Beyond Correctness: Robustness-Driven Evolutionary Self-Training for Large Language Models

Wei Guo[1,*] **Hongyao Tang**[1] **Yi Ma**[2] **Jinyi Liu**[1,3] **Pengyi Li**[1,†]
Jing Liang[1] **Yifu Yuan**[1] **Yan Zheng**[4,3] **Jianye Hao**[1,3,†]
[1]School of Computer Software, Tianjin University
[2]School of Computer and Information Technology, Shanxi University
[3]MemoraX AI [4]School of Artificial Intelligence, Tianjin University

Abstract

Outcome-based post-training methods for mathematical reasoning rely on binary feedback, treating all correct trajectories equally. However, this masks a critical distinction: some correct paths are structurally brittle and prone to cascading errors under inevitable autoregressive sampling noise, while others are robust and maintain their correctness despite these natural decoding variations. Optimizing indiscriminately over these brittle solutions leads to inefficient learning, as the model wastes capacity learning brittle reasoning strategies. To address this, we propose Trajectory Robustness-Driven Evolutionary Self-Training (TREST). Instead of relying solely on binary feedback, our framework explicitly prioritizes trajectory robustness by integrating evolutionary algorithms with supervised fine-tuning. Crucially, our analysis reveals that trajectory robustness serves as a strong natural proxy for genuine mathematical insight. By conceptualizing reasoning paths as evolving individuals within a population, this evolutionary process naturally marginalizes brittle, brute-force calculations in favor of robust, insight-driven strategies. By effectively internalizing these strategies, TREST achieves superior reasoning performance. Experiments on a range of mathematical benchmarks demonstrate that under aligned budgets, TREST outperforms outcome-based baselines, establishing a highly effective alternative to conventional post-training paradigms.

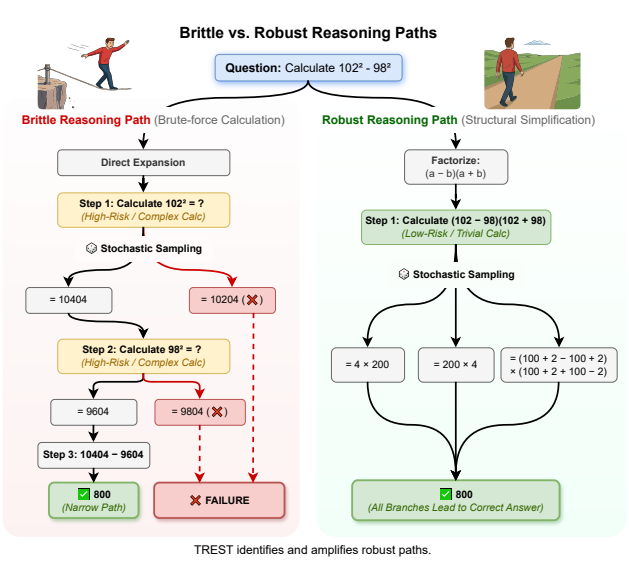
1 Introduction

Improving the reasoning ability of large language models (LLMs) has become a central research focus. Paradigms like reinforcement learning with verifiable rewards (RLVR) [11, 4] and rejection sampling fine-tuning (RFT) [25, 20] have become standard practice for complex mathematical and logical tasks. However, a critical post-training bottleneck remains: objective reasoning domains generally lack naturally continuous feedback signals. Consequently, these pipelines largely rely on binary rewards derived solely from verifiable outcomes, unless paired with computationally expensive process reward models.

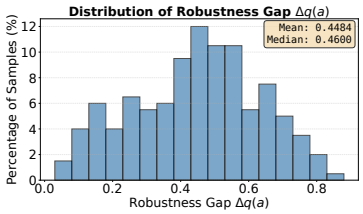
This outcome-based binary filter creates a critical training signal blind spot. By treating all correct trajectories as equally valuable, it discards crucial information about the intrinsic quality of the reasoning path. Consequently, it fails to provide a granular signal indicating *which* correct reasoning path is most reliable for long-term training, lacking the ability to measure a trajectory's fault tolerance and capacity for local exploration. This information loss is particularly dangerous given the mechanics of autoregressive generation, where LLMs suffer from compounding error probabilities and the

*Email: guo_wei_2024@tju.edu.cn.
†Corresponding authors. Emails: lipengyi@tju.edu.cn and jianye.hao@tju.edu.cn.

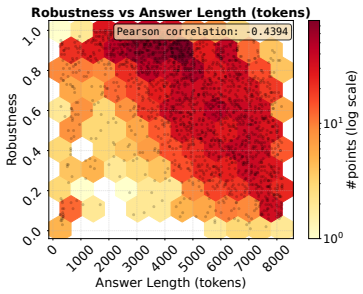
40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Theoretical Topology of Reasoning Paths



(b) Distribution of Robustness Gap



(c) Robustness vs. Answer Length

Figure 1: **Motivation and empirical evidence of the reasoning robustness gap. (a)** Conceptual illustration: a brittle computational path (left) relies on lengthy and intricate calculations with more opportunities for numerical errors, while a robust path (right) uses structural simplification to avoid such calculations and maintain dense valid neighborhoods. **(b)** Distribution of the intra-question robustness gap (i.e., the range $\Delta q(a)$ between the maximum and minimum offspring correctness) across 200 evaluated problems, revealing a massive quality chasm currently masked by binary rewards. **(c)** A hex-binned scatter plot showing a moderate negative correlation ($r = -0.4394$) between answer length (tokens) and robustness. While concise paths tend to be more robust, length alone is an insufficient proxy for trajectory quality.

snowball effect [35]. Without granular metrics, binary filters indiscriminately absorb *brittle reasoning paths* (overly complex derivations relying on mechanical token-level precision). Here, lengthy and intricate calculations create more opportunities for numerical errors and cascading failure (Figure 1a, left). Conversely, *robust reasoning paths* favor structural simplification. They are semantically transparent: even under natural decoding variance, structural robustness ensures the local neighborhood remains densely populated with valid alternatives (Figure 1a, right).

Empirical evidence confirms this theoretical gap. A pilot study on 200 moderate-difficulty questions from DeepScaleR-Preview-Dataset (hereafter referred to as the DeepScaleR training pool) [23] reveals that *variation in robustness is ubiquitous across correct solutions to the same problem.* By subjecting valid trajectories to controlled local perturbations, we estimate their expected offspring correctness rate, $q(a)$. As shown in Figure 1b, the intra-question robustness gap (Δq) is substantial, frequently approaching 0.5. This confirms that current outcome-based pipelines, which assign a uniform reward of 1, mask a massive quality chasm. Furthermore, while robustness shows a moderate negative correlation with trajectory length ($r = -0.4394$, Figure 1c), simple length penalties insufficiently proxy reasoning quality, necessitating the direct prioritization of the endogenous robustness signal $q(a)$. We also investigate whether explicitly favoring shorter trajectories improves training performance; the experimental setup and results are reported in Appendix F. Relying solely on binary correctness ignores this distinction and causes severe computational waste. Although Process Reward Models (PRMs) [31, 18] and continuous signal reshaping [15, 28, 6, 24] provide denser supervision, they overlook the intrinsic *robustness* of reasoning paths, leading models to absorb rigid solutions prone to compounding errors.

To overcome the binary reward bottleneck, we propose **Trajectory Robustness-Driven Evolutionary Self-Training (TREST)**. We complement binary correctness feedback with a fine-grained, continuous quality measure: *reasoning path robustness*. TREST treats correct reasoning paths as evolving individuals and generates offspring through local perturbations. Using the offspring correctness rate

as a measure of robustness and an endogenous fitness signal, TREST favors robust paths over brittle ones as anchors for further expansion.

By iteratively expanding around these robust anchors, TREST converts isolated correct trajectories into dense clusters of high-quality variations. Unlike process-based pipelines relying on computationally intensive external models, TREST bypasses external reward estimation. By utilizing the offspring correctness rate, a natural byproduct of the evolutionary process, as an endogenous fitness signal, it directs generation compute toward collecting high-quality reasoning trajectories, recycling valid perturbations directly for robust fine-tuning.

