Step Back to Leap Forward: Self-Backtracking for Symbolic Reasoning and Planning in Language Models

Xiao-Wen Yang[1, 2], Xuan-Yi Zhu[1, 2], Ding-Chu Zhang[1, 2], Wen-Da Wei[1, 2], Jie-Jing Shao[1], Zhi Zhou[1], Lan-Zhe Guo[1, 3, †] and Yu-Feng Li[1, 2, †]

[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
[3]School of Intelligence Science and Technology, Nanjing University, China.
{yangxw, zhuxy, weiwd, zhangdc, shaojj, zhouz, guolz, liyf}@lamda.nju.edu.cn

Abstract

Although autoregressive language models demonstrated remarkable performance across various tasks, their effectiveness in symbolic reasoning and decision-making scenarios remains constrained. Recent research indicates that training language models to emulate symbolic search algorithms (e.g. depth-first search or A* algorithm) can yield strong improvements in their symbolic reasoning and planning capabilities. However, existing methods only achieve superficial imitation of symbolic search trajectories, as their generation processes lack explicit backtracking mechanisms. This limitation prevents models from truly mastering symbolic search, often resulting in rigid and redundant outputs with poor solution quality. To address this issue, we propose a self-backtracking mechanism that enables LLMs to autonomously determine when to backtrack through specialized training, effectively utilizing this capability to scale during inference. By introducing a self-improvement strategy, the model can further refine its search process into optimal solution generation, improving problem-solving efficiency. Empirical evaluations demonstrate that our method boosts LLMs' reasoning on the Countdown task by 40% over optimal-path supervised fine-tuning (SFT) and improves both performance and efficiency on the Maze Navigation task.

Introduction

Recent advances in transformer-based autoregressive large language models (LLMs) have demonstrated remarkable capabilities across various domains (Zhao et al. 2023; Achiam et al. 2023). However, their performance on structured symbolic tasks—particularly in reasoning (Valmeekam et al. 2023; Plaat et al. 2024; Yang et al. 2025b) and planning (Shao et al. 2024a,b) remains limited in both accuracy and efficiency. Investigating LLMs' capability for such tasks is crucial for addressing more complex problems (Sullivan and Elsayed 2024). Currently, researchers have proposed methods to enhance LLMs' symbolic reasoning and planning. A prominent approach employs supervised training to enable models to mimic symbolic search algorithms (e.g., depth-first search or A*) (Lehnert et al. 2024; Gandhi et al. 2024; Su et al. 2024; Moon, Park, and Song 2024). Specifically, models learn to predict search trajectories that encode the full execution process of these algorithms. Representative works include Searchformer (Lehnert et al. 2024), which emulates A* search, and Stream of Search (SoS) (Gandhi et al. 2024), which combines hybrid trajectories from multiple algorithms. By guiding LLMs to reflect and explore within context, these methods substantially improve LLMs' symbolic reasoning and planning capabilities.

However, we identify fundamental limitations in these methods: constrained by the inherent autoregressive nature of language models, they achieve only superficial emulation of search trajectories. The autoregressive framework compels models to treat preceding tokens as fixed historical context rather than modifiable state representations, thereby precluding genuine algorithmic backtracking—the core of symbolic search algorithms that enables systematic revision of prior decisions and exploration of alternative paths. This limitation leads to frequent generation of invalid reasoning loops and erroneous state transitions in context, resulting in redundant and rigid outputs that impair precise reasoning and planning capabilities. Moreover, recent reinforcement learning (RL)-based large reasoning models (OpenAI 2024; Qwen 2024), such as DeepSeek-R1 (DeepSeek 2024), fail to acquire explicit exploration and backtracking capabilities. Empirical studies indicate that these RL-based approaches cannot surpass the search and backtracking performance of their base models (Gandhi et al. 2025; Yue et al. 2025).

To address this challenge, we present a novel approach: the **Self-Backtracking** technique, which enables language models to autonomously determine when to perform backtracking through training. Specifically, the model is trained to recognize suboptimal states by generating a designated special token, which indicates that backtracking is required at the current position. During inference, the model leverages this learned mechanism to perform backtracking, enabling reevaluation of prior decisions and explore multiple reasoning trajectories from both width and depth. Finally, through expert iteration based on the explored results, the model achieves self-improvement (Tian et al. 2024), further enhancing the efficiency of generating optimal paths. Figure 1 illustrates the framework of our proposed method.

This method fits well with symbolic search, greatly improving reasoning and planning performances while also making the output more efficient. Experiments are per-

