

Extended Abstract

Motivation Even after RL fine-tuning, language models reason through unstructured natural-language chains of thought whose reliability remains limited for multi-step mathematical tasks. On the Countdown task, where a model must compose arithmetic operations over given numbers to reach a target, models frequently make arithmetic errors, fail to commit to correct solutions, and exhaust their generation budget without answering. We explore two approaches, Structured Backtracking (BT) and Targeted Failure Correction (FC), to make the mathematical reasoning more robust, sharing the same motivation but differing in strategy.

Method Prior work on backtracking focuses on unverifiable textual recovery and relies on external search tools. Our first approach, Structured BT, reconstructs the reasoning process top-down by introducing explicit `<state_K>` and `<backtrack_K>` tokens that checkpoint intermediate states and signal branch abandonment with programmatic state restoration. To train this behavior, we develop a deterministic Countdown solver that labels all reachable states and curate synthetic backtracking traces for supervised fine-tuning. The model is then further trained using RLOO with a structural process reward model (PRM) for dense feedback on search quality. While the model learns to exhibit backtracking, it underperforms the default baseline because the grammar’s bookkeeping competes with arithmetic reasoning for the scarce capacity of a 0.5B model. This failure motivates our second approach: Targeted FC, which addresses the same underlying reliability problem bottom-up. Analyzing BT’s underperformance identifies three concrete bottlenecks—arithmetic mistakes, inconsistent commitment to correct solutions, and rambling without answering—and FC targets each sequentially through (1) an arithmetic PRM that penalizes incorrect calculations during RLOO, (2) rejection-sampling fine-tuning (RFT) on diverse correct rollouts to consolidate single-attempt accuracy, and (3) IPO preference optimization against rambling.

Implementation All experiments use Qwen 2.5 0.5B Base on the Countdown task. Structured BT generates 1,200 synthetic traces via the deterministic solver narrated by GPT-4.1-mini, registers 10 special tokens, and trains with RLOO using a rule-based structural PRM. Targeted FC warm-starts each stage from the previous checkpoint: arithmetic PRM RLOO, RFT on 5,849 distinct correct rollouts, and IPO on 1,131 preference pairs contrasting correct solutions against rambling failures.

Results Structured BT underperforms the default RLOO baseline (pass@1 0.32 vs. 0.56), with NO-ANSWER rates roughly doubling to 54%. Targeted FC improves pass@1 from 0.56 to 0.62 (+11%), reduces the consistency gap from 0.20 to 0.12, and suppresses NO-ANSWER below the baseline (25.4% vs. 25.9%). Each stage successfully addresses its targeted failure mode while preserving gains from preceding stages.

Discussion A small model’s limited capacity constrains what post-training can achieve: structured BT’s grammar consumes capacity needed for reasoning itself, while targeted FC succeeds because each lightweight stage adds minimal overhead. Notably, correcting arithmetic mistakes is largely decoupled from final-answer correctness—the accuracy gains come from consolidating correct solutions (RFT) and suppressing rambling (IPO), not from cleaner arithmetic. A persistent failure mode remains where the model misses near-solutions and never revisits them, which is precisely the kind of failure that successful backtracking could address, suggesting the two approaches are complementary rather than competing.

Conclusion Under the capacity constraints of a small model, lightweight sequential interventions targeting specific failure modes compose more effectively than monolithic structural changes. Our negative result on BT provides a cautionary lesson that structural search grammars may exceed a small model’s capacity budget, while targeted FC demonstrates that the same goal of robust reasoning can be approached incrementally. Future work includes applying structured BT at larger model scales and replacing randomly generated dead-end branches with strategically chosen ones that better reflect natural reasoning failures.

From Structured Backtracking to Targeted Failure Correction for Robust Mathematical Reasoning

Eric Xia

Department of Computer Science
Stanford University
ericxia@stanford.edu

Christine Li

Department of Electrical Engineering
Stanford University
cli27@stanford.edu

Abstract

Even after RL fine-tuning, language models reason through unstructured natural-language chains of thought with limited reliability for multi-step mathematical tasks. We explore two approaches on the Countdown task with a 0.5B model. Our first approach, Structured Backtracking (BT), introduces state-checkpoint and backtrack tokens for explicit search, trained via supervised fine-tuning on solver-curated traces (SFT) and REINFORCE Leave-One-Out (RLOO) with a dense rule-based process reward. While the model learns to backtrack, it underperforms the default baseline because the grammar’s bookkeeping consumes capacity the model needs for reasoning. This motivates our second approach, Targeted Failure Correction (FC), a lightweight three-stage pipeline that sequentially addresses concrete bottlenecks identified from BT’s failure analysis: (1) RLOO with a structural process reward penalizing arithmetic mistakes, (2) rejection-sampling fine-tuning on diverse correct rollouts to consolidate single-attempt accuracy, and (3) preference optimization against rambling. FC achieves pass@1 of 0.62 (+11%), cuts the consistency gap from 0.20 to 0.12, and reduces arithmetic error rates threefold. Our results suggest that under the capacity constraints of a small model, lightweight sequential interventions targeting specific failure modes compose more effectively than monolithic structural changes.

