

How Much Backtracking is Enough? Exploring the Interplay of SFT and RL in Enhancing LLM Reasoning

Hongyi James Cai^{*1}, Junlin Wang^{*1}, Xiaoyin Chen^{2,3}, Bhuwan Dhingra¹

¹Duke University, ²Mila - Québec AI Institute, ³Université de Montréal
{hongyi.cai, junlin.wang2}@duke.edu, xiaoyin.chen@mila.quebec,
bdhingra@cs.duke.edu

Abstract

Recent advancements in large language models (LLMs) suggest that reinforcement learning (RL) effectively internalizes search strategies, yielding significant improvements on challenging reasoning tasks through extended chains of thought. While backtracking is widely viewed as the core mechanism enabling this improvement, its precise dynamics and how SFT and RL interact to optimize for it, remain poorly understood. In this work, we argue that backtracking—the ability to explicitly revisit and revise earlier reasoning steps—is a key operator that enables the transition from linear generation to non-linear tree search for LLMs, unlocking strong performance on combinatorially complex tasks. We motivate this claim by first studying the interplay between supervised fine-tuning (SFT) warm-up and RL across eight reasoning tasks: Countdown, Sudoku, Arc 1D, Advanced Geometry, Color Cube Rotation, List Functions, Zebra Puzzles, and Self Reference. We find that a regular SFT warm-up using self-sampled CoT sequences provides a modest boost to subsequent RL compared to training without any SFT warm-up; however, such gains saturate as tasks become more difficult. Motivated by this failure mode, we introduce a backtracking-centric training recipe. By synthetically varying the number of explicit backtracking steps in the SFT warm-up, we show that (i) longer CoTs containing backtracks stabilize and amplify RL, and (ii) the optimal backtrack depth scales with task difficulty—zero for Arc 1D, one for Countdown, and five or more for Sudoku. Our findings establish that principled backtracking is essential for improving the model’s reasoning capability.

1 Introduction

Recent large language models such as DeepSeek-R1 (Guo et al., 2025) and OpenAI’s o1 (OpenAI, 2024) have demonstrated remarkable reasoning abilities on various complex tasks. Their success stems from allocating substantially more inference-time compute, letting the model perform an internal search via extended chains of thought before committing to an answer (Snell et al., 2024). Reinforcement learning has emerged as the most effective way to unlock these long reasoning traces (Shao et al., 2024; Team et al., 2025; Schulman et al., 2017; Zelikman et al., 2022; Rafailov et al., 2023). In the DeepSeek-R1 era, a key shift is from open-ended process rewards that score intermediate steps (Lightman et al., 2024; Guan et al., 2025; Zhang et al., 2025; Zheng et al., 2025) to verifiable, rule-based rewards that check final answers (Zeng et al., 2025; Chen et al., 2025), providing a sharper signal that steers exploration toward correct trajectories.

Although reinforcement learning often improves LLM reasoning, the scope and limits of the underlying enhancement mechanisms remain under-explored. Gandhi et al. (2025) highlight this by showing large RL gains for Qwen but only marginal ones for Llama. One hypothesis attributes Qwen’s advantage to the use of “thinking” tokens such as “wait” and “verify” (Xie et al., 2025; Gandhi et al., 2025). More broadly, RL appears to primarily amplify behaviors

^{*}Equal contribution.

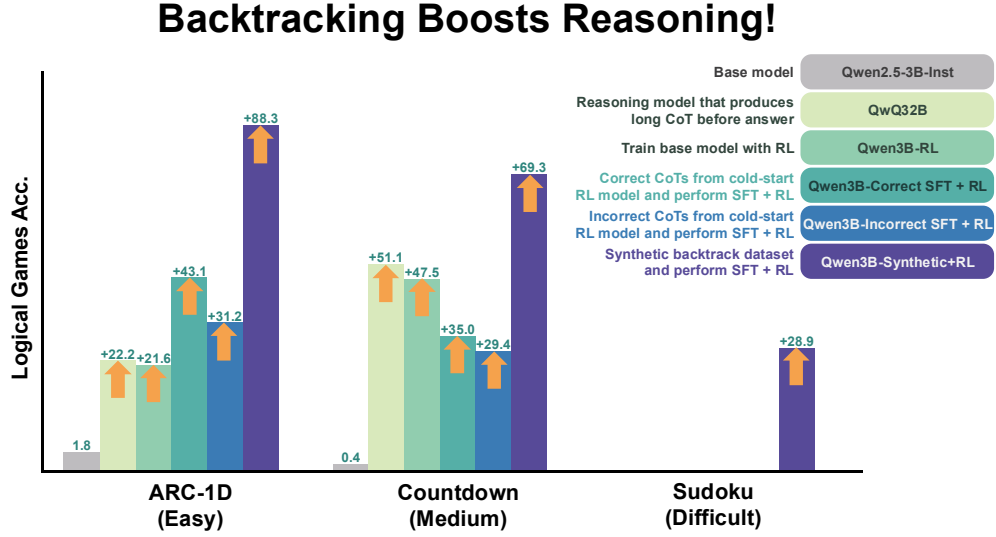


Figure 1: We engineer SFT data with synthetic backtracks to achieve significant improvements over three reasoning tasks, Arc-1D, Countdown, and Sudoku, each representing a different difficulty.

already present in the base model—either inherited from pretraining (Zhao et al., 2025) or generalized from SFT warm-up (Chu et al., 2025)—rather than consistently improving accuracy through longer traces alone (Qin et al., 2025; Wang et al., 2025; Yue et al., 2025).

This motivates a closer look at backtracking, which increasingly emerges in top reasoning models such as DeepSeek-R1 (Guo et al., 2025) and OpenAI’s o1 series (OpenAI, 2024), and suggests a shift from *reasoning-as-generation* to *reasoning-as-search*. In classical cognitive science, problem solving is often framed as search through a state space (Newell & Simon, 1972); for autoregressive LLMs, backtracking is the key operator that breaks strict linear generation by treating prior outputs as revisable state assignments rather than immutable facts. Viewed through this lens, backtracking is not a task-specific trick but a general operator for tree search—central to exploration, planning, and multi-hop reasoning (Qin et al., 2025). It ties together detecting a flawed partial solution (verification) and revising it (recovery), yielding a concrete, controllable handle on search behavior. This perspective motivates our study of how SFT warm-up interacts with RL via backtracking frequency: when is warm-up necessary, which warm-ups matter, and how much backtracking is optimal before it becomes counter-productive?

To answer these questions, we run controlled experiments that isolate how the *presence* and *structure* of warm-up data shape downstream RL. We first benchmark pure RL (no reasoning specific SFT warm-up) against cold-start RL using self-sampled CoTs as warm-up, showing that simple warm-ups can help but become insufficient on harder tasks. We then introduce synthetic warm-ups with explicit, tunable backtracking (constructed via depth-first search or heuristic search for three representative tasks) and show that backtracking is a reliable knob for unlocking RL.

Taken together, our controlled experiments reveal a precise recipe for effective RL: initializing with synthetic SFT data containing an appropriate number of backtracks—matched to the difficulty of the task—consistently pushes the model beyond its base reasoning capabilities.

Our main contributions are:

Table 1: Brief explanation of each SFT warm-up setup explored in this paper.

