, potentially bypassing the need for extensive search when direct solutions are viable. Note that, however, the verifier f may not always be available in certain scenarios. To ensure the applicability of SEAL in such cases, we provide further discussions in Appendix I.2.

State Decomposition. For more complex problems that cannot be solved directly, we decompose the current state s^{cur} into subproblems, mirroring how humans break down challenging tasks:

$$S^{next} = D(s^{cur}), \tag{2}$$

where D is the decomposition function generating substates $S^{next} = \{s_1^{next}, s_2^{next}, \ldots\}$. While LLMs can suggest promising decompositions (Yao et al., 2023a), Obs. 2 reveals the risk of overlooking valid paths. Thus, we maintain the ability to systematically enumerate substates when needed, balancing between efficient exploration of valid paths and thorough coverage of the state space $\mathcal{S}$.

State Validity Checking. During search, it is inevitable that we may explore some invalid states, especially if we decompose into all possible substates. Exploring and expanding these invalid substates too much will significantly increase the search space. Thus, inspired from Obs. 3, we leverage LLMs to assess the potential of substates toward goals, similar to how humans quickly judge if it is worthy to continue exploration:

$$c^{next} = M(p_c(S^{next})), \tag{3}$$

where p_c prompts the LLM M to evaluate state validity. Additional examples p_c are in Appendix D.2. In this paper, we opt for binary decisions (c_k^i) about whether to continue exploration on the substates S^{next}, ensuring efficient pruning on invaid states while maintaining search on valid states.

Learning-guided Ranking After validating substates, we prioritize paths similar to how humans naturally focus their attention on the most promising approaches. Leveraging Obs. 3, we employ LLMs to compute a priority score of each state:

$$v(s^{cur}) = M(p_v(s^{cur})) \tag{4}$$

where $v(s^{cur})$ denotes the LLM-estimated value using prompt p_v, indicating how likely the current state leads to the goal state s^{goal}. States with higher scores are prioritized in the exploration queue, enabling efficient traversal of promising solution paths. Details of p_v are in Appendix D.3.

Overall Alogirhtm Combining these components together into search, we implement the framework of SEAL, which is shown in Fig. 2. The full algorithm is in Appendix C. Our search strategy focuses on exploring promising paths deeply before considering alternatives, similar to how humans naturally approach problem-solving. This design allows SEAL to scale to complex problems effectively and efficiently through its combination of learning-guided intuition and systematic exploration.

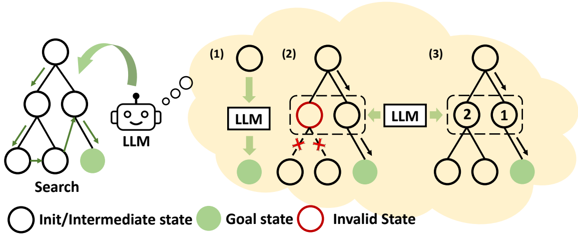


Figure 2: SEAL intergrating learning into search with LLMs: **(1)** Direct solution generation, **(2)** State validity checking, **(3)** Learning-guided state ranking.

4 Toward Complete Search via Learning

While SEAL effectively balances efficiency and effectiveness, our observation indicates potential compromises in solution accuracy. In high-stakes domains such as autonomous driving (Mao et al., 2023) and pandemic response planning (Du et al., 2024), ensuring completeness is fundamental to search algorithms, as overlooking any viable solution can have severe consequences. This motivates us to develop a search framework that rigorously ensures completeness. Next, we formalize search completeness and introduce SEAL-C, an enhanced variant of SEAL that integrates efficient learning-guided exploration with formal guarantees of completeness.

4.1 Formalizing Search Completeness

We begin by formalizing search completeness–the guarantee of finding a solution when one exists:

Definition 4.1 (Search Completeness)**.** A search algorithm is complete if and only if for any initial state $s^{init} \in \mathcal{S}$ and goal state s^{goal}, whenever there exists a valid solution path $P = (s^0, ..., s^n)$ where $s^n \in s^{goal}$, the algorithm is guaranteed to find it.

4.2 SEAL-C: Achieving Search Completeness

Building on this definition, we first analyze potential completeness compromises in SEAL, then present SEAL-C's mechanisms for ensuring rigorous completeness. The full algorithm of SEAL-C is shown in Alg. 2.

How Can Completeness Be Compromised? According to Sec.3.4, SEAL uses LLMs for state decomposition and validity checking, inspired by Obs. 2 and 3. However, our analysis in Sec.3.2 reveals that they may inadvertently ignore valid states during decomposition or prematurely terminate exploration of valid paths, compromising completeness.

Learning-Guided Complete State Decomposition. To ensure completeness while maintaining efficiency, SEAL-C employs a learning-guided complete state decomposition strategy by combining learning-based prioritization with a fallback mechanism for exhaustive state expansion:

$$S^{next} = M(p_d(s^{cur}))||(D(s^{cur}) \setminus M(p_d(s^{cur}))) \quad (5)$$

where $||$ denotes ordered concatenation, ensuring LLM-generated states $M(p_d(s))$ are explored first. $D(s)$ is the complete state decomposition function from Eq. (2). This approach prioritizes exploration of likely valid states while ensuring no potential solution is overlooked, guaranteeing completeness while benefiting from learning-guided efficiency.

Two-phase Ranking. To further improve efficiency, SEAL-C introduces a two-phase ranking strategy for S^{next}. Instead of ranking all states at once, it first ranks and explores the LLM-generated states $M(p_d(s^{cur}))$, which are more likely to reach s^{goal}. Only when no solution is found does the algorithm proceed to rank and explore the remaining states from $D(s^{cur})$, which significantly reduces the search space by avoiding unnecessary ranking of supplementary states.

5 Can LLMs Learn to Search by Themselves?

Having integrated learning to enhance search, a reverse but natural question emerges: *Can LLMs execute search autonomously to improve the reasoning capabilities?* Recent advances in scaling test-time computation suggest that systematic exploration could enhance LLMs' problem-solving abilities. While models like QwQ-32B (Qwen, 2024b) show promising reasoning capabilities, they struggle with focused problem-solving, often producing unfocused, recursive outputs. This motivates us to explore how search benefits LLMs and whether LLMs possess the potential for self search.

To systematically investigate LLMs' self-search capabilities, we consider two types of prompts: high-level self-search, which relies solely on LLMs' internal knowledge, and low-level self-search, which explicitly encodes SEAL's search strategies into the prompts. Further details and illustrative examples are in Appendix E, with additional discussions and analyses provided in Sec. 6.4.

6 Experiments

In this section, we conduct experiments to answer the following research questions: (**RQ1**) How effectively and efficiently does SEAL solve tasks compared to existing search methods? (**RQ2**) How efficient is SEAL when conducting a rigorously complete search? (**RQ3**) Do LLMs inherently possess the capability for self-search?

6.1 Experiment Settings

Tasks. To evaluate the effectiveness of SEAL, in addition to the Game of 24 used in Sec. 3.1, we select two widely adopted planning tasks for evaluating LLMs' planning abilities: Mini Crosswords (Yao et al., 2023a) and Blocksworld (Valmeekam et al., 2022). Further details on these tasks are provided in Appendix G.1.

Baselines. Following the settings in Sec. 3.1, we consider four LLM-based searches, VANILLA COT, MAJORITY VOTE, BEST-OF-N, and BEAM SEARCH, and three traditional searches, DFS, BFS and EXHAUSTIVE SEARCH. We also involve a baseline BEAM SEARCH+RV that adds a rule-based verifier to the beam search method. Details on the baselines are provided in Appendix G.2.

Models. Our experiments use both closed-source models (GPT-4o-mini, GPT-4o (Hurst et al., 2024)) and open-source models (Qwen2.5-72B-Instruct (Qwen, 2024a), QwQ-32B-Preview (Qwen, 2024b), DeepSeek-R1 (Guo et al., 2025)). Note that GPT-4o-mini is a small language model (SLM), and QwQ-32B-Preview and DeepSeek-R1 are the state-of-the-art LRMs. Model and implementation details are in Appendix G.3 and G.5.

Evaluation Metrics. We follow Sec. 3.1 to use two metrics: (i) pass rates (PR) and (ii) search steps (SS). More details are in Appendix G.4.

6.2 RQ1: Effectiveness and Efficiency Evaluations

We conduct experiments on the three tasks using three different LLM backbones: GPT-4o-mini, GPT-4o, and Qwen2.5-72B-Instruct. Detailed task setups are in Appendix H. The results are reported in Table 2. The key observations are: **(i)** SEAL significantly reduces search steps compared to brute-force searches. Specifically, it reduces search steps by up to 99.1% compared to DFS and still achieves state-of-the-art search steps compared to other LLM-based methods. This validates SEAL's efficiency in navigating the search space. **(ii)** SEAL achieves a near-perfect pass rate across all settings, outperforming other LLM-based methods. BEAM SEARCH+RV achieves better pass rates than BEAM SEARCH, indicating that integrating rule-based verifiers into LLM-based methods enhances their reliability in problem-solving. However, it still falls

Table 2: Results of different search methods across three tasks. Specifically, the results of LLM-based search baselines are reported as the average values across three LLMs. "TL" indicates that the number is too large. We also highlight the SS results of SEAL that are comparable to state-of-the-art performance (marked in green).