1 Introduction

Fine-tuning language models with reinforcement learning has driven substantial progress on reasoning tasks [1, 8, 12]. Yet mathematical reasoning remains distinctly challenging: it requires multi-step planning over a combinatorial search space and correct arithmetic execution at every intermediate step [3, 7]. Even after RL fine-tuning, models still reason through unstructured natural-language chains of thought whose reliability remains limited. The central question of this work is how to make this reasoning process more robust, and we explore two approaches that share this motivation but differ in strategy. We study both approaches on Countdown [9, 13], where a model must compose arithmetic operations (OP) over given numbers to reach a target.

Standard RL-trained models express backtracking implicitly through natural language, but this provides no mechanism for verifiable state restoration [5, 15]. Our first approach, Structured Backtracking (BT), introduces explicit `<state_K>` and `<backtrack_K>` tokens that checkpoint intermediate states and signal branch abandonment. We curate a dataset of synthetic backtracking traces via a deterministic solver and train the model on these traces via supervised fine-tuning (SFT), followed by REINFORCE Leave-One-Out (RLOO) with a process reward model (PRM) that provides dense feedback on search quality [7, 14]. However, BT ultimately underperforms the default baseline—the grammar’s bookkeeping competes with arithmetic reasoning for the scarce capacity of a 0.5B model.

Thus, we pivot to a lightweight alternative. Analyzing BT’s failure sharpens the vague notion of “unreliable reasoning” into three concrete bottlenecks: arithmetic mistakes within the reasoning trace, inconsistent commitment to correct solutions across attempts, and rambling without producing an answer. This motivates Targeted Failure Correction (FC), a three-stage pipeline that addresses each bottleneck sequentially through an arithmetic PRM during RLOO, rejection-sampling fine-tuning (RFT) [16] on diverse correct rollouts to consolidate single-attempt accuracy, and IPO preference optimization against rambling. Each stage warm-starts from the previous checkpoint, preserving prior gains. FC achieves pass@1 of 0.62 (+11% over the default baseline), and we discover that arithmetic mistakes in the trace are largely decoupled from final-answer correctness.

Our contributions are:

- A synthetic data-generation pipeline that produces structured backtracking traces via a deterministic solver for supervised warm-starting.
- Structured BT, an explicit backtracking framework with a programmatic structural PRM, and a negative result demonstrating capacity limitations at small model scale.
- Identification of three concrete reasoning bottlenecks from structured BT’s failure analysis as actionable targets for post-training intervention.
- Targeted FC, a lightweight three-stage pipeline that sequentially addresses each bottleneck, achieving pass@1 of 0.62 (+11%).

2 Related Work

RL fine-tuning for reasoning. Modern post-training pipelines for language models typically begin with supervised fine-tuning (SFT) on curated demonstrations, followed by preference optimization or online RL. Ouyang et al. [8] established the preference-based paradigm with RLHF, and subsequent work has introduced more efficient variants: Rafailov et al. [11] reformulate the constrained RL problem as a supervised classification objective over preference pairs (DPO), and Azar et al. [2] relax the Bradley–Terry assumption to reduce overfitting (IPO). For online policy-gradient methods, Ahmadian et al. [1] propose REINFORCE Leave-One-Out (RLOO), which uses a leave-one-out baseline to reduce variance without a learned value function. These methods have been applied to mathematical reasoning with notable success. Our work primarily builds on this SFT, IPO, and RLOO foundation, extending them with new datasets, loss functions, and integration strategies for our proposed structured BT and targeted FC methods.

Structured search for backtracking. Prior work has explored structured search for backtracking to improve mathematical reasoning in language models. Yao et al. [15] represent reasoning as an explicit tree (Tree of Thoughts) and use external BFS/DFS controllers to explore branches at inference time. Gandhi et al. [5] train models to express search and backtracking through structured text traces (Stream of Search) but rely on unstructured natural language to restore prior states. El et al. [4] introduce a `<clean>` token that removes failed reasoning from context for Countdown, but restart reasoning from scratch rather than restoring a previous valid state. Overall, these methods show that structured search can improve reasoning, but they either depend on external controllers, rely on unverifiable textual backtracking, or remove context destructively. Our BT approach differs by introducing explicit, verifiable state-checkpoint tokens with programmatic context restoration.

Process reward models. Outcome-only rewards provide sparse training signal for multi-step reasoning. Lightman et al. [7] provide step-level correctness labels during training, showing that intermediate feedback can be more effective than outcome-only rewards. Wang et al. [14] reduce annotation cost by using Monte Carlo rollouts to estimate step-level correctness (Math-Shepherd). In contrast, both BT and FC use programmatic PRMs that detect improper backtracking events or incorrect arithmetic statements, requiring no human annotation or rollout-based estimation.

Self-improvement for mathematical reasoning. Zelikman et al. [17] bootstrap reasoning by fine-tuning on the model’s own correct solutions with rationalization (STaR). Yuan et al. [16] scale this idea with RFT by sampling many diverse rollouts per prompt and keeping only the correct ones for supervised training. This allows the model to learn from its own distribution, reinforcing its natural reasoning strengths. Additionally, since mathematical reasoning problems often admit multiple valid

solutions, RFT teaches the model that a single problem can be approached from different angles, making its reasoning more flexible. This inspires targeted FC’s second stage, which adapts the original RFT recipe to target consistent commitment to correct answers across multiple attempts.