ABBR.	EXPLANATION
NO-SFT	RL WITH NO SFT (PURE RL)
SELF-SAMPLED SFT	SFT ON MODEL-GENERATED TRAJECTORIES
SYNTHETIC BACKTRACK SFT	SFT ON SYNTHETIC TRAJECTORIES WITH BACKTRACKING
SHUFFLED SFT	SFT ON TRAJECTORIES SHUFFLED ACROSS PROBLEMS

- A controlled comparison of pure RL vs. cold-start RL across eight reasoning tasks, clarifying when standard self-sampled CoT warm-ups help and when they plateau (Section 3).
- A backtracking-centric SFT+RL recipe based on synthetic trajectories with calibrated backtrack depth, which substantially improves performance on combinatorially hard problems (Section 4).
- Additional analysis showing that RL can still benefit from format-aligned but semantically incorrect CoT warm-ups (Section 5).

2 Methodology

In this study, we conduct a comprehensive analysis using controlled experiments to investigate the interplay between SFT and RL, focusing on how different training data mixtures influence RL outcomes. Specifically, we examine which types of SFT warm-ups benefit RL and seek to understand the underlying reasons. Table 1 presents the SFT setups we evaluate: no SFT, self-sampled SFT, synthetic backtracking SFT, and shuffled SFT. Ultimately, we aim to equip the model with the ability to tackle difficult problems beyond its base capabilities.

We employ a total of 8 reasoning tasks, adopted from Reasoning Gym (Stojanovski et al., 2025), and select a subset to further curate synthetic data to ensure our experiments provide analytical insights.

2.1 Why backtracking matters

In the context of LLM reasoning, we define backtracking as a formal state-space traversal operator. Let a reasoning task be defined by a state space S and a set of transitions T . A standard chain-of-thought approach attempts to produce a path $P = (s_0, s_1, \dots, s_n)$ such that s_n is a goal state. For complex tasks, the probability that every local transition $p(s_{i+1} | s_i)$ is correct can be low, implying a cumulative success probability that decays quickly with path length.

We frame backtracking as a transition $T_{\text{back}} : s_i \rightarrow s_j$ where $j < i$. This operator serves two key cognitive functions: (i) **validation-driven pruning**, where the model evaluates a partial state s_i against task constraints and prunes a branch upon detecting an inevitable dead-end; and (ii) **state recovery**, where the model restores a previous valid state s_j to explore an alternative transition $s_j \rightarrow s'_{j+1}$. By varying the backtrack depth in our synthetic SFT data, we effectively condition the model to recognize its search frontier and dynamically allocate computation during inference to explore alternative paths. Our framework allows backtracking frequency to be treated as a hyperparameter for Search-Aware SFT.

Finally, many common test-time scaling strategies can be viewed as implicit forms of backtracking: **parallel sampling** explores alternative branches by generating multiple candidate continuations (i.e., shallow backtracks across independent rollouts), and **self-reflection** provides the validation signal that a current trajectory is flawed (Kim et al., 2025). In both cases, verification or reflection can indicate that a current path is incorrect, but explicit backtracking is the recovery action that returns to a prior state to pursue a different transition.

2.2 Reasoning Games

The puzzle-style reasoning tasks we choose in this study encapsulates many essential properties of reasoning: multi-step deductive reasoning, symbolic manipulation under constraints, search over discrete combinatorial structures, and systematic backtracking when partial solutions fail. These are the same algorithmic primitives that underlie mathematical proofs and algebraic transformation, enabling our study to provide a proxy for broader scope of tasks. The specific reasoning games chosen are Countdown, Advanced Geometry, Arc-1D, Sudoku, Color Cube Rotation, Zebra Puzzles, List Functions, and Self Reference (Appendix B).

2.3 Model and training

We use the Qwen2.5 family of models (Team, 2024), primarily focusing on Qwen2.5-3B-Instruct for our pure-RL and self-sampled-SFT experiments. For our synthetic-backtracking experiments on Arc-1D, Countdown, and Sudoku, we show results with Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Llama3.2-3B-Instruct (Grattafiori et al., 2024). We report pass@1 accuracy, evaluating only the correctness of the final answer placed inside `<answer></answer>` tokens.

For both SFT and RL training, we adopted code from Sheng et al. (2025); Pan et al. (2025), the specific rollout lengths depend on whether the model is initialized from long or short CoT. Typically, for RL training initialized from long CoTs, we use rollout length from 4k to 8k tokens, depending on the tasks. For short CoTs, we use rollout length from 1k to 2k tokens, depending on the tasks (Appendix C).

We adopt rule-based rewards (Guo et al., 2025), where format accuracy (successfully generate a pair of `<think></think>` and `<answer></answer>` tokens with think tokens come before answer tokens) is rewarded 0.1 point, and the answer accuracy-exact match with the ground truth-is rewarded 0.9 point (Appendix D). Together, a problem may be rewarded a maximum of 1.0 point and the minimal of 0.0 point.

3 Maximizing base model reasoning capabilities

To systematically investigate how SFT interacts with reinforcement learning in enhancing large language models’ reasoning capabilities, we scientifically address a foundational question: is a warm-up stage necessary for effective RL? Specifically, we begin with two distinct approaches—pure RL (no SFT warm-up) and cold-start RL with a short self-sampled CoT warm-up—to establish baselines and evaluate their efficacy across diverse reasoning tasks. These experiments demonstrate that, although pure RL is already effective, incorporating a simple self-sampled CoT warm-up further improves performance. However, both methods face significant limitations when applied to more challenging reasoning tasks, such as Sudoku.

3.1 Pure RL - works albeit limited

We first explored pure RL, a direct RL approach without any reasoning specific SFT warm-up. The motivation behind this was to establish a clear baseline and determine whether the model could independently internalize effective search strategies and concise reasoning through pure RL exposure. In Figure 2a, we observe that just pure RL itself is able to witness improvements in model’s reasoning abilities, compared to their base model. We manually inspect a few of the generated trajectories and find that pure RL enables the model to internalize rudimentary strategies, such as explicit verbalized reasoning (e.g., in Appendix G.2) and concise thinking (e.g., in Appendix G.1). It is noticeable that for most tasks, pure RL naturally produces linear trajectories without backtracking. For Countdown, the model produces backtracking but it is unstructured. Due to probable pre-training data exposure, the model occasionally attempts to correct itself. However, these corrections are stochastic and often “lazy”—retrying the whole calculation rather than pinpointing the error.

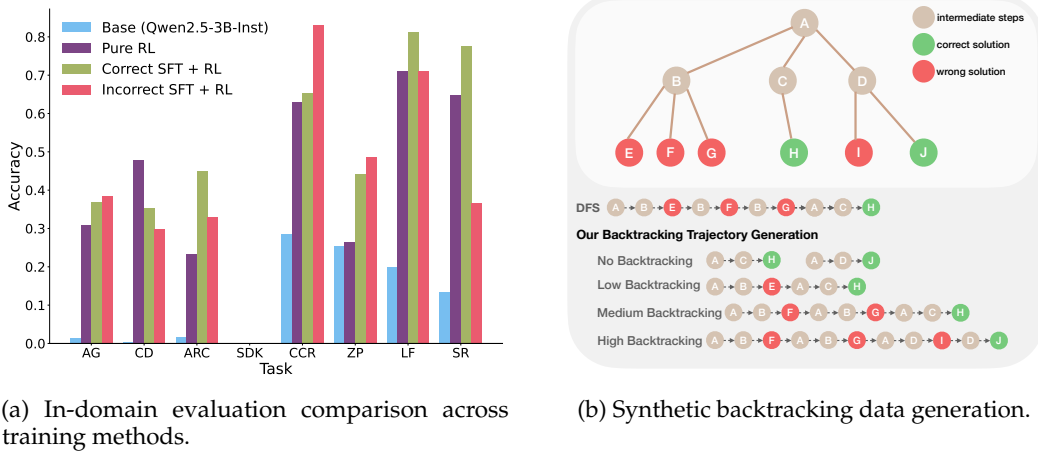


Figure 2: **(Left)** In-domain evaluation comparison between base model, pure RL, correct self-sampled SFT + RL, and incorrect self-sampled SFT + RL. **(Right)** Backtracking trajectory generation with varying depths.