Our main contributions are summarized as follows:

- **A Robustness-Centric Learning Paradigm:** We move beyond the limitations of binary rewards by establishing *structural robustness* as a quantifiable evolutionary fitness metric. By introducing the offspring correctness rate as a continuous fitness signal, TREST effectively distinguishes robust reasoning paths from brittle paths, resolving a fundamental blind spot in outcome-based post-training.
- **Adaptive Evolutionary Optimization with Entropy-Guided Mutation:** We propose an integrated evolutionary framework that adaptively reallocates computational resources via batched Thompson Sampling. By coupling this selection mechanism with entropy-guided mutation, TREST systematically deprioritizes brittle trajectories and populates the fine-tuning pool with high-quality, structurally robust reasoning paths.
- **Empirical Validation and Mechanistic Insights:** Experiments demonstrate that TREST achieves a superior reasoning ceiling compared to strong outcome-based RL baselines under identical training environments. Beyond quantitative gains, our qualitative analysis reveals a strong intrinsic correlation between trajectory robustness and deep *Mathematical Insight*, confirming that our framework successfully drives the model to favor mathematical insight over brute force calculations.

2 Related Work

Outcome-based Post-training and Reasoning Robustness. A common recipe for improving LLM reasoning relies on verifiable binary rewards, such as RFT and RLVR [33, 30, 34]. While effective, this outcome-based paradigm treats all correct trajectories equally, struggling to distinguish high-quality reasoning from brittle derivations, leading to computational inefficiencies. Process Reward Models (PRMs) [14, 31, 18] and continuous signal reshaping [15, 6] provide dense supervision but remain susceptible to reward hacking [3, 4], and their localized evaluation overlooks the global *robustness* of the reasoning chain. Recent diagnostic studies [16, 1] investigate reasoning vulnerabilities at the model level rather than quantifying trajectory-level robustness and using it to guide training. In contrast, TREST directly verifies global fault tolerance by forcing the reasoning process into alternative branches and evaluating complete continuations.

Evolutionary Search and Entropy-Aware Exploration. Evolutionary algorithms have been applied to LLM reasoning to navigate vast search spaces [19, 22, 7]. However, existing methods often rely on constrained functional spaces or expensive inference-time search, lacking a continuous metric for path robustness. To improve search efficiency, recent methods leverage step-level metrics (e.g., entropy or advantage functions) to guide exploration via trajectory expansion [13, 36, 32], gradient shaping [26, 27], or PRM rollout estimation [2]. However, these approaches face fundamental limitations: gradient shaping remains trapped in binary RL instabilities, PRM rollout estimators do not directly reuse extensively sampled trajectories for policy fine-tuning after computing a scalar score, and guided expansions restrict search to non-iterative branches without dynamic robustness feedback. None explicitly prioritize structural path robustness. TREST diverges by elevating perturbation-based robustness to an *endogenous fitness metric* within a multi-generational loop. By iteratively mutating robust offspring and retaining them, TREST constructs a high-density fine-tuning dataset, mitigating the information scarcity of binary rewards while reusing verified rollouts for fine-tuning.

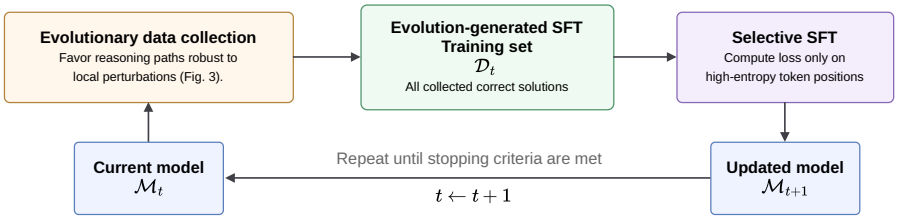


Figure 2: **Overview of TREST.** At iteration t, current model $\mathcal{M}_t$ performs evolutionary data collection (problem selection, population seeding, solution expansion) to construct SFT training set $\mathcal{D}_t$. The model is then updated via selective supervised fine-tuning (SFT), yielding $\mathcal{M}_{t+1}$.

3 TREST: Trajectory Robustness-Driven Evolutionary Self-Training

3.1 Overview

We propose Trajectory Robustness-Driven Evolutionary Self-Training (**TREST**), an alternating framework iterating between **evolutionary data collection** and **supervised fine-tuning**. Starting with an initial language model $\mathcal{M}_0$, at iteration t, TREST uses the current model $\mathcal{M}_t$ to expand verified correct solutions into a training set $\mathcal{D}_t$, and then fine-tunes the model on $\mathcal{D}_t$ to obtain $\mathcal{M}_{t+1}$:

$$\mathcal{M}_t \xrightarrow{\text{evolution}} \mathcal{D}_t \xrightarrow{\text{SFT}} \mathcal{M}_{t+1}.$$

TREST treats correct solutions as an evolving population, preferentially reproducing those remaining productive under local perturbations. This favors trajectories with dense correct neighborhoods, efficiently converting compute into useful supervision. Figure 2 illustrates this overall pipeline.

3.2 Problem Selection and Population Seeding

At each iteration, we build a selected problem set $\mathcal{P} = \{p_1, \ldots, p_N\}$ via rejection sampling from the full pool $\mathcal{Q}$. For the candidate pool, we generate M solutions per problem and compute their empirical correctness rates ρ_p. To ensure informativeness, we dynamically discard a subset of the easiest problems exhibiting the highest ρ_p, filtering out trivial tasks that offer limited learning signals (see Appendix A.1 for detailed dynamic screening mechanisms).

For each accepted problem $p \in \mathcal{P}$, all correct solutions discovered form the first generation of its verified population $\mathcal{A}_p$. To track parent productivity during downstream evolution, we maintain two statistics for each solution $a \in \mathcal{A}_p$: the number of successful offspring produced (S_a), and the number of times selected as a parent (T_a). We initialize $S_a = 0$ and $T_a = 0$ for all seeds.

3.3 Robustness-Guided Evolution

For each problem p, we evolve $\mathcal{A}_p$ up to generation $G_{\max}$, terminating early if $|\mathcal{A}_p| \geq C$. Each subsequent generation allows K reproduction attempts, involving parent selection and mutation (see Figure 3).

3.3.1 Parent Selection via Batched Thompson Sampling

A central concept in TREST is **reasoning path robustness**, defined as the expected offspring correctness rate under local perturbation:

$$q(a) = \mathbb{E}_{a' \sim \mathrm{Mutate}(a)}\left[\mathbf{1}_{\mathrm{Correct}(a')}\right],$$

where $a' \sim \mathrm{Mutate}(a)$ denotes an offspring generated by locally perturbing a and resampling its continuation, and $\mathrm{Correct}(a') \in \{0, 1\}$ indicates verified correctness. Highly robust solutions rely on structurally sound reasoning, making them resilient to perturbations and ideal parents.

Since $q(a)$ is unknown a priori, we frame parent selection as a multi-armed bandit problem. We model reproduction outcomes as Bernoulli variables and maintain a Beta posterior over each solution's success probability using its historical statistics (S_a, T_a) with a $\mathrm{Beta}(1, 1)$ prior. To balance exploitation and exploration, we select parents via Thompson sampling by drawing from these posteriors.