Search Method	Game of 24		Mini Crosswords		Blocksworld	
	PR (%)	Avg. SS	PR (%)	Avg. SS	PR (%)	Avg. SS
Traditional Search						
Exhaustive Search	100	12,928	100	TL	100	TL
Brute-force (DFS)	100	1,623	100	4,128.9	100	18,531.9
Brute-force (BFS)	100	3,429	100	TL	100	96,759.4
LLM-based Search						
Vanilla CoT	14.3	1 (↓ 99.9%)	1.7	1 (↓ 99.9%)	17.5	1 (↓ 99.9%)
Majority Vote	20.3	10 (↓ 99.3%)	3.4	10 (↓ 99.7%)	47.4	10 (↓ 99.9%)
Best-of-N	22.6	20 (↓ 98.7%)	2.5	20 (↓ 99.5%)	30.0	20 (↓ 99.8%)
Beam Search	54.0	94.8 (↓ 94.1%)	45.0	26.1 (↓ 99.4%)	0	64.4 (↓ 99.6%)
Beam Search+RV	79.0	98.8 (↓ 93.9%)	90.0	30.9 (↓ 99.2%)	0	94.8 (↓ 99.6%)
SEAL						
– GPT-4o-mini	**100**	40 (↓ 97.5%)	**100**	75.8 (↓ 98.2%)	**100**	160.6 (↓ 99.1%)
– GPT-4o	99	65.6 (↓ 96.0%)	**100**	45.0 (↓ 98.9%)	**100**	80.8 (↓ 99.5%)
– Qwen2.5-72B-Instruct	**100**	84.9 (↓ 94.8%)	**100**	98.3 (↓ 98.9%)	**100**	68.7 (↓ 99.6%)

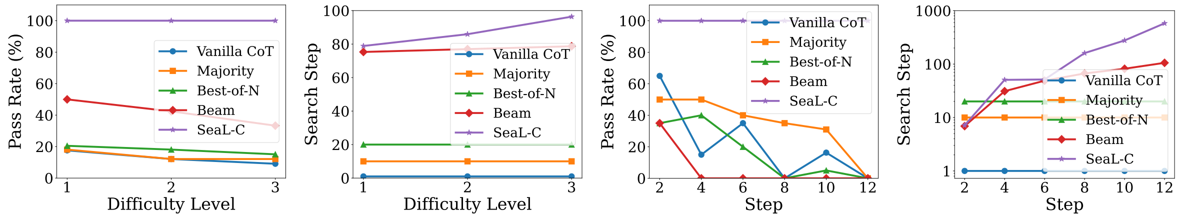
behind SEAL, confirming that current LLM-based methods do not fully conduct real search. This further highlights the superior search effectiveness of SEAL. Full search results and the discussion of SEAL with various verifiers are in Appendix I.

6.3 RQ2: Impact of Problem Difficulty on Search Completeness

We evaluate SEAL-C's performance across problem difficulties in three tasks. For Game of 24, we use three difficulty levels (Sec. 3.2); for Blocksworld, difficulty scales with minimum required action steps (2–12). Other settings follow Sec. 6.1. The results using GPT-4o-mini in the two tasks are reported in Fig. 3, which show that: **(i)** SEAL-C achieves 100% pass rates across all difficulty levels, outperforming baseline methods whose performance degrades progressively, confirming its completeness. **(ii)** As the difficulty increases, SEAL-C's search steps increase slightly but remain significantly lower than brute-force baselines. This efficiency is attributed to SEAL-C's learning-guided complete state decomposition and two-phase ranking, which prioritize promising states and effectively reduce the search space. Mini Crosswords results are in Appendix J.

6.4 RQ3: Potential of LLMs in Self-Search

We conduct an initial exploration to study whether LLMs can learn to self-search for problem-solving by testing two self-search prompts in Sec. 5 and Vanilla CoT in GPT-4o and two state-of-the-art open-sourced LRMs, QwQ-32B-Preview and DeepSeek-R1 to demonstrate the importance of search in solving tasks. Especially, we use the chat mode of DeepSeek-R1 and sample 20 problems. Since they are LRMs that are capable of searching, the results of Self-Search (High) in the two LRMs can be directly obtained from using Vanilla CoT in this model. The comparison results in Table 3 show: **(i)** Standard prompting (Vanilla CoT) in GPT-4o achieves the lowest performance, indicating LLMs cannot conduct searches and require conduct searches for complex problem-solving. **(ii)** Two LRMs achieve the best performance across all settings. Specifically, DeepSeek-R1, when using Self-Search (High), achieves an impressive pass rate of 85%. From our observations, these LRMs perform search by iteratively sampling and evaluating different answers until a correct solution is found. This underscores the importance of search in enhancing LLM problem-solving capabilities while also revealing the inefficiency of current search mechanisms, motivating the need for more efficient strategies. More details of the problem-solving processes of the two LRMs are in Appendix K. **(iii)** Both self-search approaches significantly improve performance, with low-level self-search achieving up to a 95% pass rate. This demonstrates that LLMs have the potential of effectively utilizing explicit search strategies for improving problem-solving. **(iv)** Low-level self-search, which incorporates SEAL's strategies, consistently outperforms high-level self-search across both models. This validates the effectiveness of SEAL's search and suggests that better search guidance enhances LLM reasoning.



(a) Game of 24 - PR (b) Game of 24 - SS (c) Blocksworld - PR (d) Blocksworld - SS

Figure 3: Impact of problem difficulty on search completeness in Game of 24 and Blocksworld with GPT-4o-Mini.

Table 3: PR(%) of search strategies for GPT-4o, QwQ-32B-Preview and DeepSeek-R1 on the Game of 24.

Model	Method	PR(%)
GPT-4o	VANILLA COT	13
	SELF-SEARCH (HIGH)	24
	SELF-SEARCH (LOW)	**31**
QwQ-32B-Preview	SELF-SEARCH (HIGH)	62
	SELF-SEARCH (LOW)	**70**
DeepSeek-R1 (20)	SELF-SEARCH (HIGH)	85
	SELF-SEARCH (LOW)	**95**

These findings highlight both the importance of search in enhancing LLM capabilities and the effectiveness of SEAL's search strategies, motivating us to explore how to further develop LLMs' self-search capabilities and optimize search strategies, which will be our future works.

6.5 Ablation Studies

Inspired by Snell et al. (2024), we conduct ablation studies to understand the impact of search budgets on SEAL's performance. Instead of terminating the search upon finding the final state, we introduced a pre-defined search step budget, where SEAL terminates early if the budget is reached. We vary the search step budgets as $\{10, 20, 30, 50, 100, 150, 200\}$, and compare various search methods using GPT-4o-mini in Game of 24. The comparison results are reported in Fig. 4. Our analysis reveals three key findings: **(i)** SEAL consistently outperforms other methods across all search budgets, demonstrating its effectiveness in accurately solving problems even under constrained search budgets. **(ii)** Pass rates for all search methods generally improve as the search budget increases, aligning with expectations that scaling test-time computation enhances search performance. **(iii)** When the search budget is small (less than 50 steps), BEAM SEARCH performs poorly, achieving a 0% pass rate. This is due to its sequential evaluation of substates, which consumes substantial search budgets on generating and assessing intermediate substates. In contrast, SEAL focuses on diving directly toward the goal state, significantly reducing the search space and validating its efficiency in navigating complex search problems. Additional results from the ablation studies evaluating the impact of SEAL's components can be found in Appendix L.

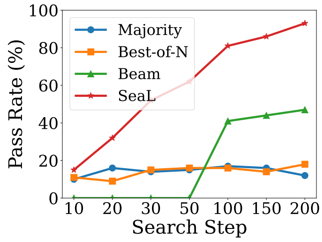


Figure 4: Impact of SS in SEAL using GPT-4o-mini.

7 Conclusion and Future Work

In this paper, inspired by the principles of search and learning (Sutton, 2019; Snell et al., 2024), we systematically investigate the integration of learning into search. We first explore how learning benefits the search process via LLMs, demonstrating that LLMs improve search efficiency by reducing search space. Then, building on these insights, we introduce a novel framework, SEAL, and its variant SEAL-C, designed to combine the reasoning capabilities of LLMs with search strategies to achieve efficient and accurate problem-solving. Extensive experiments conducted on three real-world planning tasks demonstrate that SEAL achieves near-perfect pass rates across various settings while significantly reducing search spaces, showcasing the effectiveness and efficiency of SEAL. Furthermore, we also explore how search can benefit LLMs, evaluating whether LLMs can develop self-search capabilities. We show that search significantly enhances their reasoning and learning performance. These findings highlight the bidirectional synergy between search and learning, emphasizing the potential of integrating search into LLMs.

Our research paves the way for further exploration into the convergence of search and learning. Future work will focus on investigating how LLMs can better conduct self-searches, further unlocking their potential for complex problem-solving.

8 Limitations

One potential limitation of this work is the necessity of encoding our search strategies into prompts to enable LLMs to perform self-search. In the future, it is worth exploring how to allow LLMs to autonomously conduct self-search during reasoning without explicit supervision within the prompts. Additionally, our work primarily focuses on LLMs; an important direction for future investigation is to assess the applicability of our search strategy to multi-modal LLMs.

9 Impact Statements

This paper introduces SEAL and SEAL-C, frameworks designed to enhance search processes by integrating the reasoning capabilities of LLMs with structured search strategies. The potential societal impacts of this research are broad but largely align with the established consequences of improving computational problem-solving techniques. Enhanced search efficiency and the ability to leverage LLMs for self-search could benefit applications in diverse fields such as healthcare, logistics, education, and robotics, where intelligent decision-making is crucial. However, as with all advancements in AI, there are ethical considerations. For example, the misuse of improved search strategies in domains such as automated surveillance or adversarial systems could lead to privacy or security concerns. We encourage researchers and practitioners to apply this work responsibly and ensure it aligns with ethical guidelines. We see no immediate risks or unintended negative consequences specific to this work that require urgent attention. This paper primarily contributes to foundational research in search and learning integration. Future exploration of self-search capabilities in LLMs will include careful assessment of ethical implications to ensure responsible development and deployment of these technologies.

References

Asma Ben Abacha, Wen-wai Yim, Yujuan Fu, Zhaoyi Sun, Meliha Yetisgen, Fei Xia, and Thomas Lin. 2024. Medec: A benchmark for medical error detection and correction in clinical notes. *arXiv preprint arXiv:2412.19260*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michał Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.