[†]Corresponding author

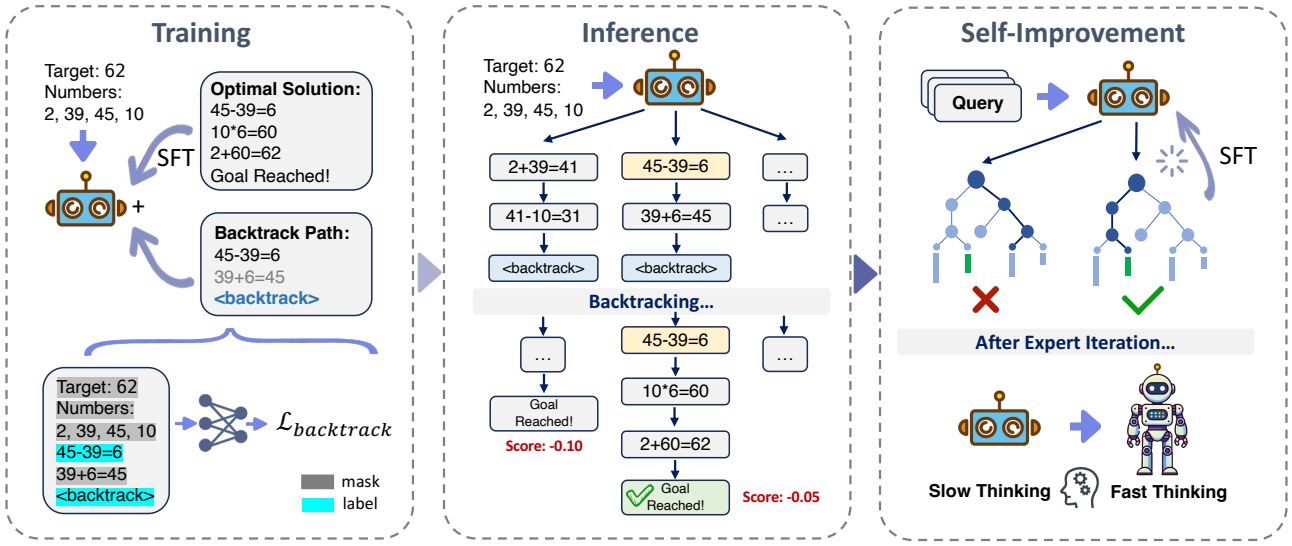

Figure 1: **The overall framework of Self-Backtracking.** During the training phase, the language model is instructed on when to backtrack. The inference phase employs a backtracking algorithm that considers both depth and breadth. The self-improvement phase employs expert iteration to enhance the efficiency of generating optimal solutions.

formed on the Countdown reasoning task (Gandhi et al. 2024) and the Maze Navigation planning task (Lehnert et al. 2024) to assess the advantages of our proposed method over baseline techniques that learn from search trajectories. The results indicate that the self-backtracking technique significantly improves the model's reasoning and planning capabilities while maintaining high output efficiency. Notably, our method demonstrates an accuracy enhancement exceeding 40% compared to the SFT approach that solely relies on the optimal reasoning solutions.

Our contributions are summarized as follows:

- **Problem**: Current symbolic trajectory learning methods are limited by autoregressive generation, lacking explicit backtracking and producing redundant and rigid outputs.
- **Method**: We introduce the self-backtracking technique, which trains LLMs to autonomously learn to backtrack and explore during inference, subsequently leveraging expert iteration to efficiently generate optimal solutions.
- **Evaluation**: Experiments on the Countdown task demonstrate over 40% performance gain over baselines, significantly improving LLM reasoning capabilities. The approach also yields substantial performance and efficiency improvements in the Maze Navigation task.

Related Work

Learn from Search Trajectories. Recently, several studies have explored using symbolic search algorithms to construct trajectory data and train transformer models to learn these search strategies, with the aim of enabling models to perform reasoning tasks. For instance, Yang et al. (Yang et al. 2022) employs Monte Carlo Tree Search (MCTS) or BFS to construct reasoning trajectories. Searchformer (Lehnert et al. 2024) and DualFormer (Su et al. 2024) utilize traces from A* search to train language models, with each trace containing state information, A* heuristic values, and search history. Stream of Search (SoS) (Gandhi et al. 2024) constructs trajectories using various search algorithms to help language models learn the commonalities across different search strategies, facilitating the discovery of new search strategies. GSoS (Moon, Park, and Song 2024) further extends SoS by integrating optimal solutions into the process of learning search trajectories. DeepSeek R1 (Guo et al. 2025) employ reinforcement learning to autonomously acquire the capability of search in language.

Learn from Mistakes. Numerous recent studies have focused on exploring whether language models possess the ability to learn from their previous mistakes and subsequently correct them. One line of techniques (An et al. 2023; Tong et al. 2024; Zhang et al. 2024b; Wang et al. 2024a) introduces the external verifiers to evaluate the reasoning paths generated by LLMs. This evaluation is then used to construct preference training data for RLHF, with training conducted using algorithms such as PPO (Schulman et al. 2017) or DPO (Rafailov et al. 2023), enabling self-improvement (Yuan et al. 2024) of the models. Another line of techniques (Cundy and Ermon 2024; Zhang et al. 2024c; Ye et al. 2024) involves pre-annotating error examples, allowing models to identify whether their current outputs contain issues and adaptively perform backspace during testing to regenerate content. In this paper, our method shows an inherent ability to learn from mistakes and achieves further self-improvement by learning from its explored paths.

Inference Strategies of LLMs. Many strategies for the inference phase have been proposed to enhance problem-solving capabilities of LLMs. Classical methods such as greedy decoding, beam search (Teller 2000; Graves 2012),

and majority voting (Wang et al. 2022; Zhou et al. 2025) have been widely adopted. Additionally, Best-of-N (BoN) (Li et al. 2022) is a typical algorithm that generates N complete answers through sampling and selects the optimal one based on the evaluation of a reward model. Recently, approaches that combine search algorithms with LLMs, such as best-first search (Yao et al. 2024), A* search (Zhuang et al. 2023), guided beam search (Xie et al. 2024b), and MCTS (Choi et al. 2023; Wan et al. 2024; Zhang et al. 2024a; Xie et al. 2024a) have gained increasing attention due to their inference scaling law (Wu et al. 2024; Snell et al. 2024). These methods often require additional components such as a verifier, outcome reward model (Lightman et al. 2024), or process reward model (Lightman et al. 2024; Wang et al. 2024b), which increases computational cost. This paper proposes a novel method that integrates the verifier within the model to save computational resources while leveraging the advantages of search algorithms, demonstrating scalability in symbolic reasoning and planning.