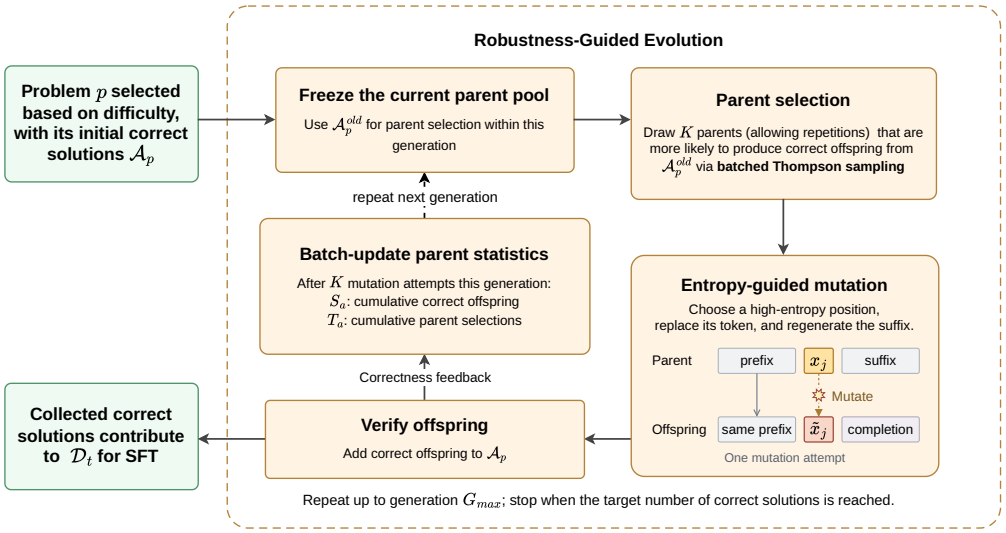


Figure 3: **Detailed view of robustness-guided evolution.** Parent selection utilizes batched Thompson sampling. Offspring are generated via entropy-guided mutation; only correct offspring are retained. Parent statistics update after each generation; new correct offspring become eligible parents in the next generation.

We adopt a **batched-update** variant of Thompson sampling. At the start of each generation, we freeze the candidate set to $\mathcal{A}_p^{\text{old}}$. All K selections rely on these fixed posteriors, and statistics update only post-generation. New correct offspring enter $\mathcal{A}_p$ immediately but become eligible parents only in the next generation, preventing overreaction to high-variance early outcomes.

3.3.2 Entropy-Guided Mutation

Given parent $a = (x_1, \ldots, x_L)$, let $\pi_i(v) = P(v \mid p, x_{<i})$ denote the next-token distribution. We compute token-level entropy $H_i = -\sum_v \pi_i(v) \log \pi_i(v)$. To identify candidate mutation locations, we restrict our pool to the first 80% of the reasoning trajectory, excluding the trailing 20% since concluding steps typically involve deterministic calculations rather than meaningful logical pivots. Within this valid prefix, we select a subset of positions exhibiting the highest token-level entropy (see Appendix A.2). We uniformly sample a position j from these candidates, drawing a replacement token from the renormalized distribution excluding the original x_j:

$$\tilde{\pi}_j(v) = \frac{\pi_j(v)}{1 - \pi_j(x_j)} \text{ if } v \neq x_j \text{ else } 0, \qquad \tilde{x}_j \sim \tilde{\pi}_j.$$

Conditioned on this modified prefix, we autoregressively regenerate the suffix to obtain offspring a_{child}. This operation acts as both a local search operator and a diagnostic probe of expandability. If a_{child} is correct, we add it to the population ($\mathcal{A}_p \leftarrow \mathcal{A}_p \cup \{a_{\text{child}}\}$) and initialize $S_{a_{\text{child}}} = 0, T_{a_{\text{child}}} = 0$.

3.4 Policy Distillation via Selective SFT

Post-evolution, we aggregate all verified correct solutions into the training set $\mathcal{D}_t = \{(p, a) \mid p \in \mathcal{P}, a \in \mathcal{A}_p\}$, and update the model to distill the discovered trajectories into parameters:

$$\mathcal{M}_{t+1} = \arg\min_{\mathcal{M}} \sum_{(p,a)\in\mathcal{D}_t} \mathcal{L}_{\text{SFT}}(\mathcal{M}; p, a).$$

Inspired by recent findings in RL [27], we restrict the SFT loss exclusively to the high-entropy tokens targeted during mutation. The cited study demonstrates that restricting updates to high-entropy "forking tokens" improves reasoning while preventing the disruption of deterministic, low-entropy segments. By adapting this principle, TREST focuses gradients solely on critical decision nodes to efficiently internalize the discovered logical transitions. Algorithm 1 summarizes the procedure. We employ cross-problem batched inference to maximize GPU utilization (Appendix A.3).

Algorithm 1 Trajectory Robustness-Driven Evolutionary Self-Training (TREST)

Input: Initial model $\mathcal{M}_0$, full problem pool $\mathcal{Q}$
Hyperparams: N (problems per iter), M (initial samples per problem), $G_{\max}$ (max generations)
C (max solutions per problem), K (mutations per generation)
Output: Optimized model $\mathcal{M}$
$\mathcal{M} \leftarrow \mathcal{M}_0$
repeat
  $(\mathcal{P}, \{\mathcal{A}_p\}_{p\in\mathcal{P}}) \leftarrow \text{InitialSampling}(\mathcal{M}, \mathcal{Q}, N, M)$
  for each problem $p \in \mathcal{P}$ **do**
    for $g = 2, \ldots, G_{\max}$ **do**
      if $|\mathcal{A}_p| \geq C$ **then break**
      $\mathcal{A}_p^{\text{old}} \leftarrow \mathcal{A}_p$
      $\Delta S_a \leftarrow 0,\ \Delta T_a \leftarrow 0,\ \forall a \in \mathcal{A}_p^{\text{old}}$
      for $k = 1, \ldots, K$ **do**
        $a \leftarrow \text{ThompsonSelect}(\mathcal{A}_p^{\text{old}})$
        $a_{\text{child}} \leftarrow \text{Mutate}(\mathcal{M}, a)$
        $\Delta T_a \leftarrow \Delta T_a + 1$
        if $\text{IsCorrect}(a_{\text{child}})$ **then**
          $\mathcal{A}_p \leftarrow \mathcal{A}_p \cup \{a_{\text{child}}\}$
          $S_{a_{\text{child}}} \leftarrow 0,\ T_{a_{\text{child}}} \leftarrow 0$
          $\Delta S_a \leftarrow \Delta S_a + 1$
        end if
      end for
      $S_a \leftarrow S_a + \Delta S_a,\ T_a \leftarrow T_a + \Delta T_a,\ \forall a \in \mathcal{A}_p^{\text{old}}$
    end for
  end for
  $\mathcal{D}_t \leftarrow \{(p, a) \mid p \in \mathcal{P},\ a \in \mathcal{A}_p\}$
  $\mathcal{M} \leftarrow \text{SFT}(\mathcal{M}, \mathcal{D}_t)$
until stopping criteria are met
return $\mathcal{M}$

4 Experiments

To systematically evaluate the effectiveness and underlying mechanisms of TREST, we design our experiments to answer three core research questions:

- **Q1 (Reasoning Performance):** Does TREST yield superior complex mathematical reasoning capabilities compared to representative standard paradigms?
- **Q2 (Mechanistic Contributions):** How does each algorithmic component contribute to TREST's overall reasoning performance?
- **Q3 (Qualitative Insight):** Does the structural robustness intrinsically reflect deeper mathematical insight rather than brute-force mechanical execution?

4.1 Experimental Setup

The following setup applies to our main mathematical experiments.

Training Data and Evaluation Protocol. To ensure a controlled comparison, all aligned-budget methods train on the same 40,315-problem DeepScaleR training pool [23], which combines five source collections: AIME, AMC, Omni-MATH, STILL, and MATH [9]. We report performance separately on the five-benchmark evaluation suite released with DeepScaleR: AIME'24, AMC'23, MATH-500 [14], Minerva [12], and OlympiadBench [8]. The training pool and evaluation benchmarks are separate collections rather than corresponding splits of a single dataset.

Models and Baselines. All main experiments share the same backbone model: DeepSeek-R1-Distill-Qwen-1.5B [4]. To ensure a consistent evaluation environment and maintain a manageable

computational footprint, we uniformly enforce a maximum output length of 8,192 tokens (9,216 total context). We compare TREST against representative baselines to establish a rigorous evaluation:

- **GRPO [20]:** Selected as the primary value-model-free RL baseline, configured with competitive empirical hyperparameters to ensure a fair comparison.
- **Iterative RFT:** An iterative rejection sampling baseline that rigorously retains all logically correct trajectories, providing a strong outcome-based reference for evaluating TREST's selection mechanism.
- **Heavily-Scaled RL Pipelines:** Models such as **DeepScaleR** [23] and **FASTCuRL-v3** [21] are included strictly as high-quality reference bounds. Although their unconstrained computational training budgets drastically exceed ours, they provide context for the performance ceiling of 1.5B models in this domain.