3 Default Method

The default project pipeline comprises three training stages, SFT, IPO, and RLOO, which our subsequent structured BT and targeted FC methods build upon and adapt.

SFT. Supervised fine-tuning (SFT) uses the standard next-token prediction objective, with the loss applied only to expert completion tokens y_t given prompt x and preceding tokens $y_{<t}$:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{x,y \in \mathcal{D}} \sum_{t=1}^{|y|} \log \pi_{\theta}(y_t \mid x, y_{<t}). \quad (1)$$

IPO. Identity Preference Optimization (IPO) [2] learns from pairwise preferences over preferred (y_w) and rejected (y_l) responses. It encourages the policy to assign higher relative likelihood to preferred responses while controlling deviation from a reference policy π_{ref} :

$$\mathcal{L}_{\text{IPO}} = \left\| h_{\pi_{\theta}}^{y_w, y_l} - \frac{1}{2\beta} \right\|_2^2, \quad h_{\pi_{\theta}}^{y_w, y_l} = \log \frac{\pi_{\theta}(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \log \frac{\pi_{\theta}(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}, \quad (2)$$

where β controls the strength of the regularization.

RLOO. REINFORCE Leave-One-Out (RLOO) [1] is an online policy-gradient method that samples multiple completions for a given prompt. It uses the average reward of the other rollouts for the same prompt as a baseline to compute the advantage for sample i :

$$A_i = R(y_{(i)}, x) - \frac{1}{k-1} \sum_{j \neq i} R(y_{(j)}, x). \quad (3)$$

The policy gradient is then:

$$\frac{1}{k} \sum_{i=1}^k A_i \nabla \log \pi_{\theta}(y_{(i)} \mid x). \quad (4)$$

Since trajectories are sampled efficiently using vLLM, a separate inference engine treated as the behavior policy μ , the advantage is multiplied by a sequence-level importance weight, $w(y, x) = \exp(\log \pi_{\theta}(y \mid x) - \log \mu(y \mid x))$, to correct for distributional drift. The overall loss additionally includes entropy regularization and a KL penalty that constrains the policy from diverging too far from the reference model.

4 Structured Backtracking (BT)

4.1 Method

Structured BT aims to teach the language model to issue special tokens for explicit, verifiable backtracking. The full pipeline is illustrated in Figure 1. BT firstly trains the model on synthetic structured search traces via SFT, then apply RLOO on the warm-start SFT model with a context-restoration manager and a structural PRM.

Special tokens. We introduce two families of special tokens. `<state_K>` checkpoints the reasoning state after K arithmetic operations, recording the currently available values as expressions over the original inputs (e.g., `<state_1>` [(36+5)=41, 78]). This preserves each intermediate derivation and supports later verification of search-state consistency during RL. `<backtrack_K>` signals abandonment of the current branch and restoration to the most recent depth- K checkpoint. We set $K_{\text{max}} = 4$, yielding 10 special tokens in total, which suffices for Countdown problems with 3–4 input numbers.

Countdown solver. Given input numbers and target, the countdown solver exhaustively enumerates reachable states under binary arithmetic operations, labels each intermediate state as solvable or unsolvable, extracts verified solution paths, and identifies dead-end branches.

Synthetic data generation. Given the prompts from `cog_behav_all_strategies`, we curate new training data of reponse with structured backtracking traces through two steps. First, a `search trace builder` combines solver-verified solution paths with injected dead-end branches that require backtracking. We created different search patterns from simple linear solutions to multiple backtracks at different depths. Second, since the raw search traces contain only structured tokens and arithmetic facts, we use GPT-4.1-mini to narrates each trace into natural-language reasoning while preserving the special tokens and arithmetic facts. To ensure perfect training data, each narration is validated by strict checks, including tag order, arithmetic correctness, state-list accuracy, final answer correctness, and formatting. The validated narration serves as the new response for SFT.

Context-restoration manager. During RLOO sampling, we augment the generation stop strings with `<backtrack_K>` tokens, so generation halts whenever the model emits a `<backtrack_K>` token. The context manager truncates the text back to the most recent depth- K ancestor state, removes the failed branch, appends a backtrack marker, and resumes generation from this cleaned prefix. This repeats until the model produces a final answer or exhausts a cap of 4 rounds. The goal is to improve rollout quality during RLOO training: if a failed branch remains in context, it can mislead subsequent generation through attention over irrelevant tokens. By removing it, each continuation starts from a clean state, producing higher-quality rollouts and therefore better policy-gradient signal.

Structural process reward model. The structural PRM provides dense feedback on correct use of the backtracking grammar, complementing the sparse outcome reward. It scores each trace based on valid state references, depth consistency with the number of applied operations, correct forward-transition format at each state, correct restoration after backtracking, preservation of the given operands, and arithmetic correctness. The PRM score is computed as the average of these checks and lies in $[0, 1]$. It is then combined with the original outcome reward $R(y_{(i)}, x)$ in Equation (3) to form the augmented RLOO reward:

$$R'(y_{(i)}, x) = R(y_{(i)}, x) + \alpha \cdot \text{PRM}(y_{(i)}). \tag{5}$$

Here, α controls the strength of the structural process reward relative to the outcome reward; we set $\alpha = 0.5$ in our experiments.