Additionally, training the model in one environment also improves performance in other tasks as shown in Appendix J. Training in Advanced Geometry environment induces good generalization to Zebra Puzzles and List Functions; training in Countdown induces generalization to Advanced Geometry. However, limitations quickly became apparent, particularly on combinatorial problems like Sudoku and Zebra Puzzles, indicating that pure RL alone is insufficient for complex reasoning tasks.

3.2 Self-sampled short SFT as warm-up goes beyond pure RL

Building on the insights from pure RL, we hypothesized that a supervised fine-tuning warm-up using short self-sampled CoTs could further enhance reasoning performance by priming models with initial cognitive strategies.

Training and evaluating setup In this series of experiments, we first perform specialized pure RL training starting from Qwen2.5-3B-Instruct (Team, 2024) across 8 reasoning datasets (Stojanovski et al., 2025). After obtaining 8 RL’ed models, we subsequently collect their reasoning trajectories on the same tasks they are finetuned on, but with problems generated from a different seed (different problems). We purposefully separate the trajectories that lead to correct answer and those that lead to incorrect answers, from which we further perform the complete SFT + RL post-training pipeline starting again from vanilla Qwen2.5-3B-Instruct, and obtain 15 specialized models (7 RL’ed models initialized from correct SFT, and the other 8 initialized from incorrect SFT). We maintain identical gradient steps between pure RL and self-sampled SFT + RL to ensure fair comparison. A comprehensive evaluation is subsequently performed for all 23 models (three models per task with the exception of Sudoku, which we discuss below) in Figure 2a.

Self-sampled CoTs are effective warm-ups Short self-sampled CoT warm-up consistently demonstrated performance gains across nearly all tasks compared to pure RL in Figure 2a. Specifically, we observed significant performance improvements on tasks such as ARC and List Functions. It is also worth noting that even incorrect trajectories warm-up can improve downstream RL performance on some of the tasks. We will discuss more details in later section.

Notably, tasks like Sudoku still posed substantial challenges, highlighting the limitations of self-sampled CoTs for deeper, combinatorial reasoning. In fact, the cold-start RL pipeline effectively breaks down for Sudoku because the model cannot self-generate the necessary correct SFT initialization data, which illustrates a fundamental brittleness of the cold-start

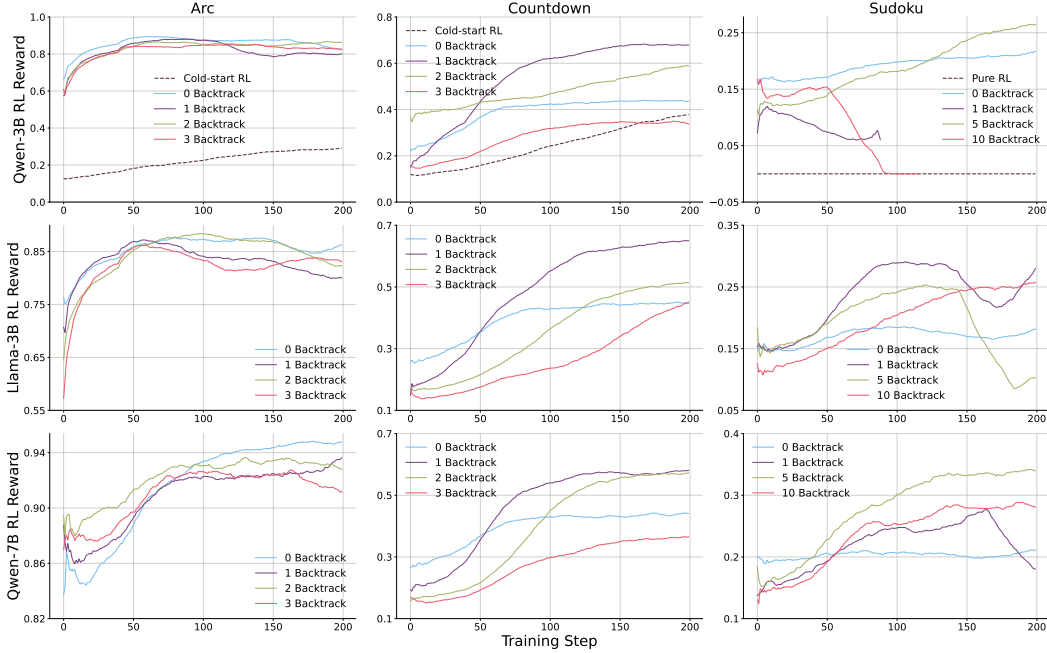


Figure 3: RL training rewards for Arc-1D, Countdown and Sudoku across Qwen2.5-3B-Instruct, Llama3.2-3B-Instruct, and Qwen2.5-7B-Instruct. We plot the RL reward curves corresponding to different backtracking warm-up strategies. The results indicate that distinct tasks require specific backtracking configurations for optimal performance.

RL paradigm. These results underscore the incremental yet limited nature of cold-start RL, motivating further exploration of more sophisticated strategies.

4 Pushing beyond boundaries with backtracking

Recognizing the limitations identified in previous methods, particularly the failure of both pure RL and short-CoT warm-ups on combinatorially hard tasks such as Sudoku, we propose a backtracking-centric training paradigm to push models beyond their baseline reasoning limits. Empirical analyses of the strongest reasoning models reveal spontaneous use of backtracking (e.g., self-correct, multiple attempts), suggesting that explicit exposure to backtracking traces during SFT can seed deeper search policies that subsequent RL can refine. We therefore hypothesize that integrating controlled amounts of backtracking into the SFT stage will enable models to internalize more powerful search strategies and unlock substantial gains on challenging tasks.

4.1 Building synthetic backtracking datasets

To systematically evaluate the impact of backtracking, we selected three representative reasoning tasks—Countdown, Arc 1D, and Sudoku—to construct synthetic datasets with varying numbers of backtrack, as well as optimal trajectories, and study whether, and how RL amplifies such behavior. The three tasks are selected because they are the more challenging subset based on vanilla model’s baseline performance in Figure 2a.

To construct synthetic datasets for Countdown and Sudoku, we use DFS solver and create a tree structure where each node represents an intermediate step (e.g., an operation between two numbers for Countdown; fill one grid with a valid number for Sudoku). Then we start from the solution node back to the root node to build optimal solutions, and include incorrect branches as detours and backtracks. For Arc 1D, we design a heuristic search that applies hand-crafted transformation functions with step-wise, verbalized reasoning. A

Table 2: Evaluation accuracies (%) for Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Llama3.2-3B-Instruct models post-trained on synthetic datasets across zero backtracks, low backtrack (1 backtrack), medium backtrack (2 backtracks for Arc and Countdown, 5 for Sudoku), and high backtrack (3 backtracks for Arc and Countdown, 10 for Sudoku). QwQ-32B is included as a baseline.