Cost Alignment and Sampling Configurations. To ensure an equitable comparison, we adopt a strict limit of 200,000 *total generated candidate trajectories (rollouts)* as our unified cost alignment metric. Because representative RL and non-RL baselines rely on different engineering frameworks, wall-clock comparisons inevitably conflate algorithmic performance with framework-level optimization. By constraining the total rollout budget, we establish a framework-agnostic comparison that isolates the core reasoning efficiency of each method.

Importantly, matching the rollout budget is a highly conservative lower-bound for TREST. Under this identical rollout count, TREST generates substantially fewer tokens via suffix-only mutations and incurs significantly lower optimization overhead by restricting SFT supervision to high-entropy tokens.

Within the cumulative training-run rollout cap, Iterative RFT and TREST allocate rollouts according to their respective data-collection procedures. Iterative RFT processes 2,048 problems per iteration and draws 8 samples per problem, spreading outcome-based collection across a broad set of problems. TREST instead targets a quota of 100 selected problems per iteration; during screening, it draws 20 initial samples per candidate problem to estimate problem difficulty and seed verified populations. It then allocates additional rollouts to multi-generational mutation within the selected problems; the resulting offspring outcomes update trajectory-level robustness estimates and guide parent selection. Each initial sample and each mutation-generated offspring is counted separately toward the same cumulative rollout cap. We also evaluate Iterative RFT with 100 problems per iteration and 20 initial samples per problem; the corresponding configuration and results are reported in Appendix E.

Implementation Details. Appendices A.4 and A.5 detail our phase-decoupled VRAM offloading architecture and the hyperparameters for all methods, respectively.

4.2 Main Results: Complex Mathematical Reasoning (Q1)

To comprehensively evaluate the reasoning performance of TREST (addressing research question **Q1**), we compare its performance against representative baselines under a strict computational alignment. As established by our budget-bounded evaluation protocol, all comparable methods are strictly constrained to a maximum search budget of 200,000 generated rollouts (Cost $\leq$ 0.2M).

Performance Comparison. As summarized in Table 1, TREST achieves leading performance across representative paradigms under the strict 0.2M rollout constraint, outperforming the primary aligned baseline (GRPO) by an absolute average margin of **+4.49%**. This superiority is most pronounced on tasks demanding deep logical planning; notably, on the AIME'24 benchmark, TREST nearly doubles the performance of standard GRPO (35.33% vs. 18.67%). Furthermore, under the strict maximum output length constraint of 8,192 tokens, TREST even outperforms the extended-compute reference bounds (e.g., DeepScaleR) on the AIME'24 benchmark, despite utilizing less than $1/12\times$ of their training budget. This highlights TREST's core advantage of reasoning efficiency: by actively deprioritizing brittle, error-prone derivations, it distills structurally robust paths that maximize reasoning performance. Practically, this efficiency translates to an aggregated cost of approximately 85.36 GPU hours for the 33 training iterations required to produce our evaluated checkpoint in our main results (detailed in Appendix A.6), offering a highly compute-efficient alternative to resource-intensive post-training methods.

Table 1: Main results on the mathematical reasoning suite. All models share the same 1.5B base and are evaluated uniformly ($T = 0.6$, **max 8,192 output tokens**). **Cost** denotes total training rollouts. All metrics are averaged over 5 independent runs (reporting mean $\pm$ standard deviation for the overall average). For aligned-budget methods ($\leq$ 0.2M), we evaluate the best-performing checkpoint to ensure a highly competitive comparison; top results are **bolded**. Extended budget references (bottom) denote external checkpoints evaluated under the identical 8,192-token constraint. Under strictly aligned budgets, TREST achieves the highest average accuracy.

Method	AIME'24	AMC'23	MATH500	Minerva	Olympiad	Average (↑)	Cost (↓)
Base Model:							
DeepSeek-R1-Distill-Qwen-1.5B	20.67%	47.95%	75.84%	23.31%	34.52%	40.46±1.19%	-
Aligned Budget ($\leq$ 0.2M Rollouts):							
Iterative RFT	23.33%	58.55%	80.88%	24.71%	39.76%	45.45±0.68%	0.197M
GRPO	18.67%	59.52%	80.96%	**26.18%**	41.39%	45.34±0.75%	0.200M
TREST (Ours)	**35.33%**	**64.82%**	**81.12%**	25.44%	**42.46%**	**49.83±1.45%**	**0.195M**
Extended Budget (Reference):							
FASTCuRL-V3	33.33%	65.30%	85.84%	31.18%	47.38%	52.61±1.60%	2.478M
DeepScaleR	34.00%	65.30%	84.64%	27.28%	46.96%	51.64±1.93%	2.519M

Mechanism Attribution: Implicit Robustness Weighting. We attribute this compute efficiency to the evolutionary reallocation of our sampling budget. By dynamically marginalizing brittle paths, TREST generates larger lineages of correct offspring for structurally robust trajectories. Consequently, these robust solutions naturally dominate the fine-tuning distribution, acting as an *implicit robustness weighting* that smoothly distills robust reasoning into the policy without requiring explicit loss modifiers.

4.3 Mechanistic Analysis and Ablation Studies (Q2)

To isolate TREST's core evolutionary mechanisms (Q2), we evaluate all variants under the same main-experiment setting, changing only the indicated mechanism(s). TREST and the three entropy-mechanism variants form a 2×2 factorial over entropy-guided mutation and selective SFT.

Table 2: **Mechanistic ablations under the main-experiment setting.** The controls replace robustness-guided parent selection with uniform parent selection, entropy-guided mutation with uniform mutation, and selective SFT with all-token SFT; all other mechanisms remain fixed. All variants use the same ≤ 0.2M cumulative rollout-budget cap. Scores are mean Pass@1 percentages over five stochastic evaluations. **Average** reports the mean $\pm$ standard deviation of the per-run five-benchmark macro averages; **Cost** is cumulative training rollouts.

Method	AIME'24	AMC'23	MATH500	Minerva	Olympiad	Average (↑)	Cost (↓)
TREST (All Components)	35.33	64.82	81.12	25.44	42.46	**49.83±1.45**	0.195M
– w/o Robustness-Guided Parent Selection	26.67	60.72	80.40	26.47	42.19	47.29±2.02	0.198M
Entropy-mechanism ablations (with TREST above: 2×2 factorial)							
– w/o Entropy-Guided Mutation	23.33	63.13	81.04	26.25	41.93	47.14±1.14	0.195M
– w/o Selective SFT	28.00	60.48	81.12	24.41	42.73	47.35±1.57	0.199M
– w/o Both Entropy Mechanisms	26.67	59.52	81.96	25.29	42.73	47.23±1.00	0.197M

TREST attains the highest five-benchmark macro average (49.83%). Replacing robustness-guided parent selection, entropy-guided mutation, or selective SFT lowers it to 47.29%, 47.14%, and 47.35%, respectively, supporting the contribution of each component under the main setting.

Interestingly, the factorial ablation reveals a coupled, non-additive pattern. With selective SFT, entropy-guided mutation raises the five-benchmark macro average from 47.14% to 49.83%, whereas under all-token SFT the two mutation strategies perform similarly (47.35% vs. 47.23%). Conversely, selective SFT provides a clear gain only when paired with entropy-guided mutation. Since broad updates to low-entropy segments can introduce substantial gradient noise [27], we hypothesize that

all-token SFT acts as an optimization bottleneck that diminishes the observable benefit of upstream exploration. These results suggest that the two mechanisms operate most effectively in tandem.

4.4 Qualitative Analysis: Does Structural Robustness Reflect Mathematical Insight? (Q3)

To uncover the mechanisms driving structural robustness (addressing research question **Q3**), we focus on **Mathematical Insight**: the ability to use theorems, abstractions, or clever simplifications to reduce problem complexity, bypassing brute-force arithmetic or exhaustive enumeration.