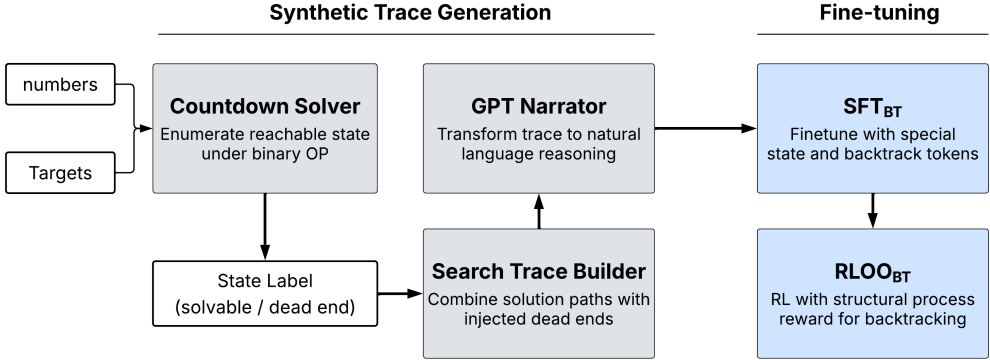


Figure 1: Structured BT pipeline. A Countdown solver enumerates search-tree traces with backtracks; GPT-4.1-mini narrates each skeleton into natural-language traces with embedded special tokens; the model is fine-tuned on these structured traces via SFT, then trained with RLOO augmented by a structural PRM and context-restoration manager.

4.2 Experimental Setup

Structured BT experiments explore explicit backtracking on the Countdown task. We first generate 1,200 synthetic structured traces (1,000 train / 200 test) from prompts in

cog_behav_all_strategies. We then fine-tune Qwen 2.5 0.5B [10] with SFT on this modified dataset for 6 epochs, using a learning rate of $5e-5$, batch size 64, gradient accumulation over 8 steps, gradient clipping at 1, weight decay of 0.01, and a cosine learning-rate scheduler with 5% warmup. Starting from the structured BT SFT checkpoint, we further apply RLOO with the updated reward function (5) for 100 steps on countdown_tasks_3to4, using a learning rate of $1e-5$, batch size 128, gradient accumulation over 128 steps, group size 8, KL coefficient 0.001, entropy coefficient 0.001, weight decay of $1e-4$, and a constant learning-rate schedule. All experiments are run on a single H100 GPU via Modal, with vLLM [6] used for sampling.

We compare structured BT against the default SFT and RLOO baselines at the corresponding training stages. Evaluation is conducted on 50 prompts from the test split of countdown_tasks_3to4, with 16 rollouts per prompt. All generations use temperature 0.6, top- p 0.95, top- k 20, and a maximum length of 1024 tokens. We report pass@1, pass@16, and the NO-ANSWER rate, defined as the fraction of rollouts that exhaust the token budget without producing a final answer.

4.3 Results

4.3.1 Structured BT Performance

Table 1: Structured BT vs. default pipeline at the SFT and RLOO stages (50 test prompts, 16 rollouts each).

	Pass@1	Pass@16	NO-ANSWER Rate %
Default SFT	0.34	0.76	9.9
SFT _{BT}	0.20	0.62	28.9
Default RLOO	0.56	0.76	25.9
RLOO _{BT}	0.32	0.56	54.0

According to Table 1, structured BT underperforms the default pipeline at both stages, reaching pass@1 0.20 after SFT against the default’s 0.34 and 0.32 after RLOO against the default’s 0.56. Besides, both structured BT SFT and structured BT RLOO achieves a much higher pass probability under multiple attempts than under a single attempt (+0.42 and +0.24). Such huge gap between pass@1 and pass@16 indicates that the policy has large instability to commit to a correct solution. Additionally, the NO-ANSWER rate after BT SFT is more than three-fold than the default SFT and BT RLOO is more than double. Such extremely higher NO-ANSWER exposes one underlying failure that BT drives the model to loop on backtracks and refuse to commit.

4.3.2 Failure Analysis

```

<state_0> [35, 61, 77]
... 61 - 35 = 26.
<state_1> [(61 - 35)=26, 77]
... 77 - (61 - 35) = 51.
<state_3> [(77 - (61 - 35))=51]    <- should be <state_2>
... a dead end.
<state_2>
<state_2> [(61 - 35)=26, 77]    <- bare token, no list
<state_2> [(61 - 35)=26, 77]    <- index does not match
depth
[... the model keeps emitting state tokens with jumping
      indices and missing lists, never converging ...]
<state_2> [(((52 + (26 + ...))=52)=19]    <- incorrect
nesting
</think>
<answer> ((52 + (26 + (77 - (61 - 35)))))) </answer>

```

Figure 2: Structural collapse in a structured BT RLOO rollout on $[35, 77, 61] \rightarrow 19$. The model mislabels <state_3> where <state_2> is correct, emits bare state tokens without value lists, and produces a malformed final expression evaluating to 52 instead of 19—imitating the grammar’s appearance without maintaining its semantics.