Daniel Cao, Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi. 2024. Automating thought of search: A journey towards soundness and completeness. *arXiv preprint arXiv:2408.11326*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yves Crama, Antoon WJ Kolen, and EJ Pesch. 2005. Local search in combinatorial optimization. *Artificial Neural Networks: An Introduction to ANN Theory and Practice*, pages 157–174.

Hongru Du, Jianan Zhao, Yang Zhao, Shaochong Xu, Xihong Lin, Yiran Chen, Lauren M Gardner, and Hao Frank Yang. 2024. Advancing real-time pandemic forecasting using large language models: A covid-19 case study. *arXiv preprint arXiv:2404.06962*.

Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Richard Howey and Derek Long. 2003. The automatic validation tool for pddl2.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Ana Jojic, Zhen Wang, and Nebojsa Jojic. 2023. Gpt is becoming a turing machine: Here are some ways to program it. *arXiv preprint arXiv:2303.14310*.

Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi. 2024. Planning with language models through the lens of efficiency. *NeurIPS*.

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*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *IJCNLP-AACL*, pages 305–329.

Jiageng Mao, Yuxi Qian, Junjie Ye, Hang Zhao, and Yue Wang. 2023. Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415*.

OpenAI. 2024. Openai o1 system card.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2023. Refiner: Reasoning feedback on intermediate representations. In *EACL 2024, Malta*.

Qwen. 2024a. Qwen2.5: A party of foundation models.

Qwen. 2024b. Qwq: Reflect deeply on the boundaries of the unknown.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20256–20264.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Richard Sutton. 2019. The bitter lesson. Blog post. Accessed: Augsust 2024.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

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

Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, Haitao Mi, Jinsong Su, and Dong Yu. 2024a. Litesearch: Efficacious tree search for llm. *arXiv preprint arXiv:2407.00320*.

Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. 2024b. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*.

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*.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. 2024c. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. *arXiv preprint arXiv:2402.02658*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*.

Hengjia Xiao and Peng Wang. 2023. Llm a*: Human in the loop large language models enabled a* search for robotics. *arXiv preprint arXiv:2312.01797*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of Thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *ICLR*.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2024. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36.

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*.

Appendix Overview

- **Appendix A**: Related Work.
- **Appendix B**: More details of Preliminary Analysis.
- **Appendix C**: Algorithm.
- **Appendix D**: Additional Details of SEAL and SEAL-C.
- **Appendix E**: Additional Details of Self-Search.
- **Appendix F**: Code.
- **Appendix G**: Experimental Settings.
- **Appendix H**: Full Analysis of RQ1: Effectiveness and Efficiency Evaluations.
- **Appendix I**: Additional Experimental Results of SEAL.
- **Appendix J**: Additional Experimental Results of SEAL-C.
- **Appendix K**: Additional Details of Problem-solving Processes of LRMs.
- **Appendix L**: Additional Experimental Results of Ablation Studies.

A Full Details of Related Works

A.1 LLM-based Search Methods

Recent advancements in test-time compute scaling have sparked growing interest in methods that enable LLMs to simulate search processes and "think longer" instead of directly generating answers in one pass. Several works (Wang et al., 2023; Hao et al., 2023; Feng et al., 2023; Yao et al., 2023a; Zhao et al., 2024; Besta et al., 2024; Wang et al., 2024a; Snell et al., 2024) adopt such approaches to enhance LLMs' problem-solving capabilities. Self-Consistency (Wang et al., 2023) employs majority voting over multiple sampled reasoning chains, based on the intuition that the solutions for complex problems are rarely unique. Similarly, Best-of-N (Snell et al., 2024) samples multiple outputs and selects the highest-scoring solution using a learned verifier or reward model (Cobbe et al., 2021; Lightman et al., 2023). RAP (Hao et al., 2023) uses LLMs as heuristic policy functions to guide Monte Carlo Tree Search (MCTS), treating LLMs as world models for state exploration. Tree of Thoughts (ToT) (Yao et al., 2023a) simulates a search tree with LLM-guided node expansion and pruning, while Graph of Thoughts (GoT) (Besta et al., 2024) generalizes this approach by modeling the search space as a graph. Despite their innovation, these methods rely solely on LLMs' intrinsic knowledge, often leading to unstable performance due to limitations in LLMs' reasoning capabilities. In contrast, our proposed SEAL integrates LLMs with traditional search strategies, ensuring both completeness and efficiency in solving decision-making tasks.

A.2 Traditional Search Methods

Traditional search methods have been extensively utilized in tasks such as combinatorial optimization (Crama et al., 2005) and pathfinding (Hart et al., 1968). Among these, brute-force search is regarded as a general yet powerful approach (Sutton, 2019), offering guaranteed solutions through exhaustive exploration of the search space. However, this method also suffers from the large search space in complex problems. Inspired by the recent discussion about search and learning (Sutton, 2019), which underscores the enduring value of general-purpose strategies that scale with computational power, we investigate how to better integrate search with learning to leverage the strengths of both paradigms. Our SEAL and SEAL-C draw inspiration from traditional search while incorporating LLM-guided reasoning to reduce search space, thereby significantly enhancing efficiency without sacrificing completeness.

A.3 LLMs for Planning and Decision-Making

Planning and decision-making involve devising strategic action sequences to achieve predefined goals from given initial states. While classical planning problems rely on algorithms such as brute-force search and A* (Hart et al., 1968) for optimal plan generation, recent advancements demonstrate that LLMs can leverage their extensive commonsense knowledge for planning (Valmeekam et al., 2023; Xiao and Wang, 2023; Yao et al., 2023b). For example, LLM A* (Xiao and Wang, 2023) integrates LLMs into the A* algorithm by using them to generate intermediate waypoints, improving efficiency in robotics and navigation tasks. Other works (Lyu et al., 2023; Jojic et al., 2023; Liu et al., 2023; Katz et al., 2024; Cao et al., 2024) exploit LLMs' programming abilities for planning tasks. For instance, LLM+P (Liu et al., 2023) translates task instructions into Planning Domain Definition Language (PDDL) and solves them using classical algorithms like A*. Thoughts of Search (ToS) (Katz et al., 2024) uses LLMs to generate planning code with human feedback, while AutoToS (Cao et al., 2024) automates this loop by enabling LLMs to validate and revise their own code.

A.4 Reasoning with LLMs

Chain-of-Thoughts (CoT) (Wei et al., 2022) was the first major breakthrough revealing that LLMs can formulate multi-step reasoning processes by using explicit prompts like "Let's think step by step." Follow-up works (Wang et al., 2023; Yao et al., 2023b; Zhou et al., 2023; Welleck et al., 2023; Shinn et al., 2024; Paul et al., 2023) further build on this paradigm. ReAct (Yao et al., 2023b) integrates reasoning with planning, interleaving reasoning steps with dynamic interactions. PAL (Gao et al., 2023) enhances CoT by leveraging LLMs' programming abilities, guiding them to generate executable code during reasoning. Several recent approaches (Welleck et al., 2023; Shinn et al., 2024; Paul et al., 2023) introduce self-evaluation capabilities, enabling LLMs to provide feedback on their intermediate reasoning steps. In this work, we systematically study how LLMs' reasoning capabilities can complement traditional search methods, enabling accurate and efficient problem-solving through our proposed SEAL framework.

B More details of Preliminary Analysis

B.1 Full Analysis: How Learning can Benefit Search

The analysis in Sec.3.3 highlights the limitations of existing search methods and underscores the need for a more effective search-and-learning framework. To better understand how learning can enhance search, we divide the 100 problems from the Game of 24 into three difficulty levels based on human success rates[2]. Other experimental settings follow those in Sec. 3.1. Our experimental results, presented in Fig. 1, reveal several key findings regarding the performance of existing search methods. Overall, we observe that **(i)** the pass rates of LLM-based methods decrease as difficulty increases, and **(ii)** while LLM-based searches require fewer search steps, they generally achieve lower pass rates compared to traditional approaches.

[2]https://www.4nums.com/game/difficulties/

Obs. 1: LLMs Perform Better for Simpler Problems.. Our analysis demonstrates that LLM-based methods achieve notably higher pass rates on less complex tasks. As illustrated in Fig. 1(a), VANILLA COT attains pass rates of 18.2%, 12.1%, and 11.7% for problems with difficulty levels 1, 2, and 3, respectively. This pattern suggests that LLMs excel at direct problem-solving for simpler cases, indicating their potential for single-step solutions rather than requiring iterative approaches. More examples are provided in Fig. 5.

Obs. 2: Learning-Based Pruning Has Precision-Coverage Trade-offs.. Among these LLM-based search methods, BEAM SEARCH achieves superior performance in pass rate. Specifically, according to Fig. 1(a), BEAM SEARCH achieves up to a 48.5% pass rate, while other LLM-based methods only achieve up to a 33% pass rate. In addition, as in Fig. 1(b), BEAM SEARCH significantly saves 92.4% SS than BRUTE-FORCE (DFS). To deeply analyze this observation, we present an example of BEAM SEARCH in solving this task in Fig. 6. From this figure, we observe that in each time of generating next intermediate steps, it only relies on LLMs to generate a few of possible next steps instead of decomposing all possible states in brute-force searches, which significantly reduces the search space and improves search efficiency. However, we also observe from Fig. 6 that it is possible that some valid states are overlooked during the intermediate step generation, leading to failure in solving tasks and reflecting a trade-off between efficiency and completeness in search.