DATA	ARC-1D				COUNTDOWN				SUDOKU			
	QWEN 3B	QWEN 7B	LLAMA 3B	QWQ 32B	QWEN 3B	QWEN 7B	LLAMA 3B	QWQ 32B	QWEN 3B	QWEN 7B	LLAMA 3B	QWQ 32B
ZERO	90.1	94.1	91.5	-	38.9	38.6	37.6	-	14.4	11.8	9.6	-
LOW	86.9	92.1	85.3	-	69.7	59.4	63.0	-	0.0	20.8	23.4	-
MEDIUM	86.1	89.3	87.2	-	60.8	57.7	48.2	-	28.9	32.1	25.2	-
HIGH	84.8	91.4	82.6	-	32.4	31.4	44.4	-	0.0	28.0	26.9	-
BASELINE	-	-	-	24.0	-	-	-	51.5	-	-	-	0.0

detour or backtrack is created by choosing an incorrect transformation and then retrying (examples in Appendix H).

In all three tasks, we define a k -backtrack trajectory as one containing k explicit recovery events, where the reasoning enters an incorrect branch, returns to the most recent valid decision point, and then resumes along the correct solution path.

4.2 Quantifying the Impact of Backtracking in SFT Warm-up

Our controlled experiments reveal that the exposure to backtrack systematically improves RL training efficiency as shown in Figure 3. However, the number of backtracks needed depends on the difficulty of the problem:

1. **Countdown — Moderate.** Every puzzle provides 4–6 numbers and a target value, producing a medium-sized arithmetic search tree. Although vanilla Qwen2.5-3B-Instruct can generate plausible attempts, it only solves 0.4% of the problems. We found that using exactly **one** backtrack is the most optimal.
2. **Arc 1D — Moderate/Easy.** Tasks are solved by applying a small pool of grid-transformation heuristics; the same baseline attains 1.8% accuracy. Using **zero** backtrack is optimal.
3. **Sudoku — Hard.** Each board contains 30–60 empty cells; every cell admits multiple candidates, yielding an astronomically large search space. A vanilla Qwen2.5-3B-Instruct baseline solves none of the test instances. Using **five or more** backtracks is optimal.

Our controlled experiments with both Qwen and Llama model families confirm a consistent pattern: harder combinatorial problems require deeper backtracking traces to seed RL, whereas easier tasks are best served by shallow or even optimal demonstrations.

For **Countdown**, RL initialized with one backtrack achieves the highest reward curve for all three evaluated models (Figure 3), whereas zero backtracks or more frequent backtracking produce consistently worse training trajectories. This suggests that a one-backtrack initialization enables models to internalize and execute efficient search strategies. Evaluation results confirm this: the one-backtrack Qwen2.5-3B, Qwen2.5-7B, and Llama3.2-3B models attain 69.7%, 59.4%, and 63.0% accuracy respectively, all outperforming QwQ-32B (Team, 2025) at 51.5% with an 8k-token context window (Table 2). By contrast, their corresponding zero-backtrack models achieve only 38.9%, 38.6%, and 37.6% respectively.

For **Sudoku**, initializing PPO with five or ten backtracks is necessary for stable, effective training, depending on the model and scale. Too few or too many backtracks can produce unstable or degraded training. For both Qwen models, five-backtrack training performs best on Sudoku: Qwen2.5-3B achieves 28.9% versus 14.4% with zero backtracks, while Qwen2.5-7B achieves 32.1% versus 11.8%. For Llama, higher backtracking counts leads

to substantially better evaluation performance, with the ten-backtrack model achieving markedly higher accuracy than the one and five-backtrack variant (Table 2). The best-performing backtracking configuration for each of the three models outperforms the larger and more powerful reasoning model QwQ-32B, which achieves 0.0% (Table 2). Notably, pure RL fails to train on Sudoku, with reward trajectories stagnating at 0.1, highlighting the necessity of our method.

For **Arc 1D**, the easiest of the three tasks, performance is generally lower when backtracking is introduced. Zero-backtrack training performs best for all three models, achieving 90.1%, 94.1%, and 91.5% accuracy for Qwen2.5-3B, Qwen2.5-7B, and Llama3.2-3B, respectively, substantially exceeding QwQ-32B’s 24.0%

See Appendix E for additional model and task scope ablations.

5 Additional analysis of interplay between SFT and RL

In this section, we document additional findings and include analysis throughout our goal of breaking the base boundary of reasoning.

5.1 Incorrect trajectories are still useful

Models initialized from incorrect reasoning CoTs display similar behaviors with those from correct trajectories. Figure 2a shows that incorrect SFT models post RL see the same trend of performance increase or decrease with correct SFT models on in-domain evaluations, with the only exception being self reference task. While initially counter-intuitive, this finding aligns with recent evidence that RL can quickly overwrite semantic errors provided the demonstrations respect the expected CoT format (Setlur et al., 2024). We conjecture that syntactic scaffolding alone primes the base model with an executable search blueprint that subsequent RL can refine. Such result also holds with the larger Qwen2.5-7B-Instruct on Countdown (Appendix F).

5.2 RL is sensitive to shuffled SFT

To probe the limits of semantic noise that an SFT warm-up can tolerate, we deliberately broke the alignment between prompts and completions: within the best-performing backtracking dataset in Section 4, we randomly shuffled questions and their corresponding CoT-answer pairs. This manipulation preserves syntactic structure—every example still contains well-formatted trajectories while obliterating the underlying reasoning. From Figure 4a, we see that the training becomes extremely ineffective, suggesting that despite having the right reasoning structure, RL training is easily vulnerable to the internal inconsistency between prompt and completion during SFT. In fact, we find that, for the shuffled Countdown and Arc 1D runs in Figure 4a, models recover only a negligible reward signal after roughly 60–80 PPO updates before stagnating. These results contrast sharply with the resilience to incorrect but aligned CoTs observed in previous subsection, indicating that RL is highly sensitive to internal inconsistencies between prompts and a reasoning traces. Manual inspection of generated outputs (Appendix G.3) reveals that the policy over-fits to spurious prompt-completion pairings that PPO cannot unlearn.