Preliminary

Problem Setup. We adopt the formal definition of symbolic reasoning and planning problem solving as proposed in SoS (Gandhi et al. 2024), where a problem is modeled as a Markov Decision Process (MDP). An MDP is characterized by the following components: a state set $\mathcal{S}$, representing all possible states within the problem domain; an action set $\mathcal{A}$, denoting all permissible actions; a transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, which defines how states transition based on actions; and a reward function $R : \mathcal{S} \to \mathbb{R}$, which assigns a numerical reward to each state. A typical task involves an initial state $s_0 \in \mathcal{S}$ and a goal state $s_g \in \mathcal{S}$, where the goal state s_g is associated with a reward of 1 ($R(s_g) = 1$), while all other states yield a reward of zero ($R(s) = 0, \forall s \neq s_g$). The solution $\mathcal{P}$ to the problem is represented as a trajectory—a sequence of states and actions—$(s_0, a_0, s_1, a_1, \ldots, s_{g-1}, a_{g-1}, s_g)$, where each successive state s_{i+1} is determined by the preceding state s_i and a valid action a_i via $s_{i+1} = T(s_i, a_i)$. The trajectory must terminate at the goal state s_g. Let Σ represent an alphabet, which is a finite, non-empty set of symbols. A string is defined as a finite sequence of symbols drawn from Σ. In this work, we focus on tasks that can be expressed purely in language, where both the state set $\mathcal{S}$ and the action set $\mathcal{A}$ consist of strings. Each state $s \in \mathcal{S}$ represents intermediate state for planning or partial solution for reasoning and each action $a \in \mathcal{A}$ represents a permissible operation that can be performed on the current state to advance the process. The transition function is defined as $T(s, a) = s \circ a$, where $\circ$ denotes string concatenation.

Backtracking Algorithm. Based on the formalization above, we can extend the state-action pairs into a tree, allowing the search process on the tree to be naturally integrated. Backtracking is a classic searching technique that incrementally constructs a solution by exploring various options and retracting decisions when encountering a dead end. This approach is particularly effective in scenarios that require exploring multiple possibilities to solve a reasoning problem, such as navigating a maze or solving puzzles like Sudoku. When the algorithm encounters a dead end, it backtracks to the previous decision point to explore alternative paths, continuing this process until a solution is found or all possibilities are exhausted. Backtracking forms the foundation for many well-known algorithms, including DFS and BFS. This study focuses on enabling LLMs to learn when to backtrack. Traditional search algorithms determine backtracking by identifying terminal states, whereas other approaches, such as A* and MCTS, employ heuristic evaluations to enable early backtracking. In this work, we aim to internalize backtracking capabilities within LLMs without relying on external tools (e.g., symbolic verifiers) or models (e.g., reward models). Furthermore, we hope to enhance existing backtracking frameworks through parallel processing to expand search spaces and improve efficiency.

Self-Backtracking in Language Models

In this section, we introduce our self-backtracking technique. During the training phase, we design a specific optimization goal and a tailored dataset format to help the language model learn when to backtrack. During the inference phase, we use the learned backtracking ability to create an efficient search algorithm, without the need for additional tools or reward models. Finally, through a expert iteration process, the model achieves self-improvement, further enhancing the efficiency of generating optimal paths.

Learn to Backtrack

Under the standard supervised fine-tuning framework, we typically employ a dataset $\mathcal{D}_{op} = \{(x_i, y_i)\}_{i \in [n_{op}]}$, where for reasoning and planning tasks, y_i represents the optimal solution. To enable the model to backtrack at appropriate times and positions, we introduce a backtracking dataset:

$$\mathcal{D}_{backtrack} = \{(x_j, \texttt{prefix}(y_j) \circ a_{err} \circ \langle \texttt{backtrack} \rangle)\}_{j \in [n_b]}$$

Here, $\texttt{prefix}(y_j)$ denotes the prefix of the optimal solution y_j, representing a partial solution; a_{err} signifies an erroneous action extended from the partial solution, which cannot lead to the correct answer; and $\langle \texttt{backtrack} \rangle$ is a special token indicating that the model needs to backtrack from the current state. The final dataset is $\mathcal{D} = \mathcal{D}_{op} \cup \mathcal{D}_{backtrack}$. In our experimental setup, the questions for both $\mathcal{D}_{op}$ and $\mathcal{D}_{backtrack}$ are configured to be identical. This configuration allows us to effectively model the dataset as a preference dataset, so we can compare more baselines.

Through this data construction, if the model learns to recognize a_{err} and utilize the $\langle \texttt{backtrack} \rangle$ token for backtracking, it acquires the ability of when to backtrack. Simultaneously, this dataset format implicitly contains information on where to backtrack, indicating that the model should revert to the state represented by $\texttt{prefix}(y_j)$. Although this design superficially appears to support only single-step backtracking, the recursive nature of the backtracking algorithm allows the model to achieve multi-step backtracking once it masters single-step backtracking.