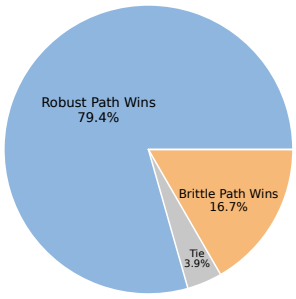


Figure 4: Pairwise comparison ($N = 102$). Robust trajectories exhibit superior Mathematical Insight in 79.4% of cases.

Table 3: Illustrative archetypes of strategic divergence. The **ID** column denotes the specific problem identifier within the DeepScaleR training pool. As shown, robust paths use central-symmetry reduction, apply pair-sum parameterization, employ systematic frameworks, maintain clear representations, or construct coherent event partitions, while brittle paths enumerate candidate lines, conduct fragmented factor searches, exhibit semantic entanglement, revert to mechanical execution, or exhibit inconsistent event modeling. Summaries of the reasoning traces are provided in Appendix C.

ID	Robust Path Reasoning Strategy	Brittle Path Reasoning Strategy
6657	Central-symmetry reduction	Enumeration of candidate lines
8622	Pair-sum parameterization	Fragmented factor search
12777	Systematic framework	Semantic entanglement
37594	Maintains clear representation	Reverts to mechanical execution
3754	Coherent event partition	Inconsistent event modeling

Experimental Design. We sampled 200 moderate-difficulty questions (pass rates $\rho \in [0.1, 0.9]$) from the DeepScaleR training pool, generating 10 *correct* paths each. Robustness was evaluated via offspring correctness across 100 mutational rollouts. To isolate strategic variance, we selected $N = 102$ problems whose robustness gap, Δq (the difference in offspring correctness between a problem's most and least robust paths) met or exceeded the dataset median. For each problem, DeepSeek-v3.2 [5] conducted pairwise Mathematical Insight comparisons between the most robust and brittle trajectories (see Appendix B for the exact evaluation prompt). A 10-run majority vote (5 forward, 5 reversed) eliminated positional bias. The judge reported definitive preferences in 96.1% of cases (4 ties), confirming its ability to reliably distinguish structural elegance from brute-force calculations.

The Dominance of Insight. As Figure 4 shows, Mathematical Insight strongly correlates with evolutionary survival. Excluding ties, trajectories with superior insight predominantly belonged to the highly robust subset (81 vs. 17 wins). This 79.4% overall win rate shows that LLM reasoning robustness effectively reflects differences in mathematical insight: more robust paths are substantially more likely to be judged insightful than brittle paths. Since mathematical insight can shorten reasoning through structural simplification, we further examine whether this association persists among nearly equal-length trajectories (Appendix G).

Archetypes of Strategic Divergence. To visualize these mechanics, we analyzed high-confidence cases where the evaluator reached a unanimous $(10/10)$ consensus on the robust path's strategic superiority. From these, after excluding invalid problem instances (e.g., those with incorrect reference answers or incomplete problem statements), we selected the 5 remaining problems with the most extreme robustness gaps (Δq) to isolate failure modes (Table 3; see Appendix C for detailed case studies, including summaries of the reasoning paths). A consistent pattern emerges: robust paths use central-symmetry reduction, apply pair-sum parameterization, employ systematic frameworks, maintain clear representations, or construct coherent event partitions to bypass complexity, whereas brittle paths enumerate candidate lines, conduct fragmented factor searches, exhibit semantic entanglement, revert to mechanical execution, or exhibit inconsistent event modeling.

4.5 Supplementary Cross-Domain Evaluation

Beyond our main study of mathematical reasoning, we conduct a supplementary code-generation experiment to explore the applicability of robustness-guided training in another verifiable domain. TREST improves held-out Pass@1 by 1.37 percentage points over its ablation without robustness-guided parent selection; experimental details and analysis are provided in Appendix H.

5 Conclusions

In this work, we introduced TREST, an evolutionary post-training paradigm that addresses the structural brittleness of LLM reasoning by establishing path robustness as a quantifiable optimization objective. By replacing external reward models with an endogenous fitness signal derived from entropy-guided perturbations, TREST systematically marginalizes brittle mechanical derivations in favor of robust reasoning strategies. Empirical evaluations demonstrate that TREST achieves superior reasoning accuracy under strictly aligned computational and context-length constraints. Furthermore, qualitative analysis reveals a strong correlation between structural robustness and mathematical insight, confirming that our framework successfully internalizes structural abstractions.

6 Limitations and Future Work

While TREST establishes a highly efficient paradigm for post-training, it has certain limitations. First, our empirical validation focuses primarily on mathematical reasoning, with a supplementary code-generation experiment; broader cross-domain validation remains future work. Second, although we demonstrate exceptional sample efficiency on 1.5B parameter architectures, verifying its scaling behavior on massive language models remains an important avenue for future research. Beyond addressing these limitations, future work could explore using PRMs to assist data selection and incorporating step-level robustness signals into PRM training.

Acknowledgments and Disclosure of Funding

This work is supported by the National Natural Science Foundation of China (Grant Nos. 62422605, 62533021, 92370132, 62541610, 624B2101), the National Key Research and Development Program of China (Grant No. 2026ZD0126104), the Tianjin Natural Science Foundation Project (No. 25JCQNJC01720), the Ministry of Education Key Laboratory of Brain-inspired Intelligent Perception and Cognition, University of Science and Technology of China (No. 2521006), and the Fundamental Research Program of Shanxi Province (Serial No. 202503021212091). We thank the anonymous reviewers for their valuable help in improving our manuscript.

References

[1] Ashwath Vaithinathan Aravindan and Mayank Kejriwal. Fragile thoughts: How large language models handle chain-of-thought perturbations. *arXiv preprint arXiv:2603.03332*, 2026.

[2] Lang Cao, Renhong Chen, Yingtian Zou, Chao Peng, Huacong Xu, Yuxian Wang, Wu Ning, Qian Chen, Mofan Peng, Zijie Chen, Peishuo Su, and Yitong Li. More bang for the buck: Process reward modeling with entropy-driven uncertainty. *arXiv preprint arXiv:2503.22233*, 2025.

[3] Jie Cheng, Gang Xiong, Ruixi Qiao, Lijun Li, Chao Guo, Junle Wang, Yisheng Lv, and Fei-Yue Wang. Stop summation: Min-form credit assignment is all process reward model needs for reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pages 131646–131671, 2025.

[4] DeepSeek-AI, Daya Guo, Dejian Yang, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[5] DeepSeek-AI, Aixin Liu, Aoxue Mei, et al. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. arXiv:2512.02556.

[6] Yiwen Duan, Jing Ye, and Xinpei Zhao. ALIVE: Awakening LLM reasoning via adversarial learning and instructive verbal evaluation. *arXiv preprint arXiv:2602.05472*, 2026.

[7] Kehua Feng, Keyan Ding, Zhihui Zhu, Lei Liang, Qiang Zhang, and Huajun Chen. CoT-Evo: Evolutionary distillation of chain-of-thought for scientific reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026.

[8] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

[9] 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. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks)*, 2021.

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[11] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, et al. TüLu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[12] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

[13] Qingbin Li, Rongkun Xue, Jie Wang, Ming Zhou, Zhi Li, Xiaofeng Ji, Yongqi Wang, Miao Liu, Zheming Yang, Minghui Qiu, and Jing Yang. CURE: Critical-token-guided re-concatenation for entropy-collapse prevention. *arXiv preprint arXiv:2508.11016*, 2025.

[14] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. Poster.

[15] Yile Liu, Yixian Liu, Zongwei Li, Yufei Huang, Xinhua Feng, Zhichao Hu, Jinglu Hu, Jianfeng Yan, Fengzong Lian, and Yuhong Liu. UltraLogic: Enhancing LLM reasoning through large-scale data synthesis and bipolar float reward. *arXiv preprint arXiv:2601.03205*, 2026.

[16] Iman Mirzadeh, Keivan Alizadeh-Vahid, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[17] Qwen Team. Qwen3 technical report, 2025. arXiv:2505.09388.

[18] Mohammad Rezaei, Jens Lehmann, and Sahar Vahdati. LLM reasoning with process rewards for outcome-guided steps. *arXiv preprint arXiv:2604.02341v1*, 2026.