6 Related Work

6.1 Test time scaling

Recent work has shown scaling up language model output length during test time can be more effective than scaling pretraining (Snell et al., 2024; Wang et al., 2024; Jiang et al., 2023; Saad-Falcon et al., 2024). As language models become increasingly larger and pre-train on increasingly large corpora; this third axis of scaling offers promising gains in performance while remaining budget friendly (Wang et al., 2025). There are mainly two directions for scaling up: parallel sampling and sequential search. Common parallel sampling methods

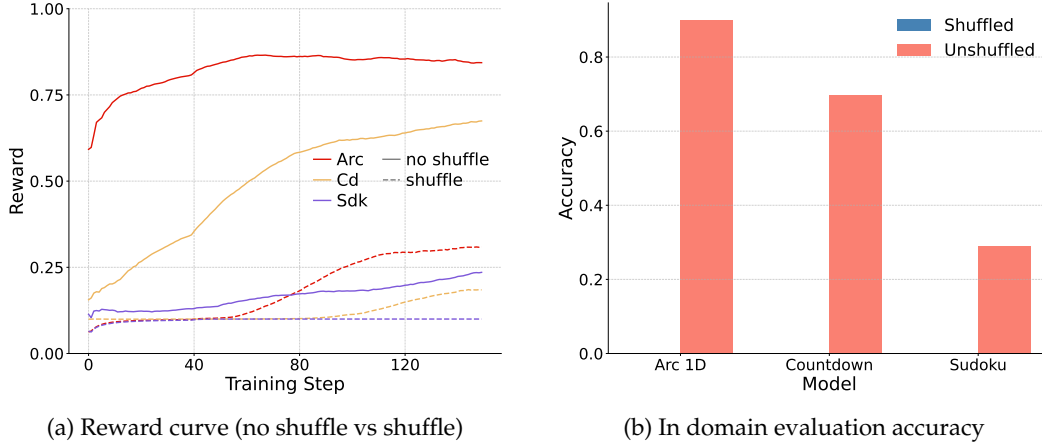


Figure 4: **(Left)** RL training reward trajectories for models post-trained on shuffled and non-shuffled synthetic datasets. **(Right)** In domain evaluation for each model post-trained on shuffled and not shuffled synthetic dataset of Arc 1D, Countdown, and Sudoku.

such as best-of-n rely on LLMs to propose answers independently N times (Li et al., 2024); whereas sequential search generates each response in sequence conditioned on previous attempt (Qu et al., 2024; Feng et al., 2023; Gandhi et al., 2024; Yang et al., 2025). There have also been attempts to combine the two and incorporate external verifiers through algorithms like Monte-Carlo Tree Search (Guan et al., 2025; Feng et al., 2023). Our work extends this line of research by examining the optimal training mixtures for efficiently scaling up model generation for reasoning.

6.2 Reinforcement Learning for reasoning

A novel dimension of scaling inference-time compute for language models has recently gained prominence through the emergence of specialized “reasoning models” trained via reinforcement learning (RL). Prior works predominantly explored on-policy and off-policy RL methods (Zelikman et al., 2022; 2024; Kazemnejad et al., 2025; Phan et al., 2023) and saw moderate successes. However, recent approaches such as DeepSeek-R1 (Guo et al., 2025) have revitalized interest in RL training utilizing verifiable rewards (RLVR), driven by Proximal Policy Optimization (PPO) and its memory-efficient variant, Group Relative Policy Optimization (GRPO) (Shao et al., 2024). A key driver behind RLVR is the emphasis on rule-based rewards derived directly from final outputs, replacing traditional reward models. This new RL training paradigm has notably revealed critical “aha moments,” where models spontaneously generate distinctive tokens such as “wait,” indicative of active internalization of search strategies and optimized trajectory discovery.

One prevailing hypothesis is that such fine-tuning allows language models to internalize search algorithms during inference-time explicitly. Methods such as Stream of Search (Gandhi et al., 2024), for instance, fine-tune models using linearized search trajectories, empowering continuous and coherent search capability within single-output generation (Yang et al., 2025; Lehnert et al., 2024). Alternatively, another hypothesis suggests that RL training utilizing verifiable rewards does not fundamentally induce novel reasoning skills but rather exploits and amplifies capabilities already learned during the pretraining phase (Yue et al., 2025). This claim is supported by observations that base pretrained models can achieve comparable or superior performance in metrics such as pass@k compared to their RL-trained counterparts. Nevertheless, the precise mechanisms behind these emergent reasoning capabilities remain poorly understood. Our research addresses this gap by systematically investigating how RL leverages SFT and the associated data mixture to unlock the reasoning capacities.

There are a few concurrent efforts investigating how fine-grained behavioral signals influence RL-trained reasoning models. [Gandhi et al. \(2025\)](#) analyzes four behavioral cues, demonstrating that such cues can accelerate RL convergence. We perform controlled experiments on more tasks and study various factors of SFT mixtures. [Qin et al. \(2025\)](#) examines whether explicit backtracking traces improve performance across CountDown and Sudoku. We go further by rigorously controlling trace correctness, format, and synthetic backtrack frequency. Finally, [Zhao et al. \(2025\)](#) explores how variations in pre-training corpus composition interact with subsequent post-training, whereas we emphasize on how post-training affects the reasoning ability.

7 Conclusion

In this work, we thoroughly inspect the interplay between SFT and RL for reasoning tasks through controlled experiments, emphasizing the role of training-data composition towards effective RL post-training. We found that Qwen2.5-3B-Instruct models, when performing pure RL, demonstrates two distinct reasoning pattern: verbalized searching and backtracking that iteratively tries to find the correct answer, and concise thinking which solves the problem in one go. To solve combinatorially hard problems like Sudoku, we incorporate synthetic backtracking into our training paradigm and evaluate it across Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Llama3.2-3B-Instruct. Our results show that the optimal backtrack depth increases with task difficulty, yielding up to a 32.1% accuracy improvement over the base model on Sudoku. Further, we have empirical evidence demonstrating that, when initializing RL training from models that have seen incorrect demonstrations, training can still be stable and effective. Leveraging the characteristics of pretrain and SFT data to continually scale model performance through RL remains an intriguing direction for future works.

Acknowledgments

Junlin Wang is partially supported by through Grant G-23- 2137070 from the Learning Engineering Virtual Institute to the University of Florida and its partner institutions, including Duke.

References

- Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, et al. An empirical study on eliciting and improving r1-like reasoning models. *arXiv preprint arXiv:2503.04548*, 2025.
- Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.
- Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.
- Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.
- Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*, 2025.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

- Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. Rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, 2023.
- Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in rl training of llms, 2025. URL <https://arxiv.org/abs/2410.01679>.
- Joongwon Kim, Anirudh Goyal, Liang Tan, Hannaneh Hajishirzi, Srinivasan Iyer, and Tianlu Wang. Astro: Teaching language models to reason by reflecting and backtracking in-context. *arXiv preprint arXiv:2507.00417*, 2025.
- Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.
- Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. Common 7b language models already possess strong math capabilities. *arXiv preprint arXiv:2403.04706*, 2024.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.
- Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice Hall, Englewood Cliffs, NJ, 1972. URL <http://garfield.library.upenn.edu/classics1980/A1980KD04600001.pdf>.
- OpenAI. Learning to reason with llms. <https://openai.com/index/learning-to-reason-with-llms/>, 2024. Accessed: 2025-03-21.
- Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. <https://github.com/Jiayi-Pan/TinyZero>, 2025. Accessed: 2025-01-24.
- Du Phan, Matthew Douglas Hoffman, David Dohan, Sholto Douglas, Tuan Anh Le, Aaron Parisi, Pavel Sountsov, Charles Sutton, Sharad Vikram, and Rif A Saurous. Training chain-of-thought via latent-variable inference. *Advances in Neural Information Processing Systems*, 36:72819–72841, 2023.
- Tian Qin, David Alvarez-Melis, Samy Jelassi, and Eran Malach. To backtrack or not to backtrack: When sequential search limits model reasoning. *arXiv preprint arXiv:2504.07052*, 2025.
- Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *Advances in Neural Information Processing Systems*, 37:55249–55285, 2024.
- Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL <https://qwen.ai/blog?id=qwen3.5>.

- Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026b. URL <https://qwen.ai/blog?id=qwen3.6-35b-a3b>.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.
- Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, et al. Archon: An architecture search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*, 2024.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. RL on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031, 2024.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.
- Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.24760*, 2025.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.
- Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL <https://qwenlm.github.io/blog/qwen2.5/>.
- Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025. URL <https://qwenlm.github.io/blog/qwq-32b/>.
- Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.
- Junlin Wang, Shang Zhu, Jon Saad-Falcon, Ben Athiwaratkun, Qingyang Wu, Jue Wang, Shuaiwen Leon Song, Ce Zhang, Bhuwan Dhingra, and James Zou. Think deep, think fast: Investigating efficiency of verifier-free inference-time-scaling methods. *arXiv preprint arXiv:2504.14047*, 2025.
- Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*, 2025.
- Xiao-Wen Yang, Xuan-Yi Zhu, Wen-Da Wei, Ding-Chu Zhang, Jie-Jing Shao, Zhi Zhou, Lan-Zhe Guo, and Yu-Feng Li. Step back to leap forward: Self-backtracking for boosting reasoning of language models. *arXiv preprint arXiv:2502.04404*, 2025.
- Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

- Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.
- Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.
- Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.
- Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025.
- Rosie Zhao, Alexandru Meterez, Sham Kakade, Cengiz Pehlevan, Samy Jelassi, and Eran Malach. Echo chamber: RL post-training amplifies behaviors learned in pretraining. *arXiv preprint arXiv:2504.07912*, 2025.
- Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024, 2025.

A Usage of LLM

We utilize LLM to assist some of the paper, specifically polish grammar and sentences. We also utilized LLMs for simple code development, for example writing a initial evaluation code. Lastly, we also use LLM to write the code for figure generation.

B Reasoning Games

Countdown The goal is to construct an arithmetic expression using a set of numbers to reach a specific target value. Each number can be used at most once, and operations are limited to basic arithmetic: addition, subtraction, multiplication, and division.

Sudoku The objective is to fill a 9×9 grid so that each row, column, and 3×3 subgrid contains all the digits from 1 to 9 exactly once.

Arc 1D The objective is to learn and apply a transformation rule that maps a one-dimensional input grid to a corresponding output grid, given several input-output examples.

Advanced Geometry Advanced Geometry contains three sub-tasks. In the angle measurement sub-task, the goal is to compute the internal angle at a specific vertex of a triangle, given the input of three vertex coordinates in the Cartesian plane. In the orthocenter sub-task, the goal is to determine the orthocenter of a triangle—the point where all three altitudes intersect, given the input of the triangle’s three vertex coordinates. In the incircle radius sub-task, the goal is to compute the radius of the incircle of a triangle, given the input of the coordinates of its three vertices.

Color Cube Rotation In the game of Color Cube Rotation, the goal is to determine the color of a specific side of a cube after a sequence of 3D rotations, given the input of its initial face-color configuration.

List Functions The goal is to generate an output list by identifying and applying an implicit rule that maps each input list to its corresponding output, given examples.

Zebra Puzzles The goal is to determine a specific attribute—such as the name of the person in a given house—based on a set of logical constraints, given the input of several people and their distinct characteristics (e.g., favorite drink, pet, or phone).

Self Reference The goal is to determine how many consistent truth assignments exist for a set of self-referential statements, given the input of seven logically interconnected claims. Each statement makes assertions about the truth or falsity of other statements (or the total number of true/false ones).

C Model training details

For RL, we investigate both PPO and GRPO. We use learning rate of $1e - 6$ for both actor and critic, KL control coefficient of 0 (no KL penalty), gradient clipping of 1.0, entropy coefficient of 0.001, value coefficient of 1.0, gae lambda of 1.0, gamma of 1.0. These hyperparameters are held constant across all runs, with the only difference being the response length to accommodate for problems with longer solution trajectories such as Sudoku. All experiments are done on 4 A100s.

D Reward Setup

Countdown We expect solution to not include the equal sign

Advanced Geometry For angle measure task: we expect a solution string rounded to two decimal places with degrees symbols attached. Failure to include degree symbol, or not rounding to two decimal places will be deemed incorrect. For orthocenter task: we expect a solution to give the coordinate either in the form `"(1.000 2.000)"` or `"(1.000, 2.000)"` or `"(1.000,2.000)"`. Failure to include `"()"` or not rounding to 3 decimal places will be deemed incorrect. For incircular radius task: we expect a solution string in the format of `"1.123"`. Failure to round to 3 decimal places will be deemed incorrect.

Arc 1D We expect a solution string in the format of `"1 2 3"`, other formats such as `"1, 2, 3"` will be deemed incorrect.

Sudoku We expect a solution string in the format of `"1 2 3\n4 5 6\n7 8 9"`, other formats such as `"1, 2, 3\n4, 5, 6\n7, 8, 9\n"` will be deemed incorrect and only receive format score. Parsing includes generating a list of lists from the solution string.

Color Cube Rotation The solution parser is case insensitive. We expects a solution string in the format of `"cyan"` or `"Cyan"`. Parsing includes lower case all letter.

Zebra Puzzles We expect a solution string in the format of `"bob"` or `"Bob"`. Parsing includes lower case all letters.

List Functions We expect a solution string either in the format of `"[1 2 3]"` or `"[1, 2, 3]"` or `"[1,2,3]"`. Failure to include `"[]"` will be deemed incorrect. Will parse to a numpy array.

Self Reference We expect a solution in the format of a single digit like `"4"`. Parsing includes converting the solution string to an integer.

E Model and Task Scope Ablations

E.1 Robustness to a newer model generation

The main experiments show a consistent pattern across Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Llama3.2-3B-Instruct: zero backtracks perform best on Arc-1D, one backtrack performs best on Countdown, and substantially deeper backtracking performs best on Sudoku, although the exact Sudoku depth varies by model. To test whether the Countdown result also holds in a newer model generation, we repeat the same backtracking experiment in Section 4 with Qwen3.5-2B (Qwen Team, 2026a).

As shown in Table 3, the one-backtrack warm-up achieves the highest pass@1 among the tested depths: 47.6%, compared with 35.2% for zero backtracks, 32.1% for two backtracks, and 29.3% for three backtracks.

Table 3: Countdown pass@1 accuracy after synthetic-backtracking SFT warm-up and RL post-training with Qwen3.5-2B.

Model	0 Backtrack	1 Backtrack	2 Backtrack	3 BT
Qwen3.5-2B	35.2	47.6	32.1	29.3

E.2 Preliminary generalization to MATH

Our synthetic backtracking datasets for Countdown and Sudoku are generated from symbolic solvers, while Arc-1D uses a heuristic search. Constructing an exhaustive search tree is considerably harder for open-ended domains such as math reasoning or coding. However, our method emphasizes on the metacognitive behavior that backtracking captures:

recognizing that the current reasoning state is flawed, returning to a prior valid state, and continuing from there.

To test whether the backtracking-centric recipe generalizes beyond puzzle-style tasks, we conduct a preliminary experiment on MATH (Hendrycks et al., 2021). We use the provided step-by-step gold solutions as zero-backtrack trajectories. To construct the one-backtrack trajectories, we prompt a stronger Qwen3.6-35B-A3B (Qwen Team, 2026b) teacher model to rewrite them into one-backtrack trajectories by injecting an intermediate error, explicit recognition, and recovery to the valid path. Therefore, we constructed our proxy search tree trajectories via distillation from the teacher model. After post-training, one-backtrack warm-up improves pass@1 over the zero-backtrack warm-up from 20.78% to 25.74%, providing initial evidence that backtracking-style warm-up can extend to broader tasks where solver based search trees are not available.