For the given dataset $\mathcal{D}$, we formulate the training loss function for the language model parameterized by θ as fol-

lows:

$$\mathcal{L}(\theta) = \mathcal{L}_{SFT}(\theta) + \mathcal{L}_{backtrack}(\theta) \tag{1}$$

The loss function comprises two primary components: firstly, the SFT loss:

$$\mathcal{L}_{SFT}(\theta) = -\frac{1}{n_{op}} \sum_{i=1}^{n_{op}} \log p_\theta(y_i|x_i) \tag{2}$$

which aims to encourage the model to generate corresponding steps and final answers based on given questions. The second loss term $\mathcal{L}_{backtrack}$ contains two parts:

$$\mathcal{L}_{backtrack}(\theta) = -\frac{1}{n_b} \sum_{j=1}^{n_b} \log p_\theta(\texttt{prefix}(y_j)|x_j) - \frac{1}{n_b} \sum_{j=1}^{n_b} \log p_\theta(\langle\texttt{backtrack}\rangle|x_j \circ \texttt{prefix}(y_j) \circ a_{err}) \tag{3}$$

The first part targets partially correct paths in backtracking samples, designed to enable the model to accurately predict partial solutions given the input. The second part focuses on the model's ability to predict the $\langle\texttt{backtrack}\rangle$ token when it has deviated from the correct path, encouraging the model to learn when to backtrack. Notably, compared to the SFT loss applied on the $\mathcal{D}_{backtrack}$ dataset, the combination of the second and third loss terms omits the loss component for predicting a_{err}. This design is reasonable as our objective is not to encourage the model to predict incorrect actions but to prevent it from falling into erroneous paths. In practical implementation, this can be achieved by applying a mask to a_{err} when computing SFT loss, as illustrated in Figure 1.

Inference with Backtracking

To incorporate the learned backtracking mechanism into the inference process, we propose a self-backtracking inference algorithm that considers both depth and breadth search. The algorithm operates through three core phases: Expansion, Backtracking, and Selection.

Expansion. In the expansion phase, given the current state s, the algorithm samples N predictions from the language model. These predictions are then categorized into two groups: those containing the $\langle\texttt{backtrack}\rangle$ token and those that do not. Predictions without the $\langle\texttt{backtrack}\rangle$ token are directly added to the candidate set, while those containing the token are processed further in the next phase.

Backtracking. During the backtracking phase, the algorithm selects $\sqrt{N}$ predictions containing the $\langle\texttt{backtrack}\rangle$ token. These states are then rolled back to their previous states, which subsequently undergo the expansion phase again. This process is repeated for b iterations (where b is a hyperparameter representing the maximum allowed backtrack steps) to progressively expand the candidate set.

Selection. Finally, in the selection phase, we evaluate all candidate paths using negative perplexity as the scoring metric (a standard measure adopted by most decoding algorithms) and return the highest-scoring result.

Algorithm 1: Expert Iteration for Self-Improvement

Input: Self-backtracking model M_0, initial Dataset $\mathcal{D}_0$, number of iterations K

for $t \leftarrow 0$ **to** $K-1$ **do**

$\mathcal{D}_t \leftarrow \{M_t(x_i) \mid x_i \in \mathcal{D}_0\}$

$\tilde{D}_t \leftarrow \{(x_i, p_i) \mid (x_i, p_i) \in \mathcal{D}_t$, path p_i is correct $\}$

$M_{t+1} \leftarrow \text{SFT}(M_t, \tilde{D}_t)$

Output: Optimized greedy decoding model M_K

This algorithm enables a flexible search mechanism, where the breadth of exploration is governed by parameter N and the depth by parameter b. It leverages the inherent backtracking capabilities learned by the language model during training without requiring external tools or reward models, while maintaining controllable computational costs throughout the generation process.

Self-Improvement

In this stage we aim to transfer the model's backtracking search abilities (slow thinking) to greedy decoding (fast thinking) through the self-improvement method. To achieve this, we employ an expert iteration strategy, which primarily consists of three steps: First, during the data generation phase, we utilize the self-backtracking inference algorithm to produce high-quality reasoning path data. Subsequently, in the expert screening phase, experts evaluate the generated data to select training samples suitable for the fast thinking model. In our experiment, we quantify the model's accuracy using an evaluator. Finally, in the training phase, the selected high-quality data is used to train the fast thinking model by SFT. Through this iterative optimization, we get continuous enhancement in the performance of the single-pass greedy decoding model. The process is shown in Algorithm 1.

Experiments

In this section, we assess the efficacy of our self-backtracking algorithm in improving LLM's performance on two challenging symbolic tasks: the Countdown problem (Gandhi et al. 2024; Moon, Park, and Song 2024) and the Maze Navigation task (Lehnert et al. 2024; Su et al. 2024). Both tasks require robust symbolic reasoning and planning capabilities, posing substantial difficulties even for human problem-solvers. Experimental results indicate that our method not only enhances task performance but also achieves greater efficiency, demonstrating clear advantages over existing approaches.