Obs. 3: Learning Provides Adaptive Search Guidance.. As in Fig. 1(a), compared with VANILLA COT, other LLM-based methods perform better. Specifically, VANILLA COT only achieve up to 18.2% pass rate, while MAJORITY VOTE, BEST-OF-N and BEAM SEARCH achieve up to 29.4%, 33.3% and 48.5% prateates, respectively. We analyze that this performance gap is because these LLM-based methods spend more search budget in verifying results via LLMs according to Fig. 1, demonstrating LLMs' capability as dynamic evaluators to leverage learned knowledge to identify promising states and guide the search process. Fig. 7 is an example of BEAM SEARCH leveraging this observation, where BEAM SEARCH employs an LLM verifier to assess intermediate states' progress toward the goal, prioritizing exploration of valid states and avoiding further exploration on other states. However, we also notice that some valid states are overlooked mistakenly. This implies that further improvements are needed to ensure critical states are not omitted.

These findings suggest that learning can substantially improve search efficiency when properly integrated into the search process. However, they also highlight the need for careful mechanism design to balance the benefits of learning-based pruning with the completeness guarantees of traditional search. This motivates our development of a systematic framework for learning-enhanced search.

B.2 Illustrative Examples

Fig. 5, 6, and 7 provide the illustrative examples of the observations presented in Sec. 3.2. The full analyses are provided in Appendix B.1. Detailed explanations of these examples can be found in the respective figure captions. Specifically:

- Fig. 5 illustrates Observation 1 in Sec. 3.2.
- Fig. 6 presents examples of Observation 2 in Sec. 3.2.
- Fig. 7 demonstrates Observation 3 in Sec. 3.2.

B.3 Full Analysis: Existing Search Algorithms

As discussed in Sec. 1, while existing search algorithms show significant promise, they still encounter notable challenges. Traditional search algorithms, while thorough, suffer from inefficiency due to exhaustive exploration of the state space $\mathcal{S}$. They lack the human-like ability to quickly identify promising paths or recognize dead ends. Conversely, LLM-based search methods, while more efficient, struggle with reliability. They heavily rely on LLMs' reasoning capabilities, which can be unstable or incomplete, particularly in complex scenarios requiring careful exploration of multiple solution paths.

To better illustrate these limitations, we conduct an analysis based on the experimental setup in Sec.3.1, applying existing search algorithms to the Game of 24 task. The results are reported in Table 1, revealing the following insights: **(i)** Traditional brute-force algorithms achieve perfect 100% accuracy but require up to $3{,}429$ search steps, demonstrating their inefficiency in searching for answers. **(ii)** While LLM-based methods significantly reduce the search space (at least 92.4% fewer steps than BRUTE-FORCE (DFS)), their pass rates peak at only 35%, highlighting their instability compared to traditional searches.

These findings underscore the necessity of a framework that combines the completeness of systematic search with the efficiency of learning-based approaches. This motivates the development of a more effective search-and-learning methodology to address these limitations.

C Algorithm

The algorithm of SEAL is provided in Alg. 1. This design allows SEAL to systematically integrate learning into the search process, reducing unnecessary state exploration and improving scalability for complex problems.

The algorithm of SEAL-C is provided in Alg. 2. This design removes the state validity checking to prevent it from compromising completeness. We use the learning-guided complete state decomposition and two-phase ranking to improve the search efficiency.

D Additional Details of SEAL and SEAL-C

D.1 Prompt Templates of Direction Solution Generation

We present the examples of using p_{solve} to directly generate solution for Game of 24 and Mini Crosswords in Table 8 and Table 9.

D.2 Prompt Templates of State Validity Checking

We present an example of using p_c to check state validity for Game of 24 in Table 10.

D.3 Prompt Templates of Learning-guided Ranking

We present an example of using p_v to generate LLM-estimated score for the states in Game of 24 in Table 11.

E Additional Details of Self-Search

To systematically explore the impact of search on LLMs and whether LLMs can develop search capabilities on their own, we consider two types of self-search prompts:

- High-level self-search: In this approach, we prompt the LLMs to solve problems through search without specifying which search method to use. This means the LLMs must rely solely on their intrinsic knowledge and reasoning capabilities. We compare this method with VANILLA

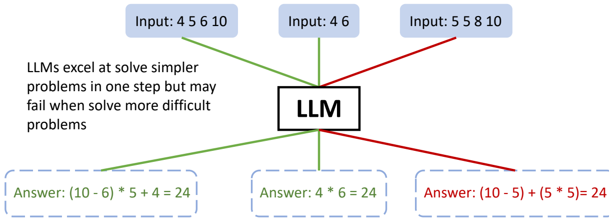


Figure 5: Illustrative examples of Observation 1 in Sec. 3.2. This figure presents three examples of calculating 24 from intermediate steps in the Game of 24 task. The green answer represents a correct equation that results in 24, whereas the red answer represents an incorrect equation that does not equal 24. We observe that LLMs perform well in solving simple tasks in one step, such as $4 \times 6 = 24$ and $(10 - 6) \times 5 + 4 = 24$, but struggle with more complex tasks in a single step (e.g., the third example, where the model fails to find a solution using numbers 5, 5, 8, and 10).

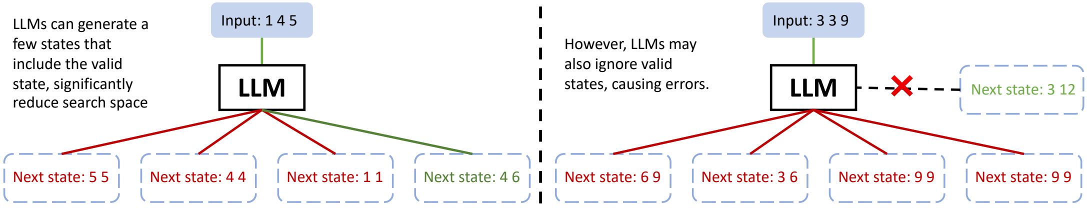


Figure 6: Illustrative examples of Observation 2 in Sec. 3.2. This figure presents two examples of using BEAM SEARCH for state decomposition to generate the next substates in the game of 24 task. Specifically, the method selects two numbers from the input set and applies a basic arithmetic operation $(+, -, \times, \div)$ to produce a new set of numbers. The green next state represents an intermediate state that can lead to 24, whereas the red next state denotes an invalid intermediate state that cannot lead to 24. The left example demonstrates that LLMs can generate a few valid next steps (e.g., selecting $4, 6$ in this case) rather than exhaustively decomposing all possible states through brute-force search. The right example illustrates that LLMs may sometimes overlook valid states and produce only invalid decomposed states.

CoT to assess the importance of search in LLM problem-solving. We show this prompt in Table 12.

- Low-level self-search: In this approach, we explicitly encode the search strategy of SEAL into the prompt. This method allows us to explore whether LLMs can perform self-search and further demonstrate the effectiveness of SEAL in improving LLM reasoning for problem-solving. We show this prompt in Table 13.

F Code

We will release our code at: `https://anonymous.4open.science/r/SeaL-B5B2`.

G Experimental Settings

G.1 Task setup

In this subsection, we introduce the three tasks used in our experiments. To demonstrate the feasibility of SEAL in various realistic applications, we conduct experiments on Game of 24 (Yao et al., 2023a), Mini crosswords (Yao et al., 2023a) and Blocksworld (Valmeekam et al., 2022).

Game of 24 (Yao et al., 2023a). This task is a mathematical reasoning challenge where the objective is to use four numbers and basic arithmetic operations (+-*/) to obtain the value 24. For example, given the numbers "1 3 3 7", a valid solution is "$1 \times 3 \times 7 + 3 = 24$". The dataset used in this task is publicly available [3]. In our experiments, we follow the setup described in (Yao et al., 2023a), selecting 100 groups of four numbers indexed from 900 to 999 as the target problems. To evaluate the effectiveness of the search methods, we compute the pass rate (PR) over these 100 problems. Additional details regarding the evaluation metrics are provided in Appendix G.4.
Mini Crosswords (Yao et al., 2023a). This task, introduced by(Yao et al., 2023a), involves solving a 5×5 mini crossword puzzle. Specifically, given five horizontal clues and five vertical clues, the aim is to fill a 25-letter grid. We adopt the experimental setup from (Cao et al., 2024), wherein each clue is accompanied by a fixed-length list of candidate words. The agent selects words from the list based on the textual descriptions of the clues. The list lengths range from $\{6, 7, 8, 9, 10, 11\}$ words. To

[3] `https://www.4nums.com/game/difficulties/`

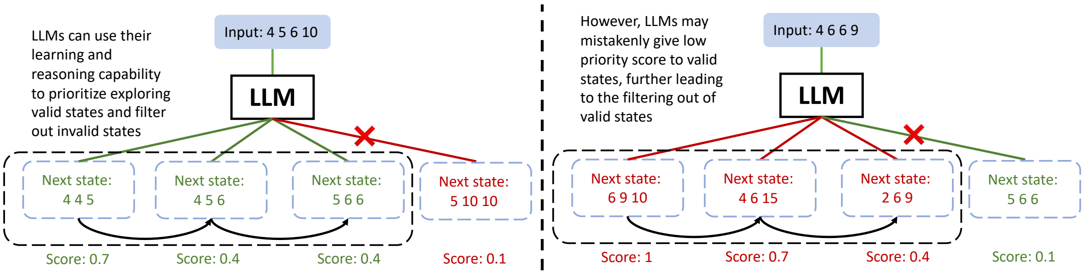


Figure 7: Illustrative examples of Observation 3 in Sec. 3.2. This figure presents two examples of using an LLM as a verifier to assess intermediate states' progress toward the goal of 24 in the Game of 24 task. The green next state represents an intermediate state that can lead to 24, whereas the red next state denotes an invalid intermediate state that cannot lead to 24. In the left example, given the input numbers $4, 5, 6, 10$, the LLM effectively assigns high scores to valid next states (e.g., $4, 4, 5$ with a score of 0.7, where $4 \times 5 + 4 = 24$) and low scores to invalid next states (e.g., $5, 10, 10$ with a score of 0.1, which cannot achieve 24). In contrast, the right example, given the input numbers $4, 6, 6, 9$, demonstrates that LLMs may mistakenly assign high priority scores to invalid states. For instance, $6, 9, 10$ receives the highest score of 1 but cannot achieve 24. This highlights the need for further improvements to ensure that critical states are not overlooked.

generate these word lists, we utilize GPT-4o (Hurst et al., 2024) to produce candidate words that either share similar meanings with or have more than two matching letters in common with the ground-truth words. Details regarding the prompts used to generate these candidate words are provided in Table 14.