[19] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco J R Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[20] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. arXiv:2402.03300.

[21] Mingyang Song, Mao Zheng, Zheng Li, Wenjie Yang, Xuan Luo, Yue Pan, and Feng Zhang. FastCuRL: Curriculum reinforcement learning with stage-wise context scaling for efficient training R1-like reasoning models. *arXiv preprint arXiv:2503.17287v3*, 2025. Version 3 (v3), revised May 2025.

[22] Anja Surina, Amin Mansouri, Lars Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with LLMs: Evolutionary search meets reinforcement learning. *arXiv preprint arXiv:2504.05108*, 2025.

[23] Sijun Tan, Michael Luo, Justin Wong, Colin Cai, Xiaoxiang Shi, William Yuan Tang, Manan Roongta, Tianjun Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Effective RL scaling of reasoning models via iterative context lengthening. `https://openreview.net/forum?id=I6GzDCne7U`, 2026.

[24] Leitian Tao, Ilia Kulikov, Swarnadeep Saha, Tianlu Wang, Jing Xu, Sharon Li, Jason E Weston, and Ping Yu. Hybrid reinforcement: When reward is sparse, better to be dense. In *The Fourteenth International Conference on Learning Representations*, 2026.

[25] Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023. arXiv:2307.09288.

[26] Jiakang Wang, Runze Liu, Fuzheng Zhang, Xiu Li, Guorui Zhou, and Ling Pan. Stabilizing knowledge, promoting reasoning: Dual-token constraints for RLVR. *arXiv preprint arXiv:2507.15778*, 2025.

[27] Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. *arXiv preprint arXiv:2506.01939*, 2025.

[28] Chenxing Wei, Jiarui Yu, Ying Tiffany He, Hande Dong, Yao Shu, and Fei Yu. ReDit: Reward dithering for improved LLM policy optimization. *arXiv preprint arXiv:2506.18631*, 2025.

[29] Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro von Werra, Arjun Guha, and Lingming Zhang. SelfCodeAlign: Self-alignment for code generation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 62787–62874. Curran Associates, Inc., 2024.

[30] Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026.

[31] Peiran Xu, Zhuohao Li, Xiaoying Xing, Guannan Zhang, Debiao Li, and Kunyu Shi. Hybrid reward normalization for process-supervised non-verifiable agentic tasks. *arXiv preprint arXiv:2509.25598*, 2025.

[32] Jiahao Yu, Zelei Cheng, Xian Wu, and Xinyu Xing. GPO: Learning from critical steps to improve LLM reasoning. In *Advances in Neural Information Processing Systems*, 2025.

[33] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

[34] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, 2025.

[35] Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A Smith. How language model hallucinations can snowball. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 59670–59684. PMLR, 2024.

[36] Tianyu Zheng, Tianshun Xing, Qingshui Gu, Taoran Liang, Xingwei Qu, Xin Zhou, Yizhi Li, Zhoufutu Wen, Chenghua Lin, Wenhao Huang, Qian Liu, Ge Zhang, and Zejun Ma. First return, entropy-eliciting explore. *arXiv preprint arXiv:2507.07017*, 2025.

A Implementation Details

A.1 Initial Sampling and Dynamic Problem Filtering

As outlined in Algorithm 1, TREST begins by sampling a selected problem set $\mathcal{P}$ of target size N (where $N = 100$ in our main experiments) from the full pool $\mathcal{Q}$. To ensure the evolutionary process operates on highly informative trajectories, a problem is deemed valid only if its empirical correctness rate ρ_p is neither 0 (too difficult to seed) nor 1 (too trivial). Furthermore, to concentrate compute on the most valuable learning signals, we discard the easiest remaining valid problems using an online quantile threshold controlled by η. The exact value of η is tailored to the difficulty distribution of the specific dataset and is detailed in the hyperparameter settings (Appendix A.5).

While evaluating and filtering problems sequentially allows screening to stop at exactly N selected problems, it prohibits efficient parallelization. To maximize GPU utilization, we introduce a **dynamic batched screening mechanism**. For each problem in a given batch, we sample $M = 20$ initial solutions to estimate ρ_p. The number of problems processed in batch $t + 1$, denoted as B_{t+1}, is dynamically adjusted based on the historical screening success rate:

$$B_{t+1} = \min\left(B_{\max}, \left\lceil \frac{R_t}{S_t} \right\rceil\right)$$

where $R_t = N - |\mathcal{P}_t|$ is the remaining number of problems required, and S_t is the historical success rate (the ratio of accumulated valid problems to total screened problems up to batch t). To adhere to hardware-level parallel generation limits (e.g., a maximum concurrent batch size of 1000 sequences), we cap the screening batch size at $B_{\max} = 50$ (since $50 \times M = 1000$).

During the screening, this online quantile threshold is continuously re-estimated using the cumulative distribution of ρ_p across all processed batches. This global estimation effectively mitigates the variance inherent in small individual batches. The screening iteratively proceeds until the accumulated valid problems reach or exceed N. Because the final batch is not strictly truncated, the resulting set $\mathcal{P}$ naturally exhibits a negligible batch overshoot (typically fewer than 3 extra problems), which minimally impacts the overall computational budget and training dynamics. Importantly, as established in Section 3.2, all correct solutions generated for the admitted problems during this screening phase are explicitly retained to initialize their respective verified populations $\mathcal{A}_p$, directly serving as the ancestral seeds for the subsequent evolutionary loop without requiring redundant generation.

A.2 Adaptive High-Entropy Token Identification and Inheritance

As established in the main methodology, TREST focuses both its search budget (mutation) and optimization gradients (selective SFT) on logical pivot points, which are proxied by high-entropy token positions. Our objective is to isolate the top γ fraction of the most uncertain tokens within a trajectory (where $\gamma = 0.2$ across all our experiments). However, efficiently identifying these target regions presents two structural challenges: the stylistic noise of terminal tokens and the high variance of trajectory-level entropy distributions.

To isolate logical uncertainty from stylistic freedom, we must address the terminal region of the generation. In autoregressive mathematical reasoning, tokens generated after the final derivation (e.g., concluding remarks, formatting, or paraphrased answers) often exhibit artificially high entropy. This uncertainty stems from stylistic freedom rather than logical complexity. If these tail tokens are included, they pollute the entropy distribution and draw attention away from the actual reasoning phase. Therefore, we explicitly truncate the entropy evaluation. We define L_{ans} as the sequence index where the final mathematical answer first appears, and only tokens in the reasoning prefix ($i \leq L_{\text{ans}}$) are considered for high-entropy selection.

Furthermore, applying a static, global entropy cutoff across all data is suboptimal because absolute entropy scales vary wildly across different problems, prompt contexts, and reasoning styles. To resolve this, we employ an **adaptive trajectory-specific thresholding** mechanism. For a given reasoning prefix, we compute the local token-level entropies H_i. We then determine a computed threshold, τ^{computed}, equivalent to the $(1-\gamma)$-quantile of this specific trajectory's entropy distribution.

While computing a trajectory-specific threshold works well for initial seeds, applying it independently to offspring trajectories introduces an evolutionary risk. If an offspring successfully generates a more

robust continuation, its overall entropy distribution may shift. Recalculating an independent threshold might cause τ^{computed} to increase, inadvertently "washing out" previously identified critical pivot points inherited from its parent. To guarantee the preservation of ancestral uncertainty, we enforce a conservative **monotonic threshold inheritance** rule. For initial seed trajectories, the final threshold is simply $\tau = \tau^{\text{computed}}$. For any offspring trajectory, the final threshold τ_{child} is constrained by its parent:

$$\tau_{\text{child}} = \min\left(\tau_{\text{child}}^{\text{computed}}, \tau_{\text{parent}}\right)$$

This inheritance mechanism ensures that once a token is identified as high-entropy in an ancestral trajectory, it retains this label unless it is overwritten during continuation regeneration.

Ultimately, for each valid trajectory, positions satisfying $H_i \geq \tau$ define the unified high-entropy token set. This discrete masking space explicitly restricts the candidate sampling pool during entropy-guided mutation (Section 3.3.2) and dictates the binary loss mask during policy distillation (Section 3.4), ensuring that the model selectively explores and internalizes only the most critical structural transitions.