Experimental Setup

Countdown. This symbolic reasoning task extends the traditional 24 Game (Yang et al. 2022) by necessitating that LLMs strategically combine a provided set of input numbers using fundamental arithmetic operations—addition, subtraction, multiplication, and division—to achieve a predefined target number. The complexity of the task stems from its expansive search space, which rigorously tests the models' ability in reasoning the correct path. We construct datasets

Method	Llama3.2-1B		Llama3.2-3B	
	Seen Targets	New Targets	Seen Targets	New Targets
▷ *Only Optimal Solution:*				
SFT + Greedy	28.60	28.92	33.98	32.68
SFT + Beam Search	31.68	31.90	35.82	34.36
▷ *RLHF:*				
DPO (Rafailov et al. 2023)	29.06	27.64	34.46	32.72
KTO (Ethayarajh et al. 2024)	28.34	27.74	33.70	32.34
Best-of-N ($N = 8$)	41.26	40.68	47.84	48.56
Best-of-N ($N = 16$)	32.40	33.94	47.28	47.80
Best-of-N ($N = 32$)	25.60	27.04	44.38	45.88
▷ *Backtracking Data:*				
Greedy	28.92	27.06	31.06	31.76
Beam Search	36.10	34.30	39.20	37.10
Self-Backtracking ($b = 0, N = 8$)	66.66	67.40	65.18	63.26
Self-Backtracking ($b = 0, N = 32$)	70.66	72.14	67.02	64.54
Self-Backtracking ($b = 1, N = 8$)	67.60	68.02	66.70	64.64
Self-Backtracking ($b = 1, N = 32$)	**73.54**	**73.78**	**68.76**	**66.64**

Table 1: **Self-Backtracking Enhances Reasoning Performance.** We report the accuracy (%) for the Countdown task across two base models (Llama3.2-1B and Llama3.2-3B) with several baseline models. Best results for each base model are **bolded**.

focusing on problem instances with four input numbers and target values ≤ 100. The training set consisted of 500,000 samples, balanced between optimal solutions and backtracking data. The test set was systematically partitioned into two distinct categories: one comprising seen targets paired with novel input combinations (denoted as Seen Targets), and the other incorporating entirely new targets (denoted as New Targets), consistent with the setting outlined in SoS (Gandhi et al. 2024). Each category contained 5,000 instances.

Maze Navigation. This planning task requires the LLM-based agent to find the shortest path from a starting position to a goal while avoiding obstacles in a grid-based environment. Our experimental data is derived from the publicly available maze dataset constructed by Searchformer (Lehnert et al. 2024). We systematically vary the maze sizes (ranging from 20×20 to 30×30 grids) to create a test environment with graduated difficulty levels, enabling quantitative assessment of planning performance across different complexity scenarios. Each maze contains randomly distributed walls occupying 30-50% of the cells. To ensure rigorous evaluation, all maze configurations are unique. We utilize the first 100,000 samples from the public dataset's training set for model training, and the first 1,000 samples from its test partition for evaluation.

Data Construction. For the Countdown task, we employ a recursive exhaustive search to construct solutions, deliberately introducing exploration errors, computational errors, and rule violations—each marked with `<backtrack>` tokens. In Maze Navigation, positive samples derive from optimal A* paths, while negative samples are generated by terminating deviated exploration trajectories with backtracking tokens. Both datasets maintain balanced error type ratios. More details of data construction are provided in Appendix.

Comparison Methods. In the Countdown task, we employ Llama3.2-1B (Dubey et al. 2024) and Llama3.2-3B (Dubey et al. 2024) as the base models. To ensure model diversity, we additionally compare Qwen3-0.6B (Yang et al. 2025a) with Qwen3-4B (Yang et al. 2025a), with experimental results presented in Appendix. The comparative experiments primarily consist of three categories of methods: The first category involves supervised fine-tuning using only optimal solution dataset $\mathcal{D}_{op}$, and compares two typical sampling strategies, namely greedy search and beam search (beams=16). The second category models the data as preference data pairs and compares various RLHF algorithms, including DPO (Rafailov et al. 2023), KTO (Ethayarajh et al. 2024), and the Best-of-N (Li et al. 2022) selection method based on the outcome reward model. The third category is based on backtracking data and compares the greedy sampling strategy. Furthermore, we compare DeepSeek-R1-like models trained with reinforcement learning, such as TinyZero (Pan et al. 2025) and MiniR1 (Schmid 2025). Additionally, we evaluate methods that learn from search trajectories, including SoS (Gandhi et al. 2024) and GSoS (Moon, Park, and Song 2024), using their reported optimal performances. In the Maze Navigation task, we conduct experiments on Llama3.2-1B (Dubey et al. 2024), primarily comparing our self-backtracking method with both optimal path SFT and Searchformer (Lehnert et al. 2024). Notably, since

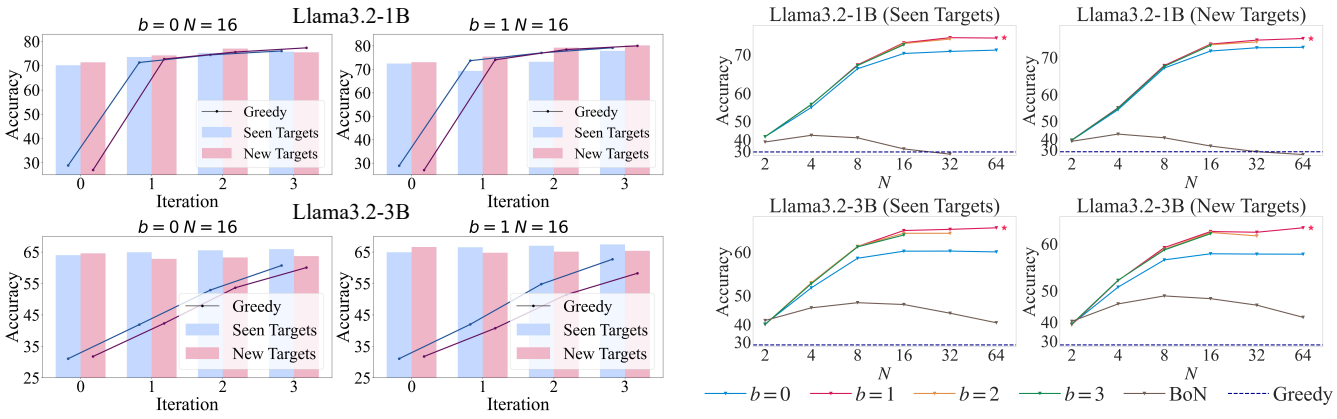

Figure 2: The left panel presents the self-improvement results and the right panel illustrates the performance curves when varing N and b on the Countdown task.

the original Searchformer did not report accuracy metrics (using pass@64 instead) and employed a different model architecture, we reproduce Searchformer's performance using identical experimental data and model for fair comparison.