Blocksworld (Valmeekam et al., 2022). In this task, the agent must rearrange blocks into specific stacked configurations. A **state** represents the current arrangement of the blocks, while an **action** is a command that manipulates them. Each action consists of one of four verbs—STACK, UNSTACK, PUT, or PICKUP—along with the corresponding objects. The **action space** is defined by the set of valid actions that comply with the domain constraints and depend on the current state of the blocks. The agent transitions between states by selecting an action and querying the LLM to predict the resulting changes in the block configuration. The state is then updated by incorporating new conditions for the blocks and removing any conditions that are no longer valid. To evaluate plan correctness, we convert both the generated plans and their corresponding textual problem descriptions into the Planning Domain Definition Language (PDDL) (Silver et al., 2024) format. We then use VAL (the Automatic Validation Tool for PDDL) (Howey and Long, 2003) to assess their validity.

Note that these tasks mainly belong to the offline planning scenario. However, we posit that our method is also applicable to online planning scenarios, where plans are dynamically updated during interactions. We aim to explore the application of our approach to such tasks in future work.

G.2 Baselines

Traditional search methods.

- BRUTE-FORCE (DFS): Depth-First Search (DFS) explores each possible solution path by diving deeply into one branch before backtracking to explore others.
- BRUTE-FORCE (BFS): Breadth-First Search (BFS) systematically explores all intermediate steps at a given depth before progressing to the next level, ensuring all possibilities are considered at each step.
- BRUTE-FORCE (DFS-PRUNE): This is a variant of DFS. Specifically, when encountering previously visited states, this method skips their exploration. As a result, it improves the search efficiency of BRUTE-FORCE (DFS) by reducing the search space.
- BRUTE-FORCE (BFS-PRUNE): This is a variant of BFS. Similar to BRUTE-FORCE (DFS-PRUNE), it also skips the exploration of previously visited states, thereby reducing the search space.

LLM-based search methods.

- BEST-OF-N (Snell et al., 2024): In this approach, we independently sample N answers from the base LLM and select the best answer according to the Process Reward Model (PRM)'s final answer judgment. Especially, we follow the setting

Algorithm 1 Algorithm of SEAL.

Input: Initial state s^{init}, goal state s^{goal}, Language Model LLM, external verifier f.
Output: Action sequence $\mathcal{A}$ transforming s^{init} to s^{goal}

function F(s^{cur}, M, f, $\mathcal{A}^{cur}$)
  $p^{cur} \leftarrow P_1(s^{cur})$ // State validity check
  if p^{cur} **then**
    return r^{cur}
  end if
  $r^{cur} \leftarrow M(cur)$ // Direct Solution Generation
  if $f(r^{cur})$ **then**
    return r^{cur} // $r^{cur} \in s^{goal}$ if r^{cur} is correctly verified by f
  end if
  $S^{next} \leftarrow \{s_1^{next}, s_2^{next}, ...\}$, $\mathcal{A}^{nc} \leftarrow \{a_1^{next}, a_2^{next}, ...\}$ // Decompose s^{cur} into substates S^{next} with correcsponding actions $\mathcal{A}^{nc}$
  while $S^{next} \neq \emptyset$ **do**
    $S'^{next} \leftarrow \text{Rank}(S^{next})$
    $s_i^{next} \leftarrow S'^{next}[0]$ // Select most possible state s_i^{next} from S'^{next}
    $\mathcal{A}^{next} = \mathcal{A}^{cur} + [a_i^{next}]$
    $r^{next} \leftarrow \text{F}(s_i^{next}, M, f, \mathcal{A}^{next})$
    if $r^{next} \neq None$ **then**
      return r^{next}
    end if
    Remove s_i^{next} from S'^{next}
  end while
  return None
end function
$\mathcal{A} \leftarrow []$
$\mathcal{A} \leftarrow F(s^{init}, M, F, \mathcal{A})$

Algorithm 2 Algorithm of SEAL-C.

Input: Initial state s^{init}, goal state s^{goal}, Language Model LLM, external verifier f.
Output: Action sequence $\mathcal{A}$ transforming s^{init} to s^{goal}

function F(s^{cur}, M, f, $\mathcal{A}^{cur}$)
  $r^{cur} \leftarrow M(cur)$ // Direct Solution Generation
  if $f(r^{cur})$ **then**
    return r^{cur} // $r^{cur} \in s^{goal}$ if r^{cur} is correctly verified by f
  end if
  $S^{next} \leftarrow \{s_1^{next}, s_2^{next}, ...\}$, $\mathcal{A}^{nc} \leftarrow \{a_1^{next}, a_2^{next}, ...\}$ // Learning-guided Complete State Decomposition
  while $S^{next} \neq \emptyset$ **do**
    $S'^{next} \leftarrow \text{Rank}(S^{next})$ // Two phase state ranking
    $s_i^{next} \leftarrow S'^{next}[0]$ // Select most possible state s_i^{next} from S'^{next}
    $\mathcal{A}^{next} = \mathcal{A}^{cur} + [a_i^{next}]$
    $r^{next} \leftarrow \text{F}(s_i^{next}, M, f, \mathcal{A}^{next})$
    if $r^{next} \neq None$ **then**
      return r^{next}
    end if
    Remove s_i^{next} from S'^{next}
  end while
  return None
end function
$\mathcal{A} \leftarrow []$
$\mathcal{A} \leftarrow F(s^{init}, M, F, \mathcal{A})$

in (Yao et al., 2023a; Hao et al., 2023; Lightman et al., 2023) to use an LLM as the PRM.

- MAJORITY VOTE (Wang et al., 2023): This method is inspired by the intuition that solutions for complex problems are rarely unique. We generate $N = 10$ outputs using the LLM and determine the final result by selecting the majority-voted answer among them. This leverages the consensus across multiple outputs to improve accuracy and robustness.

- BEAM SEARCH (Yao et al., 2023a): Beam search optimizes the PRM by focusing on its per-step predictions to identify the most promising solution paths. Following (Snell et al., 2024), we implement a beam search variant based on the BFS-based Tree of Thought (ToT) framework from (Yao et al., 2023a). Specifically, we maintain a beam width $M = 5$ for the three tasks.

- BEAM SEARCH+RV: While standard LLM-based search methods rely solely on the LLM to obtain the final answers, which is prone to hallucination and scoring misalignment. Inspired by the direct solution generation of SEAL introduced in Sec. 3.4, we implement BEAM SEARCH+RV, which integrates a rule-based verifier into the final decision stage. The framework mirrors standard beam search in its iterative path expansion and scoring but diverges in the terminal step: instead of selecting the answer with the highest PRM score, the rule-based verifier applies deterministic constraints (e.g., task-specific logical checks) to identify admissible solutions. BEAM

SEARCH+RV fails to solve the problem if no candidate satisfies the verification rules.

G.3 Language Models

To investigate the feasibility of embodying LLMs into search, we consider several close-sourced and open-sourced LLMs offering state-of-the-art reasoning capabilities:

- **GPT-4o** (Hurst et al., 2024): This model is a widely recognized LLM with strong general-purpose reasoning capabilities, making it a standard LLM for evaluating search methods.
- **GPT-4o-mini** (Achiam et al., 2023): This is a small version of GPT-4o, which has 8B parameters according to (Abacha et al., 2024). This model is included to explore the potential of using Small Language Models (SLMs) within SEAL, particularly to demonstrate its applicability in resource-constrained scenarios.
- **Qwen2.5-72B-Instruct** (Qwen, 2024a): This is a representative state-of-the-art open-sourced LLMs developed by Qwen Team. It excels in tasks requiring logical deduction, problem-solving, and multi-step reasoning. The model has been fine-tuned to align with user instructions, making it versatile for various applications, from natural language understanding to complex decision-making tasks.
- **QwQ-32B-Preview** (Qwen, 2024b): This is an experimental, state-of-the-art LRM with exceptional logical reasoning and mathematical problem-solving skills. Our experiments reveal that this model explicitly conducts search processes by trying different choices. However, as mentioned in (Qwen, 2024b), we also notice that this model may enter circular reasoning patterns, leading to lengthy responses without a conclusive answer. Thus, we involve this model in our experiments to highlight the importance of search in LLM reasoning and the effectiveness of SEAL's search strategy.
- **DeepSeek-R1** (Guo et al., 2025): The latest open-source LRM, comparable to OpenAI's o1 (OpenAI, 2024). Similar to QwQ-32B-Preview, this model demonstrates explicit search behavior in generating results, further validating the relevance of search methods in improving LLM reasoning.

G.4 Evaluation Metrics

In this subsection, we give the details of the evaluation metrics used in three tasks, which evaluate the performance of the search methods from the perspectives of effectiveness and efficiency.

Pass Rate (PR). This metric evaluates the solution quality to evaluate the problem-solving capability of the search methods. The following is how to calculate PR in the three tasks:

- Game of 24: PR is calculated as the percentage of generated equations that utilize all the given numbers exactly once and correctly evaluate to 24 across the given set of problems.
- Crosswords: PR is evaluated at three levels: letter-level, word-level, and game-level. **Letter-level PR** measures the proportion of letters in the generated solutions that match exactly with the letters in the ground-truth board. **Word-level PR** evaluates the percentage of words in the generated solutions that correspond exactly to the ground-truth answers. **Game-level PR** measures the proportion of problems that are entirely solved, where a problem is considered completely solved only if all letters in the generated solution match the ground-truth answer precisely.
- Blockswords: PR is calculated by the percentage of problems where the generated plans successfully achieve the goal within 120% of the minimum required steps.