A.3 Cross-Problem Batching for High-Throughput Evolution

While Algorithm 1 presents the evolutionary loop sequentially per problem for conceptual clarity, executing it in this naive manner results in severe GPU starvation. Because the number of mutations per generation (K) for a single problem is relatively small, sequential execution fails to saturate the parallel compute capacity of modern accelerators. To maximize inference throughput and hardware utilization, our actual system implementation transposes the nested loops: rather than completing all generations for one problem before moving to the next, we advance the generation step synchronously across all active problems.

This batched evolutionary execution proceeds through four stages, ensuring that the theoretical independence of each problem's search trajectory is perfectly preserved:

Parallel Selection and Mutation. At the start of generation g, the system iterates over all problems that have not yet terminated. For each active problem p, it independently performs Thompson Sampling on its frozen local population $\mathcal{A}_p^{\text{old}}$ and prepares K mutated prefixes. Crucially, the candidate population, statistical tracking variables (S_a, T_a), and high-entropy token thresholds remain strictly isolated per problem.

Global Batched Inference. Instead of invoking the language model sequentially, all prepared mutated prefixes from all active problems are aggregated into a single, global inference batch. This massive flat list of prefixes is dispatched to the vLLM backend [10], which efficiently manages memory and autoregressive generation across thousands of disparate sequence lengths simultaneously.

Result Demultiplexing. Upon generation completion, the newly generated offspring trajectories are demultiplexed (routed back) based on their original problem identifiers. The rule-based verification and population update ($\mathcal{A}_p \leftarrow \mathcal{A}_p \cup \{a_{\text{child}}\}$) occur strictly within each problem's isolated environment, preventing any cross-contamination of reasoning paths between distinct mathematical problems.

Asynchronous Termination. At the end of the generation, the early-stopping condition ($|\mathcal{A}_p| \geq C$) is evaluated independently for each problem. Problems satisfying this maximum solution capacity are retired from the active pool, while the rest proceed to generation $g + 1$. The global evolutionary loop automatically concludes when all problems are retired or the maximum generation limit $G_{\max}$ is reached.

Ultimately, this cross-problem batching strategy is strictly a hardware execution optimization. Because the selection, mutation, and updating mechanisms depend exclusively on the local state of problem p, this vectorized implementation is mathematically equivalent to the sequential logic described in Algorithm 1, differing only by delivering vastly superior wall-clock efficiency.

A.4 Memory Management and Multi-GPU Orchestration

Scaling evolutionary reasoning to 8K-token context lengths introduces severe memory constraints, necessitating explicit memory management and parallel execution strategies alongside the throughput optimizations discussed above.

A.4.1 Memory-Compute Trade-off via On-Demand 1-Token Probing

To execute the proportional resampling defined in the entropy-guided mutation step (Section 3.3.2), the system requires the top-k next-token distribution at the specifically sampled mutation index j. Maintaining this data for an active population that often exceeds 2,000 long-horizon trajectories imposes a severe host memory burden. Naively retaining the raw vLLM inference objects (**Raw Object Caching**) for this population scales to tens of gigabytes, making it impractical for standard compute nodes. Even if we aggressively strip all redundant Python overhead and explicitly store only the essential token IDs and numerical logprobs (**Compact Caching**), the persistent RAM footprint still exceeds multiple gigabytes, posing a significant resource risk in shared computing environments.

To strictly minimize this memory bottleneck without compromising algorithmic integrity, our primary implementation employs a dynamic memory-compute trade-off: **on-demand 1-token probing**. During the initial trajectory generation, the massive vocabulary logit tensors are immediately garbage-collected and discarded the moment the scalar token-level entropy H_i is computed. Later, when a specific high-entropy position j is actively selected for mutation, the vLLM backend performs an isolated, 1-token forward pass conditioned exclusively on the prefix $x_{<j}$. This instantly re-materializes the exact local distribution required for resampling. Given the rapid execution of a 1-token probe, this compute-for-memory trade-off is cost-effective.

From an algorithmic perspective, this probing mechanism is mathematically equivalent to the Compact Caching scheme, as both query the identical conditional distribution $P(v \mid p, x_{<j})$ (modulo negligible floating-point non-determinism at the CUDA scheduling level). Since the 1-token probes are algorithmically irrelevant, they are excluded from the rollout budget constraint established for our main experiments (as detailed in Section 4.2). This is because in the standard theoretical formulation of TREST, the mutation step merely queries a previously computed local distribution. The 1-token probe is an engineering workaround rather than an active evolutionary search step. Indeed, when the framework is explicitly configured to use the Compact Caching scheme via command-line toggles, these probes are bypassed while yielding identical evolutionary dynamics.

A.4.2 Phase-Decoupled VRAM Offloading via Process Isolation

A fundamental system bottleneck in standard on-policy RL algorithms (such as GRPO) is the requirement to frequently interleave sequence generation and parameter updates. This forces both the highly memory-intensive inference engine (maintaining massive KV caches) and the training engine (maintaining gradients and optimizer states) to co-reside in GPU VRAM within a monolithic, long-lived process. Our implementation circumvents this limitation by leveraging the natural structural decoupling of its algorithmic phases: the evolutionary search and the selective SFT update are strictly sequential.

To translate this structural decoupling into actual hardware savings, we implement a strict multi-process isolation strategy. The primary master process operates exclusively on the host CPU. It acts purely as an orchestrator, managing experimental states, generation budgets, and data pipelines, but deliberately avoids establishing any CUDA contexts.

When the framework enters the evolutionary inference phase, the master process spawns short-lived worker subprocesses dedicated solely to batched inference. These workers initialize their localized CUDA environments, load the generation model, execute the batched rollouts across the active problem set, and explicitly serialize the resulting trajectories and their associated metadata (e.g., high-entropy token positions) directly to persistent storage (disk), bypassing volatile inter-process communication. Crucially, upon completion of the inference phase, these worker processes explicitly terminate. When the framework transitions to the SFT phase, a distinct set of training subprocesses is spawned to initialize the model from the previous iteration and ingest the serialized trajectory data directly from disk.

This architecture ensures that inference and training contexts are mutually exclusive in VRAM. More importantly, instead of relying on heuristic framework-level garbage collection (e.g., `empty_cache` calls, which remain highly susceptible to incomplete memory reclamation and residual state leakage caused by lingering Python references), this design delegates memory reclamation to the operating system. When a subprocess dies, the OS ensures complete memory reclamation by clearing the CUDA context, preventing any residual memory footprint. As an intrinsic architectural benefit, routing the

inter-phase state handoff through persistent storage provides robust fault tolerance, allowing the training process to be seamlessly paused and resumed at any iteration boundary.

Consequently, this dynamic "hard offloading" drastically lowers the peak memory footprint. Practically, it enables full-parameter fine-tuning of 1.5B models with a 9,216-token context length on a single 48GB GPU server.

A.4.3 Multi-GPU Implementation

Empirical profiling of our main experiments reveals that the evolutionary rollout phase is heavily inference-bound, consuming approximately 4.6$\times$ more wall-clock time than the subsequent policy distillation (SFT) phase (e.g., $\sim$82 minutes for batched evolution versus $\sim$17 minutes for SFT per iteration). Because generation constitutes the primary throughput bottleneck, we prioritized distributing the evolutionary search across multiple GPUs, while leaving the SFT phase on a single GPU as a functional choice.

To parallelize the evolution phase, we implement problem-level data parallelism. Given W available GPU workers and a target problem quota N, the CPU master process uniformly partitions the quota such that each worker is assigned $\approx N/W$ discrete problems. The master orchestrator then spawns W independent worker subprocesses. To prevent CUDA context interference, each subprocess is explicitly pinned to a dedicated device via hardware visibility masking (e.g., `CUDA_VISIBLE_DEVICES`). Each isolated worker independently executes the batched dynamic screening (Section A.1) and the multi-generational search strictly on its assigned problem subset. Upon termination of all W workers, the master process aggregates the disjoint serialized offspring directories into a unified global dataset.