Experimental Details. Our self-backtracking algorithm is implemented using PyTorch with Deepspeed Stage 2 optimization. Training is conducted on four NVIDIA A800 GPUs, using a base learning rate of 1e-5 over three epochs. Model-specific precision is applied: FP32 for Llama3.2-1B and BF16 for Llama3.2-3B. Detailed training specifications and baseline implementations are provided in Appendix. During the inference phase, we employ beam search with a temperature of 0.7 for our method and baselines involving sampling. For an analysis of temperature stability, additional experiments are provided in Appendix.

Self-Backtracking Boosts Reasoning

Main Results. In Table 1, we present the accuracy of various methods across different models for the Countdown task. Overall, the self-backtracking technique demonstrates a significant improvement over the baseline of greedy search after SFT, with enhancements of approximately over 30% on Llama3.2-3B and over 40% on Llama3.2-1B. Notably, our method exhibits considerable advantages even when $b = 0$, i.e., without backtracking, suggesting that the `<backtrack>` token can implicitly assess the quality of the current state, effectively substituting the function of the reward model. Additionally, when $b = 1$, we find that performance further improves, indicating that backtracking to previous states enables the model to leverage search mechanisms to explore correct answers more effectively.

Self-backtracking Can Self-improve. We do further experiments to demonstrate that our algorithm can self-improve. Employing self-backtracking with configurations $b = 0, N = 16$ and $b = 1, N = 16$ on two base models and datasets respectively, we filtered correct reasoning paths from inference outputs to do expert iteration. Left panel of Figure 2 shows three-round improvement results, where bars indicate test performance using our self-backtracking during inference and lines represent the results of single-pass greedy decoding. We find that the greedy decoding achieves remarkable improvements: +40% after first iteration for Llama3.2-1B (approaching the self-backtracking performance), ultimately surpassing the initial performance by 50% relative gain after third iteration. Similar results are observed for Llama3.2-3B. Notably, $b = 1$ significantly outperforms $b = 0$, confirming backtracking's importance. These results demonstrate that our method successfully transfers search-based inference capabilities to greedy decoding through expert iteration. This enables the model to maintain high accuracy while directly generating optimal solutions and achieves output efficiency on par with SFT.

Method	**Acc(%)**	**#Tokens**
SoS (Gandhi et al. 2024)	55.45	3160
GSoS (Moon, Park, and Song 2024)	68.10	591
TinyZero (Pan et al. 2025)	61.80	402
Mini-R1 (Schmid 2025)	50.00	300
Ours ($b = 1, N = 8$)	67.81	192
Ours ($b = 1, N = 16$)	72.76	355
Ours (after self-improvement)	**79.63**	**34**

Table 2: Comparison of more baselines across accuracy and average output token count.

Analysis for Output Efficiency. We present a comparative analysis between our proposed self-backtracking method, learn-from-trajectories approaches, and two RL-based methods in Table 2. The results of our method represent the average accuracy on Seen Targets and New Targets, where "Ours (after self-improvement)" denotes the greedy decoding results obtained after three rounds of expert iteration under the $b = 1, N = 16$ configuration. For fair evaluation of computational costs, we normalize the additional rollout computations in our basic self-backtracking method (without self-improvement) to equivalent output token counts. The analysis demonstrates that our approach

Method	20×20 Grids			30×30 Grids		
	Accuracy (%)	#Tokens	BR (%)	Accuracy (%)	#Tokens	BR (%)
Optimal Path SFT	62.20	176	-	50.80	318	-
SearchFormer (Lehnert et al. 2024)	95.20	3677	-	88.10	7953	-
Self-Backtracking (Greedy)	89.30	**151**	6.0	83.00	**239**	11.5
Self-Backtracking ($b = 0, N = 2$)	94.00	268	1.7	88.00	350	5.9
Self-Backtracking ($b = 1, N = 2$)	94.20	317	0.9	88.20	395	4.0
Self-Backtracking ($b = 0, N = 8$)	94.30	1763	**0.1**	91.80	2075	0.7
Self-Backtracking ($b = 1, N = 8$)	93.10	3952	**0.1**	91.90	4762	**0.1**
Self-Backtracking (Self-improvement-1)	<u>96.20</u>	<u>154</u>	1.1	94.20	259	2.3
Self-Backtracking (Self-improvement-2)	95.90	157	<u>0.6</u>	<u>94.80</u>	<u>255</u>	1.0
Self-Backtracking (Self-improvement-3)	**96.50**	156	0.9	**95.60**	256	<u>0.3</u>

Table 3: **Self-Backtracking Improves Planning Performance.** Accuracy (%), token consumption (#Tokens), and backtrack token ratio (BR) for Maze Navigation across methods. Best results are **bolded**; second-best are <u>underlined</u>.

maintains competitive performance while reducing much computational overhead compared to alternatives. Notably, after self-improvement, our method achieves an average output count of just 34 tokens, significantly shorter than all baselines. This validates the superiority of our approach.