While the model exhibits backtracking behavior, the deeper issue is that a 0.5B model cannot reliably follow the state and backtrack structure. As shown in Figure 2, many trajectories appear well-formed

at a glance but contain incorrect token numberings—a `<state_K>` whose index does not match the number of operations applied. The model imitates the surface pattern of the grammar without tracking the search depth it encodes. At this scale, the limited capacity is largely consumed by structural bookkeeping rather than the arithmetic reasoning needed to solve the problem. The synthetic training data compounds the issue: the dead-end branches that trigger backtracking are randomly injected rather than strategically chosen, so the traces backtrack arbitrarily instead of reflecting the kind of reasoning errors a model would naturally make. The model therefore learns to imitate an arbitrary search structure rather than thoughtful problem-solving, and under RLOO it settles into backtracking indefinitely without committing, which is why the NO-ANSWER rate roughly doubles. This indicates that a small model’s capacity is better spent on high-quality, naturally reasoned traces than on a rigid synthetic search grammar.

5 Targeted Failure Correction (FC)

5.1 Method

BT’s failure shows that restructuring the entire reasoning process exceeds a 0.5B model’s capacity budget. We therefore pursue a lightweight alternative: Targeted Failure Correction (FC), which improves reasoning robustness bottom-up by addressing specific failure modes one at a time. Analyzing BT’s underperformance identifies three potential bottlenecks: arithmetic mistakes within the trace, inconsistent commitment to correct solutions (wide gap between pass@1 and pass@16), and rambling. As illustrated in Figure 3, FC applies an arithmetic process reward, rejection-sampling fine-tuning, and preference optimization against rambling in sequence, with each stage warm-starting from the previous checkpoint. Crucially, each stage trains on the model’s own natural-language rollouts from the preceding stage, keeping capacity overhead minimal while preserving prior behavioral gains.

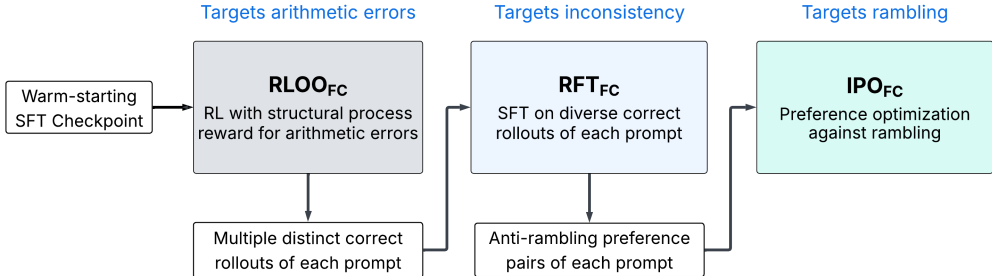


Figure 3: Targeted FC pipeline. Each stage warm-starts from the previous checkpoint and targets one failure mode: arithmetic PRM reduces calculation errors, RFT consolidates single-attempt accuracy, and anti-ramble IPO suppresses non-committal behavior.

5.1.1 Arithmetic Process Reward

To reduce arithmetic errors and their repetition during reasoning, we introduce a process reward that penalizes them directly. We scan each rollout for binary arithmetic statements of the form $a \text{ OP } b = c$, recompute the result, and count two quantities:

- n_{total} — the number of statements whose declared result is incorrect.
- n_{rep} — the number of repeated occurrences of the same incorrect statement.

The penalty combines both counts with a cap to bound its magnitude:

$$\text{penalty} = \rho n_{\text{total}} + \gamma n_{\text{rep}}, \quad \text{PRM} = -\min(\text{penalty}, \text{cap}), \quad (6)$$

with $\rho = 0.1$, $\gamma = 0.2$, and $\text{cap} = 0.5$. A single mistake incurs a penalty of -0.1 ; repeated mistakes are penalized more heavily through γ ; and the cap prevents a single noisy rollout from dominating the gradient. This reward is a single sequence-level scalar added to the default RLOO outcome reward in Equation (3), following the same augmented reward formula as Equation (5) but with the arithmetic PRM in place of the structural PRM.

5.1.2 Rejection-Sampling Fine-Tuning

The policy can solve many problems given multiple attempts but fails to commit reliably on the first try. Rejection-sampling fine-tuning (RFT) [16] addresses this by retraining the policy on its own successful rollouts, shifting probability mass toward solutions that already work.

For each training prompt x , we sample K rollouts from the current policy, retain those scored correct by the task verifier, and select a subset as supervised targets:

$$\mathcal{D}_{\text{RFT}} = \{(x, y) : y \in \text{Select}_N(\{y^{(k)} \sim \pi(\cdot | x) : \text{score}(y^{(k)}) = 1\})\}. \quad (7)$$

We sample at high temperature ($T = 1.0$) and take Select_N to be the top- N *distinct* correct rollouts chosen at random with no length preference ($N = 5$), preserving multiple genuinely different reasoning paths per prompt. The policy is then fine-tuned on \mathcal{D}_{RFT} with the standard SFT objective.

Our procedure follows the original RFT recipe with three differences: we sample from the RL-trained policy rather than the SFT model, determine distinctness by exact text rather than by intermediate equations, and train on the filtered rollouts alone without the original SFT data.