Search Steps (SS). This metric measures the search methods by counting traversed states in the search space $\mathcal{S}$. To calculate this metric for our SEAL and SEAL-C, it is further broken down into the following components:

- LLM Calls, which is composed of:
  - Number of LLM Answerer Calls: The number of steps taken to directly obtain solutions via LLMs.
  - Number of State Validity Checking Calls: This refers to the number of steps taken to verify the validity of states. While LLMs are primarily used for state validity checking, we also employ traditional rule-based methods in certain tasks to ensure completeness.
  - Number of State Ranking Calls: The number of steps taken to rank states.

- External Calls, which is composed of:
  - Number of Decomposition Calls: The number of steps used for state decomposition.
  - Number of External Verifier Calls: The number of steps used to call the external verifier for validating solutions.

For BEAM SEARCH and BEST-OF-N, the metric can be broken down into:

- LLM Calls:
  - Number of LLM Answerer Calls: The number of steps taken to obtain the next-level intermediate steps.
  - Number of LLM Verifier Calls: The number of steps where the LLM is called as an external verifier for intermediate step evaluation.

For MAJORITY VOITE, the metric can be broken down into:

- LLM Calls:
  - Number of LLM Answerer Calls: The number of steps taken to directly obtain solutions.

For the traditional brute-force search methods, DFS and BFS, the metric then can be broken down into:

- External Calls:
  - Number of Decomposition Calls: The number of steps used to perform state decomposition.
  - Number of Trasversed States: The number of explored states, including both intermediate and final steps.
  - Number of External Verifier Calls: The number of steps where the external verifier is called to evaluate the final solutions.

G.5 Implementation Details

Following the settings in prior studies (Yao et al., 2023a; Hao et al., 2023; Snell et al., 2024), we set the temperature for LLMs to 0.7. To use GPT-4o (Hurst et al., 2024) and GPT-4o-mini (Achiam et al., 2023), we set the mode as the Chat Completion modes for them. For DeepSeek-R1 (Guo et al., 2025), we use the Chat mode to run experiments.

When using BEAM SEARCH, we follow the approach from (Yao et al., 2023a) and prompt the LLM three times for each intermediate step, asking whether it was "sure," "maybe," or "impossible" to reach 24. We then averaged these responses to evaluate the intermediate steps.

For the baseline search BEST-OF-N, we use the similar setup to BEAM SEARCH. We evaluate the N solutions at the three levels mentioned above and determine the final result by averaging the responses.

For our SEAL and SEAL-C, when checking the validity of states, such as in the Game of 24, we prompt the LLM five times to get five binary evaluations ("True/False"). The final decision is based on a majority vote to determine whether we should continue exploring or not.

H Full Analysis of RQ1: Effectiveness and Efficiency Evaluations

To answer **RQ1**, we conduct experiments on the three tasks using three different LLM backbones: GPT-4o-mini, GPT-4o, and Qwen2.5-72B-Instruct. For Game of 24, we follow (Yao et al., 2023a) to select 100 problems indexed from 900 to 999. For Mini Crosswords, we randomly choose 20 problems with 11 candidate words for each clue, while for Blocksworld, we randomly select 20 problems with a minimum solution length of 8 steps. The results are reported in Table 2. From the table, we observe: **(i)** SEAL achieves a near-perfect pass rate across all settings, outperforming other LLM-based methods, which achieve pass rates of up to 54.0%, 45.0%, and 47.4% for the three tasks, respectively. This validates the effectiveness of SEAL in problem-solving. **(ii)** SEAL significantly reduces search steps compared to brute-force searches. Specifically, it reduces search steps by up to 99.1% compared to DFS and still achieves state-of-the-art search steps compared to other LLM-based methods. This validates SEAL's efficiency in navigating the search space. **(iii)** BEAM SEARCH+RV achieves better pass rates than BEAM SEARCH, indicating that integrating rule-based verifiers into LLM-based methods enhances their reliability in problem-solving. However, it still falls behind SEAL, confirming that current LLM-based methods do not fully conduct real search. This further highlights the superior search effectiveness of SEAL. Full search results and the discussion of SEAL with various verifiers are in Appendix I.

I Additional Experimental Results of SEAL

I.1 Full Comparison Results on Three Tasks using Various LLMs

We present the complete results of the compared search methods using various LLM models for the Game of 24, Mini Crosswords, and Blocksworld tasks in Table 4, Table 5, and Table 6, respectively. The results align with the observations discussed in Sec. 6.2.

I.2 Impact of Verifier in Search with SEAL

In Sec. 3.4, we introduce the direct solution generation component of SEAL, which leverages LLMs to generate solutions and uses a verifier f to evaluate their correctness. The verifier can take various forms, such as a traditional rule-based verifier, a PRM (Lightman et al., 2023), or an ORM (Uesato et al., 2022). For our implementation, we opt for a rule-based verifier such that we can focus on explorating search. However, there are scenarios where a verifier f may not be available. To evaluate the effectiveness of SEAL under such conditions, we implement a variant of SEAL that excludes both the verifier and the direct solution generation component. The results on the *Game of 24* task using GPT-4o-mini are shown in Table 7. From the table, we observe that both variants of SEAL —with and without a verifier—achieve high accuracy. Notably, the variant without a verifier (SEAL +NV) achieves a 90% pass rate while requiring only 44.9 search steps. This demonstrates the effectiveness and flexibility of SEAL across various settings.

J Additional Experimental Results of SEAL-C

In this section, we will provide the additional results about the impact of problem difficulty on the completeness of search on Mini Crosswords task using GPT-4o-mini, which is shown in Fig 8. Specifically, Fig. 8 (a) - (c) present the results of letter-level, word-level and game-level pass rates, respectively. Fig. 8 (d) is the result of search step. The results align with the observations discussed in Sec. 6.3.

K Additional Details of Problem-solving Processes of LRMs

In this section, we present a detailed analysis of how existing Large Retrieval Models (LRMs) utilize search strategies to solve problems, underscoring the critical role of search in effective problem-solving.

Table 15 illustrates an example of QwQ-32B-Preview's problem-solving process in the Game of 24 task. From this table, we observe that QwQ-32B-Preview explicitly explores multiple potential answers through an iterative search process, continuing until a correct solution is identified and verified. This mechanism of LRMs likely explains why LRMs exhibit superior problem-solving capabilities compared to general LLMs such as GPT-4o, further emphasizing the importance of search in leveraging LLMs for complex problem-solving.

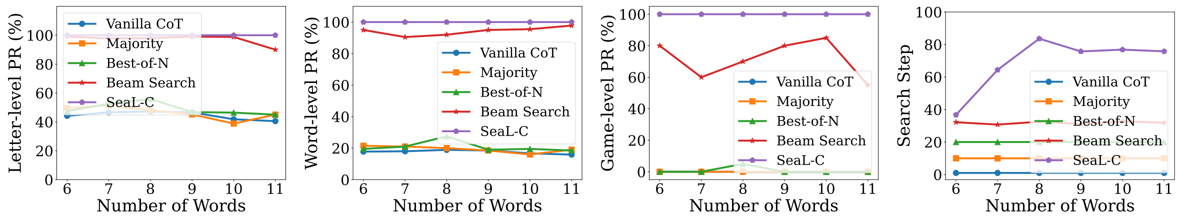
However, we also note a limitation in QwQ-32B-Preview's approach: its search strategy sequentially evaluates and verifies candidate solutions one by one, which may be suboptimal in terms of efficiency. In contrast, Tables 16,17, and18 illustrate the problem-solving process when employing our SEAL's search strategy. From the table, we observe that QwQ-32B-Preview can understand and execute our SEAL's search strategy, allowing it to arrive at solutions more efficiently and effectively—without relying on exhaustive trial-and-error. This observation suggests that LRMs can significantly benefit from more advanced search methodologies, reinforcing the need to explore ways to enhance the self-search capabilities of LLMs.

L Additional Experimental Results of Ablation Studies

In this section, we present additional ablation studies to analyze the impact of each key component of SEAL, as introduced in Sec. 3.4, and to further explore their contributions toward ensuring effective and efficient search. Specifically, we evaluate several ablated variants: (i) SEAL/V: this variant removes the state validity checking and directly proceeds to rank the substates after decomposition; (ii) SEAL/D: this variant excludes the direct solution generation component, relying solely on the generation of next-level intermediate steps at each stage; and (iii) SEAL/R: this variant disregards the learning-guided ranking component, exploring substates sequentially based on their default order. We compare SEAL against these variants on the Game of 24 using GPT-4o-mini, and the results are summarized in Fig. 9. From the figure, we observe the following:

Table 4: Results of different search methods in the Game of 24 task.

Model	Method	PR (%)	Avg. SS	Avg. LLM Calls	Avg. Ext. Calls
	Exhaustive Search	100	12,928	0	12,928
	Brute-force (DFS)	100	1,623	0	1,623
	Brute-force (BFS)	100	3,429	0	3,429
GPT-4o-mini	Vanilla CoT	13	**1**	1	0
	Majority Vote	15	10	10	0
	Best-of-N	17	20	20	0
	Beam Search	46	77	77	0
	Beam Search+RV	72	81	77	4
	SeaL	**100**	40.5	33.4	7.1
GPT-4o	Vanilla CoT	13	**1**	1	0
	Majority Vote	23	10	10	0
	Best-of-N	24	20	20	0
	Beam Search	81	83.4	83.4	0
	Beam Search+RV	95	87	83.4	3.6
	SeaL	**100**	66.8	55.9	10.9
Qwen2.5-72B	Vanilla CoT	17	**1**	1	0
	Majority Vote	23	10	10	0
	Best-of-N	27	20	20	0
	Beam Search	35	124	124	0
	Beam Search+RV	70	128.2	124	4.2
	SeaL	**100**	84.9	68.5	16.4



(a) Letter-level PR (b) Word-level PR (c) Game-level PR (d) SS

Figure 8: Impact of problem difficulty on the completeness of search on Game of 24 and Blocksworld tasks using GPT-4o-mini.