Table 4: MATH pass@1 accuracy after post-training Qwen2.5-0.5B-Instruct with zero- and one-backtrack warm-ups.

Task / Model	0 Backtrack	1 Backtrack
MATH / Qwen2.5-0.5B-Instruct	20.78	25.74

F Incorrect vs. Correct Warmup Comparison

Table 5: Countdown accuracy after RL with correct and incorrect self-sampled SFT warm-ups for Qwen2.5-7B-Instruct.

Model	Task	SFT trajectory	Pass@1 (%)
Qwen2.5-7B-Instruct	Countdown	Correct	37.5
Qwen2.5-7B-Instruct	Countdown	Incorrect	40.5

G Example Model Outputs

G.1 List Functions Pure RL model outputs

Pure RL Outputs: List Functions

```
<|im_start|>assistant
Let me solve this step by step.
<think> The rule in the examples seems to be reversing the order
of the elements in the list. Let's apply this rule to the input
list [74, 72, 49, 43, 41, 3, 30, 85, 48]. Reversing [74, 72, 49,
43, 41, 3, 30, 85, 48] gives us [48, 85, 30, 3, 41, 43, 49, 72,
74]. </think>
<answer> [48, 85, 30, 3, 41, 43, 49, 72, 74] </answer><|im_end|>
```

G.2 Countdown Pure RL model outputs

Pure RL Outputs: Countdown

```
<|im_start|>assistant
Let me solve this step by step.
<think> First, I need to create an expression that equals 194
using the numbers 14, 17, 2, 45, and 8. Let's try some
combinations. If I add 45 and 17, I get 62. Then if I add 14 to
62, I get 76. Adding 8 to 76 gives me 84. If I add 2 to 84, I
get 86. I need to reach 194, so I need to try different
combinations. Let's try  $45 * 2 = 90$ , then  $90 + 17 = 107$ , then
 $107 + 14 = 121$ , then  $121 + 8 = 129$ . This doesn't work. I need
to find a better combination. Let's try  $17 * 8 = 136$ , then  $136
+ 45 = 181$ , then  $181 + 14 = 195$ , then  $195 - 2 = 193$ . Still not
194. I need to find a different approach. Let's try  $2 * 14 =
28$ , then  $28 + 45 = 73$ , then  $73 + 17 = 90$ , then  $90 + 8 = 98$ .
Still not 194. I need to find a different combination. Let's
try  $45 * 2 = 90$ , then  $90 + 14 = 104$ , then  $104 + 17 = 121$ , then
 $121 + 8 = 129$ . Still not 194. I need to find a different
approach. Let's try  $17 * 14 = 238$ , then  $238 - 45 = 193$ ,
then  $193 + 8 = 201$ , then  $201 - 2 = 199$ . Still not 194. I need
to find a different combination. Let's try  $14 * 2 = 28$ , then
 $28 + 45 = 73$ , then  $73 + 17 = 90$ , then  $90 + 8 = 98$ . Still not
194. I need to find a different approach. </think>
```

G.3 Countdown RL model from shuffled synthetic data output

Shuffled RL Outputs: Countdown

```
<|im_start|>assistant
Let me solve this step by step.
<think>
Step 1:  $51 - 23 = 28$ . Step 2:  $28 * 36 = 1008$ . Step 3:  $1008 -
57 = 951$ . Step 4:  $951 - 48 = 885$ . Wait, this doesn't lead to
the correct solution. 885 is not the correct answer. Let me
go back to step 2 and keep thinking from there. Step 3:  $1008
- 27 = 981$ . Step 4:  $981 - 48 = 933$ . This matches the problem
statement. This is the solution.
</think>

<answer> $51 - 36 * 36 - 57 - 48 - 27$ </answer><|im_end|>
```

H Example Backtracking Trajectories

Countdown

```
Step 1:  $47 - 27 = 20$ . Step 2:  $20 * 10 = 200$ . Step 3:  $200 - 47 = 153$ . Wait, this
doesn't lead to the correct solution. 153 is not the correct answer. Let me
go back to step 2 and keep thinking from there. Step 3:  $200 - 76 = 124$ .
This matches the problem statement. This is the solution.
```

```
<answer> $10 * (47 - 27) - 76$ </answer>
```

Truncated Example of Sudoku

Looking at the cell at (1,1). Based on the current board, row 1 is missing [3, 4, 6, 7, 8], column 1 is missing [1, 3, 4, 5, 6, 7, 9], and the 3x3 box is [1, 3, 4, 6, 7]. Taking the intersection of these sets, the possible values for cell (1,1) are [3, 4, 6, 7]. Let's try to fill cell (1,1) with 3.

Looking at the cell at (1,2). Based on the current board, row 1 is missing [4, 6, 7, 8], column 2 is missing [2, 3, 4, 6, 7, 8, 9], and the 3x3 box is [1, 4, 6, 7]. Taking the intersection of these sets, the possible values for cell (1,2) are [4, 6, 7]. Let's try to fill cell (1,2) with 4.

Looking at the cell at (1,3). Based on the current board, row 1 is missing [6, 7, 8], column 3 is missing [5, 6], and the 3x3 box is [1, 6, 7]. Taking the intersection of these sets, the possible values for cell (1,3) are [6]. Let's try to fill cell (1,3) with 6.

Looking at the cell at (1,6). Based on the current board, row 1 is missing [7, 8], column 6 is missing [1, 5, 7, 8], and the 3x3 box is [3, 5, 7, 8]. Taking the intersection of these sets, the possible values for cell (1,6) are [7, 8]. Let's try to fill cell (1,6) with 7.

Looking at the cell at (1,8). Based on the current board, row 1 is missing [8], column 8 is missing [1, 3, 4, 5, 7, 8], and the 3x3 box is [1, 2, 3, 4, 6, 7, 8]. Taking the intersection of these sets, the possible values for cell (1,8) are [8]. Let's try to fill cell (1,8) with 8.

Looking at the cell at (2,4). Based on the current board, row 2 is missing [1, 3, 4, 7], column 4 is missing [3, 4, 5, 6, 8, 9], and the 3x3 box is [3, 5, 8]. Taking the intersection of these sets, the possible values for cell (2,4) are [3]. Let's try to fill cell (2,4) with 3.

Looking at the cell at (2,7). Based on the current board, row 2 is missing [1, 4, 7], column 7 is missing [1, 2, 6, 8], and the 3x3 box is [1, 2, 3, 4, 6, 7]. Taking the intersection of these sets, the possible values for cell (2,7) are [1]. Let's try to fill cell (2,7) with 1.

Looking at the cell at (2,8). Based on the current board, row 2 is missing [4, 7], column 8 is missing [1, 3, 4, 5, 7], and the 3x3 box is [2, 3, 4, 6, 7]. Taking the intersection of these sets, the possible values for cell (2,8) are [4, 7]. Let's try to fill cell (2,8) with 4.

Looking at the cell at (2,9). Based on the current board, row 2 is missing [7], column 9 is missing [1, 3, 4, 6, 7, 8], and the 3x3 box is [2, 3, 6, 7]. Taking the intersection of these sets, the possible values for cell (2,9) are [7]. Let's try to fill cell (2,9) with 7.