5.1.3 Preference Optimization Against Rambling

Neither previous stage addresses rambling—long reasoning traces that never commit to an answer. Because RFT only exposes the policy to correct examples, it teaches what to do but not what to avoid. Suppressing rambling therefore requires a contrastive signal. We construct preference pairs contrasting correct solutions against long, non-committal failures and train on them using IPO.

We sample 16 rollouts per training prompt at temperature 1.0 from the RFT model itself, which serves as both the warm-start and IPO reference policy. Responses are partitioned into two pools:

$$\text{chosen} : \text{score}(y) = 1, \quad \text{rejected} : \text{score}(y) \neq 1 \wedge \text{len}(y) \geq T. \quad (8)$$

We set $T = 1200$ characters, a threshold that cleanly separates committed traces from rambling ones: 95% of correct traces fall under 931 characters, while 90% of rambling failures exceed 1,695 characters. Each pool is deduplicated by exact text, capped at two rollouts per pool per prompt, and paired index-by-index. Only prompts with at least one rollout in each pool are included, yielding 1,131 preference pairs.

5.2 Experimental Setup

Targeted FC experiments investigate sequential three-stage correction of targeted failure modes in the Countdown task. In the first stage, we start from the default RLOO checkpoint and train on `countdown_tasks_3to4` using RLOO augmented with an arithmetic process reward for 100 steps. The RLOO hyperparameters follow the BT RLOO setup in Section 4.2. In the second stage, RFT resumes from the Stage-1 checkpoint and samples 16 rollouts per prompt from the train split of `countdown_tasks_3to4`, retaining the top-5 distinct correct rollouts for each prompt as new training examples. We then fine-tune the Stage-1 checkpoint with SFT on this filtered dataset for 1 epoch, using a learning rate of 5×10^{-6} , batch size 16, and gradient accumulation over 4 steps, with the remaining optimizer settings matching the BT SFT setup in Section 4.2. In the third stage, we construct anti-rambling preference pairs from rollouts of the RFT checkpoint as described in Section 5.1.3 and train the model with IPO for 1 epoch, using $\beta = 0.1$ and the same optimizer settings as RFT.

We use the same evaluation setting and hardware as the BT experiments (Section 4.2). Each stage is benchmarked incrementally against the default RLOO checkpoint to isolate the contribution of each intervention. In addition to `pass@1`, `pass@16`, and `NO-ANSWER` rate, we report the following diagnostic metrics:

- Consistency gap: `pass@16` – `pass@1`, measuring the gap between solution capability and reliable execution.
- BAD-NUMS: the fraction of rollouts that use an invalid number set.
- WRONG-VAL: the fraction of rollouts that use valid numbers but produce an incorrect final value.
- Arithmetic incidence: the fraction of rollouts containing at least one arithmetic error.

- Arithmetic volume: the average number of arithmetic operations per rollout.
- Per-statement error rate: the ratio of incorrect arithmetic statements to total arithmetic statements, measuring the density of arithmetic mistakes within reasoning traces.

5.3 Results

Table 2 reports all metrics for the three stages of the post-training pipeline benchmarked on the default RLOO baseline. The complete sequence improves pass@1 from 0.56 to 0.62. Crucially, each stage successfully mitigates its targeted failure mode while preserving the behavioral gains of the preceding stages.

Table 2: Targeted FC results across the three stages, benchmarked against the default RLOO baseline (50 test prompts, 16 rollouts each).

	Default RLOO	RLOO _{FC}	RFT _{FC}	IPO _{FC}
Pass@1	0.56	0.56	0.60	0.62
Pass@16	0.76	0.76	0.74	0.74
Consistency Gap	0.20	0.20	0.14	0.12
NO-ANSWER Rate %	25.9	27.2	29.6	25.4
BAD-NUMS %	10.9	9.8	9.1	8.8
WRONG-VAL %	6.2	7.5	3.5	4.8
Arithmetic Incidence %	32.2	18.8	18.8	21.4
Arithmetic Volume (ops)	29.1	29.2	29.6	28.4
Arithmetic Error Rate %	3.04	0.98	1.03	1.40

5.3.1 Arithmetic process reward

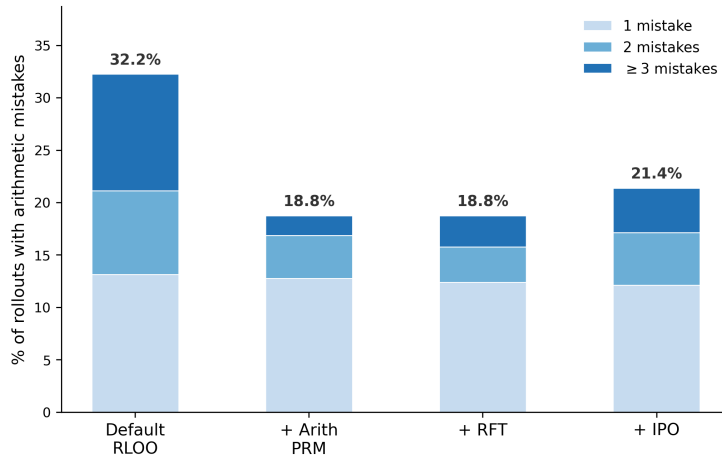


Figure 4: Arithmetic mistakes per rollout across the targeted FC stages. The arithmetic PRM collapses the heavy ≥ 3 -mistake tail; this gain persists through subsequent RFT and IPO stages.