Analysis for Different b and N. We conduct experiments by varying the b and N, and generate performance curves under different b values as N increases, as illustrated in Figure 2. The results show that the performance of BoN initially increases and then decreases with larger N, which we attribute to the reward hacking. On the contrary, our method exhibits a consistent improvement with increasing N, eventually stabilizing, indicating a clear test-time scaling law in breadth. Furthermore, when backtracking is permitted ($b = 1$), the performance improves more rapidly with N and achieves a higher overall performance, underscoring the necessity of backtracking. Surprisingly, increasing b does not result in a significant scaling phenomenon in depth. This is because the diversity of outputs from secondary backtracking significantly decreases, leading to only marginal improvements compared to $b = 1$. Overall, these results confirm that our approach scales effectively in both breadth and depth while maintaining controllable costs.

Ablation and Further Analysis. We include further experimental analysis and ablation studies in Appendix, covering error type categorization, the impact of training data ratios and how temperature affects the performance.

Self-Backtracking Boosts Planning

Main results. Table 3 presents a comprehensive comparison of various methods on the Maze Navigation task, showing substantial improvements achieved by our self-backtracking approach over baseline methods. Accuracy measures the percentage of correctly predicted paths. For 20×20 mazes, the optimal solution rate increases from 62.20% (SFT) to 93.10% ($b = 1, N = 8$). Notably, even without explicit backtracking ($b = 0$), the implicit state evaluation enabled by the backtrack token alone boosts the accuracy from 50.8% to 88.0% on 30×30 mazes, indicating the model's acquired capability to autonomously identify invalid exploration paths. Searchformer (Lehnert et al. 2024), which utilizes entire trajectories, exhibits lower efficiency compared to our approach. Through comprehensive self-improvement experiments conducted under the configuration $b = 1, N = 8$, our approach maintains superior accuracy across all iterations of self-improvement. Notably, our method exhibits consistently lower token consumption, even surpassing the efficiency of optimal-path SFT. This empirical evidence suggests that our model possesses the capability to identify more concise and optimal solution paths.

Analysis for Backtrack Token Ratio. We further report the backtrack token ratio (BR), defined as the proportion of the <backtrack> token in the model's output after training, for our self-backtracking method in Table 3. In contrast to the Countdown task, we observe a lower BR in the Maze Navigation task under greedy decoding, with only 6% for 20×20 and 11.5% for 30×30 mazes. This indicates that despite maintaining a 1:1 ratio between optimal solutions and backtracking data in the training set, the model exhibits a stronger preference for predicting optimal solutions on this benchmark—a finding consistent with Ye et al.'s results (Ye et al. 2024). Notably, our method achieves an 8.9% improvement over greedy decoding on 30×30 mazes when $b = 1$ and $N = 8$, suggesting that most error samples containing the backtrack token are resolved by our inference algorithm.

Conclusion

In this study, we propose a novel Self-Backtracking technique that addresses critical limitations in current learning from search tracjectories methods of autoregressive LLMs by enabling them to internalize the search process, particularly the ability to autonomously determine when to backtrack. Experimental evaluations conducted on both the Countdown task and Maze Navigation task demonstrate statistically significant improvements in both performance and computational efficiency.

Acknowledgments

This research was supported by the Jiangsu Science Foundation (BK20243012, BK20232003, BG2024036), Natural Science Foundation of China (62576162), and the Fundamental Research Funds for the Central Universities (022114380023).

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

An, S.; Ma, Z.; Lin, Z.; Zheng, N.; Lou, J.-G.; and Chen, W. 2023. Learning from mistakes makes llm better reasoner. *arXiv preprint arXiv:2310.20689*.

Choi, S.; Fang, T.; Wang, Z.; and Song, Y. 2023. KCTS: Knowledge-Constrained Tree Search Decoding with Token-Level Hallucination Detection. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Cundy, C.; and Ermon, S. 2024. Sequencematch: Imitation learning for autoregressive sequence modelling with backtracking.

DeepSeek. 2024. Deepseek-r1-lite-preview is now live: unleashing supercharged reasoning power!

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ethayarajh, K.; Xu, W.; Muennighoff, N.; Jurafsky, D.; and Kiela, D. 2024. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

Gandhi, K.; Chakravarthy, A.; Singh, A.; Lile, N.; and Goodman, N. D. 2025. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*.

Gandhi, K.; Lee, D.; Grand, G.; Liu, M.; Cheng, W.; Sharma, A.; and Goodman, N. D. 2024. Stream of Search (SoS): Learning to Search in Language. *First Conference on Language Modeling*.

Graves, A. 2012. Sequence Transduction with Recurrent Neural Networks. arXiv:1211.3711.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*.

Lehnert, L.; Sukhbaatar, S.; Su, D.; Zheng, Q.; McVay, P.; Rabbat, M.; and Tian, Y. 2024. Beyond A*: Better Planning with Transformers via Search Dynamics Bootstrapping.

Li, Y.; Lin, Z.; Zhang, S.; Fu, Q.; Chen, B.; Lou, J.-G.; and Chen, W. 2022. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*.

Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; and Cobbe, K. 2024. Let's verify step by step.

Moon, S.; Park, B.; and Song, H. O. 2024. Guided Stream of Search: Learning to Better Search with Language Models via Optimal Path Guidance. *arXiv preprint arXiv:2410.02992*.

OpenAI. 2024. Learning to Reason with Large Language Models.