Looking at the cell at (3,1). Based on the current board, row 3 is missing [1, 2, 3, 5, 6, 7, 8], column 1 is missing [1, 4, 5, 6, 7, 9], and the 3x3 box is [1, 7]. Taking the intersection of these sets, the possible values for cell (3,1) are [1, 7]. Let's try to fill cell (3,1) with 1. The current board is:

```

3 4 6 1 2 7 5 8 9
2 5 8 3 6 9 1 4 7
1 _ 9 _ _ 4 _ _ _
_ _ _ 7 3 _ 9 _ _
_ _ 3 2 8 _ 7 _ _
_ _ 1 _ _ _ 2 _ _
_ 1 2 _ _ 6 9 _ _
8 _ 4 7 9 2 3 _ 5
_ _ 7 _ _ _ 4 6 2

```

Arc-1d

Looking at these three examples. Let's identify all the blocks of sequences of consecutive 2s separated by gaps. Then let's try to recolor the left half of each block to 8, and try to keep the right half unchanged.

Applying this rule to the test input, we get the output grid:

Example 1: 0 0 0 0 0 0 0 3 0 3 3 0

Example 2: 2 0 2 0 2 0 8 8 2 2 2 0

Example 3: 2 0 2 0 8 8 8 8 2 2 2 2

The outputs do not match the given example outputs.

Let's try to find another pattern.

Let's first identify the main block | a group of consecutive cells with the same color. It looks like the block is surrounded by noise: individual pixels of the same color placed away from the block. Let's try remove all the pixels by setting them to 0. Applying this rule to the test input, we get the output grid:

Example 1: 0 0 0 0 0 0 0 0 3 3 0

Example 2: 0 0 0 0 0 0 2 2 2 2 0

Example 3: 0 0 0 0 2 2 2 2 2 2 2

The outputs match the given example output. Let's apply this rule to the test input and obtain the answer.

<answer>0 8 8 8 8 0 0 0 0 0 0 0</answer>

I Qwen2.5-3B-Instruct Training Token Counts

Table 6: Token counts for different datasets and training methods on Countdown.

Dataset / Method	# Output tokens
Optimal synthetic dataset	108.81
1 backtrack dataset	796.16
2 backtrack dataset	883.26
3 backtrack dataset	745.31
Correct cold start RL	625.03
Incorrect cold start RL	221.37
Pure RL	442.40

Table 7: Token counts for different datasets and training methods on Arc 1D.

Dataset / Method	# Output tokens
Optimal synthetic dataset	270.43
1 backtrack dataset	473.64
2 backtrack dataset	664.81
3 backtrack dataset	889.47
Correct cold start RL	127.70
Incorrect cold start RL	352.76
Pure RL	118.80

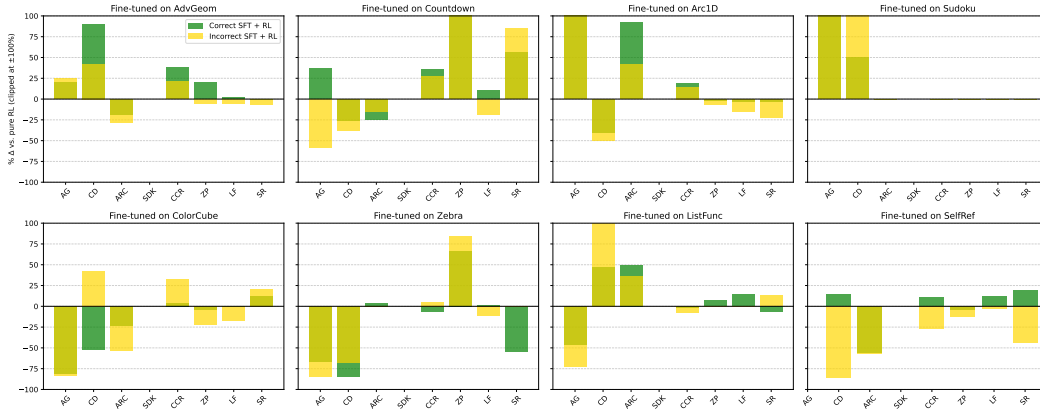
Table 8: Token counts for different datasets and training methods on Sudoku.

Dataset / Method	# Output tokens
Optimal synthetic dataset	4702.90
1 backtrack dataset	7028.17
5 backtrack dataset	7690.53
10 backtrack dataset	7607.92
Correct cold start RL	
Incorrect cold start RL	
Pure RL	449.55

J Self Sampled CoTs model evaluations

Table 9: Evaluation accuracy (in percentage) of the Qwen2.5-Instruct and QwQ-32B baselines and the pure RL’ed Qwen2.5-3B models. The row names are the model names (or the task on which they were RL’ed), and the column names represent the evaluation tasks.

Model	AG	CD	ARC	SDK	CCR	ZP	LF	SR
Qwen2.5-3B-Instruct	1.5	0.4	1.8	0.0	28.6	25.4	19.9	13.4
Qwen2.5-7B-Instruct	5.2	1.9	6.4	0.0	28.1	38.8	31.4	13.8
QwQ-32B	34.4	51.5	24.0	0.0	13.5	—	74.8	—
AdvGeom	30.9	1.9	2.1	0.0	24.4	31.9	24.1	13.1
Countdown	4.3	47.9	3.3	0.0	22.1	14.3	21.4	8.6
Arc1D	0.1	1.0	23.4	0.0	24.1	31.9	29.9	15.6
Sudoku	1.2	0.2	2.5	0.0	25.9	26.9	21.7	14.8
ColorCube	14.9	1.9	3.0	0.0	62.9	33.1	24.7	10.5
Zebra	7.6	1.9	2.4	0.0	34.6	26.5	24.0	11.2
ListFunc	12.0	1.5	2.8	0.0	34.0	30.8	71.2	12.5
SelfRef	0.0	0.7	5.4	0.0	26.8	35.2	18.5	64.8

Figure 5: Correct and incorrect short self-sampled CoTs RL model evaluation. These plots measure the performance delta (Δ) against a pure RL baseline. The results demonstrate that while some tasks (e.g., Countdown) promote broad reasoning generalization across the suite, others (e.g., ColorCube) exhibit narrow task-specific improvements or negative transfer to unrelated domains.

K Response Length Comparison

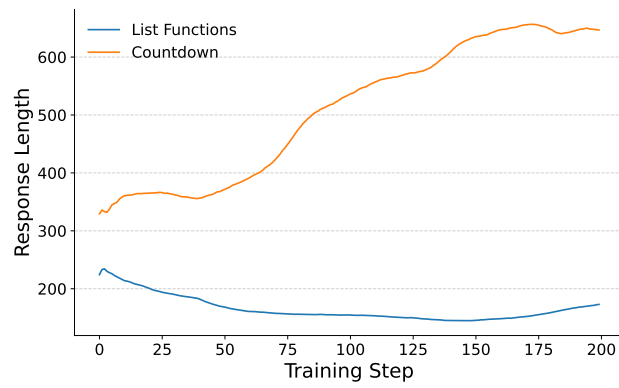


Figure 6: Response length comparison between List Functions and Countdown. Divergent Length Scaling in Countdown vs. List Functions. The chart shows that as training progresses, the model develops longer, more elaborate responses for Countdown tasks, whereas it becomes more brief and direct when performing List Functions.