The process reward sharply improves arithmetic accuracy. As shown in Table 2, incidence falls from 32.2% to 18.8%, and the per-statement error rate drops threefold from 3.04% to 0.98%. Because the overall volume of arithmetic remains constant at roughly 29 statements per rollout, we can confirm that the model learns to make fewer mistakes rather than artificially avoiding math to hack the reward. As shown in Figure 4, this intervention drastically reduces the number of rollouts with multiple arithmetic mistakes, and the gains persist through the rest of the pipeline.

However, pass@1 and pass@16 remain essentially unchanged from the default RLOO model. The reason is that arithmetic mistakes in the reasoning trace are largely decoupled from final-answer correctness. Figure 5 shows that the model explores many candidate calculations during its search, and

a wrong calculation on a branch it later abandons does not affect the expression it ultimately commits. In the default RLOO model, 20% of correct rollouts still contain at least one arithmetic mistake, and 35% of all rollouts with a mistake are nonetheless correct. Since suppressing these mistakes produces cleaner reasoning without changing which answer is committed, reducing arithmetic errors alone cannot translate to better accuracy.

```

<think>
1. 35 - 3 = 32
   84 - 35 = 49
   84 - 3 = 71      <- wrong (=81)
[...]
3. 84 - 3 = 77      <- wrong (=81)
   77 - 35 = 42
[... more attempts ...]
8. New approach:
   84 / 3 = 28
   28 + 35 = 63!
</think>
<answer> (84 / 3 + 35) </answer>

```

Figure 5: Arithmetic mistakes decoupled from answer correctness. This Default RLOO rollout on $[3, 35, 84] \rightarrow 63$ miscomputes $84 - 3$ on exploratory branches yet commits $(84/3) + 35$ from a separate valid path, reaching the correct target despite trace-level errors.

5.3.2 Rejection-sampling fine-tuning

Table 2 shows that RFT slightly improves performance stability with a smaller consistency gap, lifting pass@1 to 0.60 while keeping pass@16 near its peak (0.74) and leaving the arithmetic gains intact (per-statement error rate 1.03%). The improvement may be because sampling diverse correct traces ($T=1.0$, $N=5$) provides the model with multiple reasoning paths for the same problem, encouraging better reasoning generalization and thus higher answer accuracy and performance stability.

5.3.3 Preference optimization.

Table 2 shows that while the first two stages gradually raise the NO-ANSWER rate from the baseline of 25.9% to 29.6%, the final preference optimization stage successfully suppresses rambling to 25.4%, dropping below the original baseline. This improvement pushes pass@1 to its highest value of 0.62.

5.3.4 Failure Cases

Premature commit
(uses too few numbers):

```

<think>
1. First, get close to 82:
   - 83 - 1 = 82
[...]
4. This works perfectly!
</think>
<answer>83 - 1</answer>

```

Hallucinated operand
(uses invalid numbers):

```

<think>
First attempt:
6 + 14 = 20
20 + 42 = 62 (too high)
[... reuses 20 below ...]
42 + 20 = 62
62 - 14 = 48
48 - 6 = 42 (This works!)
</think>
<answer>(42+20)-14-6</answer>

```

Missed near-solution
(fails to recover and stuck):

```

<think>
First attempt:
26 + 4 = 30 (too low)
26 - 4 = 22 (too low)
[... a few paths ...]
New approach:
26 + 4 = 30
30 + 2 = 32
32 + 11 would be too high
[... repeats this ~24x,
never revisiting
26 - 4 = 22 ...]
Wait, let's try:
26 + 4 =

```

Figure 6: Three persistent failure modes after the full targeted FC pipeline. Left $[83, 78, 1, 39] \rightarrow 82$: premature commit using only two of four numbers. Middle $[6, 14, 42] \rightarrow 42$: hallucinated operand reusing the intermediate result $20=6+14$. Right $[26, 4, 2] \rightarrow 44$: missed near-solution, never revisiting $26-4=22$ despite $(26-4) \times 2=44$.

Figure 6 illustrates three failure modes that persist after the full pipeline. The first trajectory shows a premature commit where the model reaches the target but fails to use all provided numbers. The

second trajectory demonstrates a hallucinated operand, as the model invalidly reuses an intermediate calculation as a given input. The third trajectory reveals a lack of structured backtracking: the model fixates on a dead path and exhausts its token budget instead of revisiting promising earlier steps. These cases highlight fundamental limits in the model’s reasoning and coherence at this scale that post-training alone cannot overcome.

6 Discussion

Our two approaches yield contrasting outcomes: structured BT provides a negative result that reveals capacity limitations, while targeted FC demonstrates that targeted sequential interventions can improve reasoning. We discuss the limitations of each, persistent challenges, and broader implications.