Pan, J.; Zhang, J.; Wang, X.; Yuan, L.; Peng, H.; and Suhr, A. 2025. TinyZero. https://github.com/Jiayi-Pan/TinyZero.

Plaat, A.; Wong, A.; Verberne, S.; Broekens, J.; van Stein, N.; and Back, T. 2024. Reasoning with large language models, a survey. *arXiv preprint arXiv:2407.11511*.

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

Rafailov, R.; Sharma, A.; Mitchell, E.; Manning, C. D.; Ermon, S.; and Finn, C. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*.

Schmid, P. 2025. Mini-R1: Reproduce Deepseek R1 Aha Moment a RL Tutorial. https://www.philschmid.de/mini-deepseek-r1.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shao, J.-J.; Hao, H.-R.; Yang, X.-W.; and Li, Y.-F. 2024a. Learning for long-horizon planning via neuro-symbolic abductive imitation. In *Knowledge Discovery and Data Mining*.

Shao, J.-J.; Yang, X.-W.; Zhang, B.-W.; Guo, L.-Z.; and Li, Y.-F. 2024b. Chinatravel: A real-world benchmark for language agents in chinese travel planning.

Snell, C.; Lee, J.; Xu, K.; and Kumar, A. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Su, D.; Sukhbaatar, S.; Rabbat, M.; Tian, Y.; and Zheng, Q. 2024. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. *arXiv preprint arXiv:2410.09918*.

Sullivan, R.; and Elsayed, N. 2024. Can Large Language Models Act as Symbolic Reasoners? *arXiv preprint arXiv:2410.21490*.

Teller, V. 2000. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition.

Tian, Y.; Peng, B.; Song, L.; Jin, L.; Yu, D.; Han, L.; Mi, H.; and Yu, D. 2024. Toward self-improvement of llms via imagination, searching, and criticizing. *Advances in Neural Information Processing Systems*, 52723–52748.

Tong, Y.; Li, D.; Wang, S.; Wang, Y.; Teng, F.; and Shang, J. 2024. Can LLMs Learn from Previous Mistakes? Investigating LLMs' Errors to Boost for Reasoning. *arXiv preprint arXiv:2403.20046*.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*.

Wan, Z.; Feng, X.; Wen, M.; McAleer, S. M.; Wen, Y.; Zhang, W.; and Wang, J. 2024. Alphazero-like Tree-Search can Guide Large Language Model Decoding and Training.

Wang, C.; Deng, Y.; Lyu, Z.; Zeng, L.; He, J.; Yan, S.; and An, B. 2024a. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*.

Wang, P.; Li, L.; Shao, Z.; Xu, R.; Dai, D.; Li, Y.; Chen, D.; Wu, Y.; and Sui, Z. 2024b. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 9426–9439.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; and Zhou, D. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *International Conference on Learning Representations*.

Wu, Y.; Sun, Z.; Li, S.; Welleck, S.; and Yang, Y. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *Advances in Neural Information Processing Systems*.

Xie, Y.; Goyal, A.; Zheng, W.; Kan, M.-Y.; Lillicrap, T. P.; Kawaguchi, K.; and Shieh, M. 2024a. Monte Carlo Tree Search Boosts Reasoning via Iterative Preference Learning.

Xie, Y.; Kawaguchi, K.; Zhao, Y.; Zhao, J. X.; Kan, M.-Y.; He, J.; and Xie, M. 2024b. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*.

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

Yang, M. S.; Schuurmans, D.; Abbeel, P.; and Nachum, O. 2022. Chain of thought imitation with procedure cloning. *Advances in Neural Information Processing Systems*, 36366–36381.

Yang, X.-W.; Shao, J.-J.; Guo, L.-Z.; Zhang, B.-W.; Zhou, Z.; Jia, L.-H.; Dai, W.-Z.; and Li, Y.-F. 2025b. Neuro-Symbolic Artificial Intelligence: Towards Improving the Reasoning Abilities of Large Language Models. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 10770–10778.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*.

Ye, T.; Xu, Z.; Li, Y.; and Allen-Zhu, Z. 2024. Physics of Language Models: Part 2.2, How to Learn From Mistakes on Grade-School Math Problems. *arXiv preprint arXiv:2408.16293*.

Yuan, W.; Pang, R. Y.; Cho, K.; Sukhbaatar, S.; Xu, J.; and Weston, J. 2024. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*.

Yue, Y.; Chen, Z.; Lu, R.; Zhao, A.; Wang, Z.; Song, S.; and Huang, G. 2025. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? *arXiv preprint arXiv:2504.13837*.

Zhang, D.; Zhoubian, S.; Hu, Z.; Yue, Y.; Dong, Y.; and Tang, J. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.

Zhang, X.; Du, C.; Pang, T.; Liu, Q.; Gao, W.; and Lin, M. 2024b. Chain of Preference Optimization: Improving Chain-of-Thought Reasoning in LLMs. *Advances in Neural Information Processing Systems*.

Zhang, Y.; Chi, J.; Nguyen, H.; Upasani, K.; Bikel, D. M.; Weston, J.; and Smith, E. M. 2024c. Backtracking improves generation safety. *arXiv preprint arXiv:2409.14586*.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Zhou, Z.; Tan, Y.; Li, Z.; Yao, Y.; Guo, L.-Z.; Li, Y.-F.; and Ma, X. 2025. A Theoretical Study on Bridging Internal Probability and Self-Consistency for LLM Reasoning. In *Advances in Neural Information Processing Systems*.

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