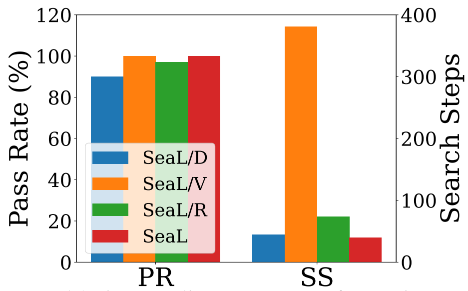


Figure 9: Ablation studies on Game of 24 using GPT-4o-mini to explore the contribution of each key component in SeaL.

- While SeaL achieves comparable pass rates to SeaL/V, it requires significantly fewer search steps. This improvement is attributed to the state validity checking component, which leverages the reasoning capabilities of LLMs to evaluate state validity and determine whether further exploration is warranted. By filtering out invalid states, the search budget is conserved, preventing unnecessary exploration. These findings corroborate our earlier analysis in Sec. 3.2.

- Both SeaL/D and SeaL achieve similar pass rates; however, SeaL/D requires more search steps. This discrepancy arises because the direct solution generation component in SeaL can resolve simpler problems within intermediate substates in a single step, eliminating the need for further exploration. This observation aligns with our analysis in Sec. 3.2, which highlights the capability of LLMs to solve simpler problems more efficiently.

- The search steps for SeaL/R are higher than for SeaL, confirming the effectiveness of the learning-guided ranking component in prioritizing valid states that lead to the goal state earlier. Notably, SeaL/R achieves a slightly lower yet comparable pass rate to SeaL. This reduction can be attributed to occasional misclassification of valid states as invalid due to the limitations in the reasoning capabilities of LLMs. This result underscores the necessity of incorporating the full SeaL pipeline in scenarios where a 100%

Table 5: Results of different search methods in Mini Crosswords (11-words) task.

Model	Method	Letter-level PR	Word-level PR	Game-level PR	Avg. SS	Avg. LLM Call	Avg. Ext. Call
	Exhaustive Search	100	100	100	TL	0	TL
	Brute-force (DFS)	100	100	100	4,128.9	0	4,128.9
	Brute-force (BFS)	100	100	100	TL	0	TL
GPT-4o-mini	Vanilla CoT	40.5	15.9	0	1	1	0
	Majority Vote	45.0	19.0	0	10	10.0	0
	Best-of-N	45.0	18.5	0	20.0	20.0	0
	Beam Search	97.8	90.0	55.0	30.9	30.9	0
	Beam Search+RV	99.2	99.0	90.0	30.9	29.1	1.8
	Ours	100	100	100	75.8	10.2	65.6
GPT-4o	Vanilla CoT	54.4	26.0	5.0	1	1	0
	Majority Vote	61.6	31.0	5.0	10	10.0	0
	Best-of-N	65.6	34.0	5.0	20.0	20.0	0
	Beam Search	68.0	61.5	35.0	21.3	21.3	0
	Ours	100	100	100	45.0	19.4	25.6
Qwen2.5-72B	Vanilla CoT	58.3	25.6	0	1	1	0
	Majority Vote	55.6	26.3	5.0	10	10.0	0
	Best-of-N	60.4	28.5	5.0	20	20.0	0
	Beam Search	96.8	88.5	65.0	32.0	32.0	0
	Ours	100	100	100	68.1	27.0	41.1

Table 6: Results of different search methods in the Blocksworld (8-steps) task.

Model	Method	PR (%)	Avg. SS	Avg. LLM Calls	Avg. Ext. Calls
	Exhaustive Search	100	TL	0	TL
	Brute-force (DFS)	100	18,531.9	0	18,531.9
	Brute-force (BFS)	100	96,759.4	0	96,759.4
GPT-4o-mini	Vanilla CoT	0	**1**	1	0
	Majority Vote	35	10	10	0
	Best-of-N	0	20	20	0
	Beam Search	0	66.8	66.8	0
	Beam Search+RV	0	94.8	90.8	4.0
	SeaL	**100**	160.6	50.2	110.4
Qwen2.5-72B	Vanilla CoT	35	**1**	1	0
	Majority Vote	60	10	10	0
	Best-of-N	60	20	20	0
	Beam Search	0	68.7	68.7	0
	SeaL	**100**	68.7	22.9	45.8

pass rate is critical, particularly in high-stakes applications.

Table 7: Result of the impact of verifier in the Game of 24 task using GPT-4o-mini.

Method	PR (%)	Avg. SS	Avg. LLM Calls	Avg. Ext. Calls
VANILLA COT	0	**1**	1	0
MAJORITY VOTE	35	10	10	0
BEST-OF-N	0	20	20	0
BEAM SEARCH	46	77	77	0
BEAM SEARCH+RV	46	77	77	0
SEAL+NV	90	44.9	38.2	6.7
SEAL	**100**	40.5	33.4	7.1

Table 8: The prompt of direct solution generation for solving Game of 24

Use the given numbers and basic arithmetic operations (+, -, , /) to reach 24. Each step should only select two of the remaining numbers to calculate a new number, aiming to reduce the total count of numbers by merging the selected pair into their result. The steps should methodically progress towards constructing an expression that equals 24 when evaluated. Each remaining number can be only selected up to once. For example:

Input: 4 4 6 8
Steps:
step1: 4 + 8 = 12; (left: 4 6 12)
step2: 6 - 4 = 2; (left: 2 12)
step3: 2 * 12 = 24; (left: 24)
Answer: (6 - 4) * (4 + 8) = 24

[MORE FEW-SHOW EXAMPLES]
Ensure that each arithmetic operation is possible and leads to the correct remaining numbers. The final answer should correctly reflect the steps performed to achieve 24. If an error occurs in calculation, revise the final expression accordingly.
Given the following input, the generated output should be formatted exactly as above:
Input: [INPUT]

Table 9: The prompt of direct solution generation for solving Mini Crosswords

Solve 5x5 mini crosswords by selecting appropriate words from provided candidate lists. Given an input of 5 horizontal clues and 5 vertical clues, a list of words is given for each clue. Consider intersecting constraints with other words, solve this step by step, generate thoughts about 5-letter word from the corresponding list fits each clue, and then select the most suitable word for each clue. Then an output of 5 rows, where each row is 5 letter separated by space. # Few-shot example 1 Input: h1. A lunar valley h2. A fatty oil h3. To entice h4. To lower; to reduce h5. A solitary person v1. According to the roster v2. Another name for Port-Francqui v3. An illicit lover; a European lake v4. To lisp v5. To come in Thoughts: h1. A lunar valley: RILLE h2. A fatty oil: OLEIN h3. To entice: TEMPT h4. To lower; to reduce: ABASE h5. A solitary person: LONER v1. According to the roster: ROTAL v2. Another name for Port-Francqui: ILEBO v3. An illicit lover; a European lake: LEMAN v4. To lisp: LIPSE v5. To come in: ENTER Output: R I L L E O L E I N T E M P T A B A S E L O N E R Input: INPUT

Table 10: The prompt of state validity checking for solving Game of 24

Your task is to analyze a mathematical game where the goal is to use basic arithmetic operations (+, -, *, /) to achieve the target number 24. You are given several sets of numbers. For each set, determine if it is possible (likely/unlikely) to achieve 24 using any combination of numbers and operations. For each set, each number must be used exactly once. If one of these sets are likely to obtain 24, return the answer Yes and these sets that are likely to reach to 24. Examples: Example 1: Input: State 0: 8 3 1 State 1: 6 4 State 2: 7 7 2 State Precheck: Answer: yes; Reason: State 1 is likely to reach 24 Example 2: Input: State 0: 8 State 1: 11 2.66 State 2: 7 7 State Precheck: Answer: no; Example 3: Input: State 0: 24 State 1: 12 State Precheck: Answer: yes; Reason: State 0 and State 2 can directly reach to 24 because the single number 24 is directly equal to 24, 24 = 24. Given the following input sets, generate the output in the exact format above: Input: [INPUT]

Table 11: The prompt of learning-guided state ranking for solving Game of 24

Evaluate if given numbers can reach 24, each given number must be used exactly once (sure/likely/impossible) Current numbers: 10 14; Calculation: 10 + 14 = 24; Comment: I can obtain the 24 by using current numbers; Conclusion: sure Current numbers: 11 12; Calculation: 11 + 12 = 23; Calculation: 12 - 11 = 1; Calculation: 11 * 12 = 132; Calculation: 11 / 12 = 0.91; Comment: I cannot obtain the 24 by using current numbers; Conclusion: impossible Current numbers: 5 7 8; Calculation: 5 + 7 + 8 = 12 + 8 = 20; Calculation: (8 - 5) * 7 = 3 * 7 = 21; Comment: I cannot obtain 24 now, but numbers are within a reasonable range; Conclusion: likely [MORE FEW-SHOT EXAMPLES] Given the following current number, the generated output should be formatted exactly as above: Current number: input;

Table 12: The prompt of high-level self-search for solving Game of 24

You are a problem solver for the Game of 24. Given four numbers, your goal is to find a mathematical expression that equals 24 using each number exactly once, with the allowed operations being addition (+), subtraction (-), multiplication (*), and division (/). Follow these steps precisely:

Game of 24 Rules
Rule 1. Each number must be used exactly once.
Rule 2. Use only basic operation (+-*/) for calculation.

Input
You will receive four numbers. For example: "Input: 1 2 4 7"

Instruction:
You are required to get 24 with the 4 input numbers by using the idea of search for your thoughts in the intermediate steps.