Limitations of structured BT. The synthetic backtracking data injects dead-end branches randomly, which may produce traces too naive to provide useful training signal—the model learns to imitate an arbitrary search structure rather than strategic problem-solving. Crafting more realistic dead ends that resemble natural reasoning errors could help, but doing so is difficult without human assessment. Furthermore, with only 1,000 synthetic training traces, the dataset may be too small for the model to learn the underlying backtracking logic rather than merely memorizing the surface format. For a 0.5B model, learning a structured search grammar while simultaneously performing arithmetic reasoning may inherently exceed the capacity budget, which explains why BT consumed the majority of our development time yet still underperformed. This failure underscores a cautionary lesson: the overhead of structural approaches should be weighed against simpler alternatives, and a promising future direction is to evaluate BT on larger models where the capacity cost is proportionally smaller.

Persistent failure modes after targeted FC. Although targeted FC improves accuracy by sequentially targeting three identified bottlenecks, a persistent failure mode remains: the model misses near-solutions and never revisits promising branches it briefly explored. This is precisely the kind of failure that successful backtracking could address, highlighting the continued importance of recovery ability in multi-step mathematical reasoning and suggesting that the goals of structured BT and targeted FC are complementary.

Broader impact. Both structured BT and targeted FC are general-purpose approaches that could apply to reasoning domains beyond mathematical tasks. Structured BT could enhance any task that benefits from systematic search, such as combinatorial optimization or program synthesis. Targeted FC for specific failure modes through process rewards and preference optimization could be adapted to domains such as commonsense reasoning or multi-hop question answering.

7 Conclusion

We investigated two approaches to improving multi-step arithmetic reasoning in a 0.5B language model on the Countdown task. Structured BT introduces special backtracking tokens with a rule-based PRM that provides dense feedback on search quality. While the model exhibits backtracking behavior after BT training, it underperforms the default baseline—the model learns the bookkeeping format of state and backtrack tokens without acquiring genuine search logic. Analyzing this failure reveals three concrete reasoning bottlenecks—arithmetic mistakes within the trace, inconsistent commitment to correct solutions, and rambling that exhausts the token budget without answering—which targeted FC addresses through a sequential pipeline of an arithmetic PRM, rejection-sampling fine-tuning, and anti-rambling preference optimization. Targeted FC achieves pass@1 of 0.62 (+11% over the RLOO baseline), cuts the consistency gap to 0.12, and reduces arithmetic error rates threefold.

Two insights emerge from this work. First, a small model’s limited capacity budget constrains what post-training can achieve: BT’s structural grammar consumes capacity needed for reasoning itself, while targeted FC’s lightweight sequential interventions succeed precisely because each stage adds minimal overhead. This suggests that under capacity constraints, targeting specific failure modes one at a time composes more effectively than monolithic structural changes or multi-objective reward shaping. Second, arithmetic mistakes in the reasoning trace are largely decoupled from final-answer

correctness; the accuracy gains come from consolidating correct solutions (RFT) and suppressing non-committal behavior (IPO), not from cleaner arithmetic.

For future work, we identify two directions: (1) applying structured BT at larger model scales where the capacity overhead of structural bookkeeping is proportionally smaller, and (2) replacing randomly generated backtrack skeletons with high-quality, strategically chosen traces that better reflect natural reasoning failures.

8 Team Contributions

- **Eric Xia:** Designed the extension pipeline and completed the proposal; implemented structured BT, including synthetic SFT trace generation and context restoration during sampling; implemented targeted FC, including rejection-sampling fine-tuning and preference optimization; contributed to the final report.
- **Christine Li:** Implemented the default baselines (SFT, IPO, and RLOO) and completed the milestone report; extended RLOO with process reward models for both structured BT and targeted FC; contributed to the final report and created the poster.

Changes from Proposal. We completed the backtracking-based extension proposed initially, referred to as structured BT in this report. Because structured BT underperformed and revealed several failure modes, we developed an additional pipeline, targeted FC, that sequentially targets specific failures through three correction stages.

References

- [1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs. arXiv:2402.14740 [cs.CL]
- [2] Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. 2023. A General Theoretical Paradigm to Understand Learning from Human Feedback. arXiv:2310.12036 [cs.AI]
- [3] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heather Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168 [cs.LG]
- [4] Batu El, Mehmet Hamza Erol, and Hannah Park-Kaufmann. 2025. Reinforcement Learning Training for Dynamic Context Management in Mathematical Reasoning. Preprint. Under review.
- [5] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. 2024. Stream of Search (SoS): Learning to Search in Language. arXiv:2404.03683 [cs.AI]
- [6] Woosuk Kwon, Zhuohan Li, Sicheng Zuo, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180 [cs.LG]
- [7] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let’s Verify Step by Step. arXiv:2305.20050 [cs.LG]
- [8] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]
- [9] Jiayi Pan, Junjie Zhang, Xingyao Wang, Hao Peng, and Graham Neubig. 2025. TinyZero. Open-source reproduction of DeepSeek-R1-Zero on small models.

- [10] Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. Technical Report.
- [11] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG]
- [12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]
- [13] Various. 2024. Countdown Numbers Game. (2024). A combinatorial reasoning benchmark requiring multi-step arithmetic to reach a target from given numbers.
- [14] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2023. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. arXiv:2312.08935 [cs.CL]
- [15] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL]
- [16] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models. arXiv:2308.01825 [cs.CL] <https://arxiv.org/abs/2308.01825>
- [17] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping Reasoning With Reasoning. arXiv:2203.14465 [cs.LG]