Wrap your your final solution in a special tag <solution> like <solution> 2 + 3 * 5 + 7 = 24 </solution>. Stop when find a correct solution.
Input: [INPUT]

Table 13: The prompt of low-level self-search for solving Game of 24

You are a problem solver for the Game of 24. Given four numbers, your goal is to find a mathematical expression that equals 24 using each number exactly once, with the allowed operations being addition (+), subtraction (-), multiplication (*), and division (/). Follow these steps precisely:
Game of 24 Rules
Rule 1. Each number must be used exactly once.
Rule 2. Use only basic operation (+-*/) for calculation.
Rule 3. Not to generate code to solve the game. And not to return empty content.

Input
You will receive four numbers. For example: "Input: 1 2 4 7"

Instruction
You are required to get 24 with the 4 input numbers by using the idea of search to formalize your thoughts in the intermediate steps. Specifically, the idea search is based on the following stages:
Stage 1. State Precheck. Precheck if it is possible to obtain 24 from the numbers in the current state, where each number is used exactly once. Stop expansion in the current state if precheck fails.
Stage 2. Direct Solution Attempt. Attempt to directly find a solution using the current numbers by thinking step by step.
Stage 3. Check the the correctness of the generated direct solution. If correct, stop early and use it as the final solution. If not, jump to the next stage. Note that all numbers in the solution must from current state and be used exactly once.
Stage 4. Problem decompositon. Decompose current problem into several sub-problems by using basic operation (+-*/) to see if you can solve these subproblems to get 24. For instance. "1 2 4 7" can be decomposed into "2 4 8" by "1 + 7" and "1 2 7" by "4 / 2", where thinking about "2 4 8" is easier than " 1 2 4 7".
Stage 5. State ranks. After getting several decomposed sub-problems, you will evaluate each substate's potential to reach 24. The metric is sure = 1.0, likely = 0.5, impossible = 0.1. Then give a sorted substates list, highest potential first.

Repeat the above stages until a valid action is found. Note that you can conduct backtrace to switch to another subproblem if current subproblem cannot achieve 24. For example, if you find that "1 2 7" cannot achieve 24 even by further decomposition, like "2 7", "1 7" from "1 2 7" or even "14" from "2 7", you can backtrace to another high-layer subproblems like "2 4 8" until you finally find a solution.

Wrap your your final solution in a special tag <solution> like <solution> 2 + 3 * 5 + 7 = 24 </solution>. Stop when find a correct solution.
Input: [INPUT]

Table 14: The prompt of generating candidate words for clues in the Mini Crosswords task

Generate different types of 5-letter words based on a clue and meaning:

Clue: "[TEXTUAL CLUE]"
Ground truth word: [GROUND-TRUTH WORD]

Generate EXACTLY:
1. [NUMBER OF WORDS WITH SIMILAR MEAININGS] five-letter words that are semantically related to the clue's meaning (different from ground truth)
2. [NUMBER OF WORDS WITH SIMILAR LETTERS] five-letter words that share at least 2 letters in the same positions as the ground truth word "[GROUND-TRUTH WORD]"

Requirements:
- Each word must be exactly 5 letters
- Do not include the ground truth word
- Related words must have meanings close to the clue
- Letter-matching words must share at least 2 letters in the same positions as the ground truth word
- Generate exactly the requested number of words

Format your response strictly as:
Related words ([NUMBER OF WORDS WITH SIMILAR MEAININGS] words):
- word1: brief explanation of relevance
- word2: brief explanation of relevance
[continue until exactly [NUMBER OF WORDS WITH SIMILAR MEAININGS] words]

Unrelated words ([NUMBER OF WORDS WITH SIMILAR LETTERS] words):
- word1
- word2
<continue until exactly [NUMBER OF WORDS WITH SIMILAR LETTERS] words>

Table 15: An example of the problem-solving process of using QwQ-32B-Preview for the Game of 24 task

Input:
4 5 6 10

Problem-solving process:
To solve the game of 24 with the numbers 4, 5, 6, and 10, we need to form an expression using each number exactly once and the operations of addition (+), subtraction (-), multiplication (*), and division (/) to reach the result of 24.

Let's explore different combinations step by step.

First, let's consider multiplication and addition since they are likely to help us reach higher numbers.

Option 1:
Multiply 4 and 6 to get 24, but we still need to incorporate 5 and 10. However, since we have to use all numbers exactly once, this direct multiplication alone isn't sufficient.

Option 2:
Consider combining addition and multiplication.

Let's try:

(5 - 4) * 10 + 6 = 1 * 10 + 6 = 10 + 6 = 16
Not 24.
[OMITTING 10 ADDITIONAL OPTIONS....]

Option 13:
Another approach:

(5 + 10) / 5 * 4 = 15 / 5 * 4 = 3 * 4 = 12

Not 24.
Option 14:
Let's try:

(10 - 4) * 5 - 6 = 6 * 5 - 6 = 30 - 6 = 24

Yes!

So, one possible solution is:

(10 - 4) * 5 - 6 = 24

Let's verify:

First, subtract 4 from 10: 10 - 4 = 6

Then, multiply by 5: 6 * 5 = 30

Finally, subtract 6: 30 - 6 = 24
Perfect!

Table 16: An example of the problem-solving process of using SELF-SEARCH (LOW) QwQ-32B-Preview for the Game of 24 task.

Input:
3 3 7 12

Problem-solving process:
I'm going to try to solve this Game of 24 puzzle with the numbers 3, 3, 7, and 12. The goal is to use each number exactly once with the operations of addition, subtraction, multiplication, and division to reach 24.

Stage 1: State Precheck

First, I need to check if it's possible to get 24 with these numbers. All numbers are positive integers, and 24 is also a positive integer, so it's plausible. I should consider the range of possible values. The smallest value I can get might be around the smallest number, which is 3, and the largest could be much higher, especially with multiplication. So, it seems possible.

Stage 2: Direct Solution Attempt

Let me try to find a direct solution by trying out some combinations:

- 3 + 3 + 7 + 12 = 25 (too high)

- 12 * 2 = 24, but I don't have a 2 directly. Maybe (7 - 3) = 4, but 12 * 4 = 48, which is too high.

- 7 * 3 = 21, then 21 + 3 = 24, but I still have 12 left to use, which is a problem because I have to use each number exactly once.

- Wait, 21 + 3 is 24, but I have an extra 12. Maybe I can involve division or subtraction.

- What if I do 21 + (12 / 3) = 21 + 4 = 25, which is too high.

- Alternatively, 21 + 3 - 12 = 12, which is too low.

- Maybe another approach: 12 * 7 = 84, then 84 / 3 = 28, which is too high.

- 84 / 3.5, but I don't have 3.5; I have two 3's and a 7 and 12.

- Perhaps I can do 12 * (7 - 3) = 12 * 4 = 48, which is too high.

- Then 48 / 2, but again, I don't have a 2.

- Wait, I have two 3's, so maybe 3 / 3 = 1, but 48 / 1 = 48, still too high.

- Maybe another path: 7 * 12 = 84, then 84 / 3 = 28, which is still too high.

- 28 - 4 = 24, but I don't have a 4.

- Hmm.

Stage 3: Check Direct Solution
From the above attempts, I haven't found a direct solution yet. So, I need to proceed to decompose the problem.

Table 17: An example of the problem-solving process using SELF-SEARCH (LOW) with QwQ-32B-Preview for the Game of 24 task (continued from Tab. 16).

Input:
3 3 7 12

Problem-solving process:
Stage 4: Problem Decomposition

Let me try to decompose the problem into sub-problems. I'll pick two numbers and combine them using one of the operations, then see if I can reach 24 with the resulting number and the remaining two numbers.

Option 1: Combine 3 and 3.

- 3 + 3 = 6

- Now, the remaining numbers are 6, 7, 12.

- Can I get 24 from 6, 7, 12?

- 6 * 7 = 42, then 42 - 12 = 30, too high.

- 6 * 12 = 72, then 72 / 7 = 10.285, not 24.

- 7 * 12 = 84, then 84 / 6 = 14, not 24.

- 12 - 6 = 6, then 6 * 7 = 42, which is too high.

- 12 + 7 = 19, then 19 * 6 = 114, too high.

- Doesn't seem promising.

Option 2: Combine 3 and 7.

- 3 + 7 = 10

- Remaining numbers: 10, 3, 12.

- Can I get 24 from 10, 3, 12?

- 10 + 12 = 22, then 22 + 3 = 25, too high.

- 10 * 12 = 120, then 120 / 3 = 40, too high.

- 12 - 10 = 2, then 2 * 3 = 6, too low.

- 12 / 10 = 1.2, then 1.2 * 3 = 3.6, too low.

- Doesn't seem to work.

Option 3: Combine 3 and 12.

- 3 + 12 = 15

- Remaining numbers: 15, 3, 7.

- Can I get 24 from 15, 3, 7?

- 15 + 7 = 22, then 22 + 3 = 25, too high.

- 15 * 7 = 105, then 105 / 3 = 35, too high.

Table 18: An example of the problem-solving process using SELF-SEARCH (LOW) with QwQ-32B-Preview for the Game of 24 task (continued from Tab. 17).

Input: 3 3 7 12
Problem-solving process: - 15 - 7 = 8, then 8 * 3 = 24. Wait, that's 24! - So, 15 - 7 = 8, then 8 * 3 = 24. - But I need to use all numbers exactly once, so let's see: - Original numbers: 3, 3, 7, 12. - Combined 3 and 12 to get 15. - Then, 15 - 7 = 8. - Then, 8 * 3 = 24. - Wait, but I've used one 3 in the combination and another 3 separately, which is fine since there are two 3's. - So, this seems to work. **Stage 5: State Ranks** I've found a potential solution: (3 + 12 - 7) * 3 = 24. Let me verify: - 3 + 12 = 15 - 15 - 7 = 8 - 8 * 3 = 24 Yes, that's correct. **Final Solution** $$(3 + 12 - 7) \times 3 = 24$$