Conversely, because the selective SFT phase reduces computational overhead by restricting supervision to high-entropy tokens, a single iteration for the 1.5B model completes in less than 20 minutes on a single GPU. While parallelizing the SFT process across GPUs would theoretically yield further speedups, it was not strictly necessary to achieve our required experimental throughput. Therefore, the framework explicitly spawns a single, dedicated training subprocess on the primary GPU. This allocation effectively balances development overhead with hardware utilization, correctly concentrating engineering efforts on the most critical inference bottlenecks.

A.5 Hyperparameter Configurations

Table 4 details the TREST configuration used for the main comparison and main-setting ablations. The specific algorithmic and optimization parameters for the baselines are detailed below.

Iterative RFT: Framework-Aligned Internal Baseline. To rigorously isolate algorithmic contributions from engineering optimizations, we implemented the Iterative RFT baseline using the exact same underlying codebase and inference infrastructure as TREST. By explicitly disabling the evolutionary mutation pipeline and the difficulty-based initial screening mechanism, our framework naturally degrades into a standard Iterative RFT pipeline. This shared implementation eliminates framework-level confounders, ensuring a pure algorithm-to-algorithm comparison. Table 5 details its specific iteration constraints.

GRPO: Representative External RL Baseline. In addition to the internal baseline, we include GRPO to compare against a representative value-model-free reinforcement learning paradigm. For GRPO, we utilized the highly optimized `verl` framework paired with effective empirical hyperparameters to ensure a fair comparison. Table 6 summarizes the optimization and rollout configurations.

A.6 Compute Infrastructure and Budget

To ensure transparency and reproducibility, we explicitly detail the hardware infrastructure and computational budget utilized for the evaluations in this study.

Main Experiments: All primary training runs were conducted using NVIDIA Ampere-architecture GPUs (80GB VRAM each). To provide a standardized comparison of computational overhead, we report the cumulative resource consumption of each method relative to an algorithmic anchor of approximately 0.2M rollouts. The aggregate training statistics to reach the budget-aligned checkpoints are as follows:

Table 4: Hyperparameter configuration for TREST under the main-experiment setting.

Parameter	**Value**
General Setup	
Training Data	DeepScaleR training pool
Base Model	DeepSeek-R1-Distill-Qwen-1.5B
Training Regime	Full Fine-Tuning (FFT)
Initial Sampling Parameters	
Target problem quota (N)	100
Initial samples per problem (M)	20
Initial sampling temperature (T)	1.0
Easy-problem filtering fraction (η)	10%
Evolutionary Search Parameters	
Max solutions per problem (C)	20
Max generations ($G_{\max}$)	6
Mutations per generation (K)	10
Max generation length	$8,192$ tokens
Selective SFT Parameters	
Optimizer	AdamW
Learning Rate	4×10^{-5}
Batch size	4
Gradient accumulation steps	32

Table 5: Hyperparameter configurations for the Iterative RFT baseline.

Parameter	**Value**
Data & Sampling Parameters	
Problems processed per iteration	$2,048$
Samples per problem	8
Sampling temperature	1.0
Max generation length	$8,192$ tokens
Supervised Fine-Tuning (SFT) Parameters	
Optimizer	AdamW
Learning rate	4×10^{-5}
Per-device batch size	2
Gradient accumulation steps	64

- **TREST (Ours):** 33 iterations, totaling 195,170 rollouts and approximately 85.36 GPU hours.
- **Iterative RFT:** 12 iterations, totaling 196,608 rollouts and approximately 102.47 GPU hours.
- **GRPO:** 195 steps, totaling 199,680 rollouts and approximately 161.94 GPU hours.

The cumulative rollout cost of each main-setting ablation is reported in Table 2.

We note that the total computational cost of the entire research project naturally exceeded these reported figures due to preliminary algorithmic exploration, codebase debugging, and initial hyperparameter tuning.

Table 6: Hyperparameter configurations for the GRPO baseline.

Parameter	Value
Algorithm & Optimization Parameters	
Optimizer	AdamW
Actor learning rate	1×10^{-6}
KL penalty coefficient (β)	0.001
KL estimator type	Low-variance KL
Train batch size	128
PPO mini-batch size	64
Total training steps	195
Rollout & Sampling Parameters	
Rollout multiplicity (n)	8
Rollout temperature	0.6
Max generation length	$8,192$ tokens

A.7 Standardized Prompt Templates

To ensure fair comparison and evaluate the intrinsic reasoning capabilities of the models without relying on heavy prompt engineering, we adopt standard, minimal prompt templates for all generation and evaluation phases.

For the main full-parameter experiments and main-setting ablations, we utilize the default prompt format used in the DeepScaleR training setup. This format employs an empty system prompt and appends a standard zero-shot Chain-of-Thought (CoT) trigger to the user query. This identical template is applied across TREST, all baseline methods (e.g., GRPO, Iterative RFT), and all main-setting ablation variants to guarantee strict formatting parity.

```
[System]
(None)

[User]
{problem}
Let's think step by step and output the final answer within \boxed{}.
```

B Prompt for Pairwise Mathematical Insight Evaluation

To ensure the reproducibility and transparency of our qualitative analysis in Section 4.4, we provide the exact prompt used for the DeepSeek-v3.2 [5] evaluator.

The prompt was designed to strictly isolate **Mathematical Insight** from mere computational correctness. It forces the evaluator to act as a "value-neutral structural parser" and mandates a strict binary JSON output (`"1"` or `"2"`). Note that while the prompt enforces a forced-choice verdict for each individual run, our consensus mechanism (a 10-run majority vote with 5 forward and 5 reversed orders) naturally accounts for positional bias and allows for ties (i.e., a $5:5$ split) at the aggregate level.

```
[System]
Relative Pairwise Trajectory Analyzer

You are a highly precise, value-neutral structural parser. I will provide a
mathematical problem along with two correct reasoning trajectories (Trajectory
1 and Trajectory 2).
```

```
Your task is to compare the two trajectories ONLY on the following dimension:

Mathematical Insight vs. Brute-Force Calculation:
Since the problem requires efficient resolution, which trajectory demonstrates
superior mathematical insight by utilizing theorems, abstractions, or clever
mathematical properties to significantly reduce the problem's complexity?

Contrast this with a brute-force state where the model relies on massive step-
by-step arithmetic calculations or rigid case-by-case enumeration. These
exhaustive methods unnecessarily bloat the reasoning path, often causing the
trajectory to enter a meandering state where it loses track of its sub-goals
and the core mathematical objective.

Return your answer in strict JSON with exactly one key:
{"answer": "1" | "2"}

Use:
- "1" if Trajectory 1 relies more on mathematical insight and less on brute-
force.
- "2" if Trajectory 2 relies more on mathematical insight and less on brute-
force.

[User]
Problem:
{problem}

Trajectory 1:
{trajectory_1_think_content}

Trajectory 2:
{trajectory_2_think_content}

Output only the required strict JSON.
```

C Extended Case Studies: Archetypes of Strategic Divergence

In Section 4.4, we showed that structural robustness effectively reflects whether a reasoning trajectory relies on **Mathematical Insight** rather than brute-force calculation. As summarized in the main text (Table 3), robust trajectories actively use central-symmetry reduction, apply pair-sum parameterization, employ systematic frameworks, maintain clear representations, or construct coherent event partitions to bypass problem complexity. In contrast, brittle paths frequently enumerate candidate lines, conduct fragmented factor searches, exhibit semantic entanglement, revert to mechanical execution, or exhibit inconsistent event modeling.

To fully visualize the underlying mechanics of this strategic divergence, we provide the summary of the reasoning traces for the top-5 high-confidence archetypes (identified via a unanimous $10/10$ evaluator consensus and the most extreme robustness gaps among valid problem instances). These extended examples explicitly contrast the execution stability of elegant strategic formulation against the hazards of blind, exhaustive derivation.

DeepScaleR Problem 6657:

> *A parallelogram has vertices* $(10, 45)$*,* $(10, 114)$*,* $(28, 153)$*, and* $(28, 84)$*. A line through the origin divides the parallelogram into two congruent polygons. If the slope of the line is* m/n*, where* m *and* n *are relatively prime positive integers, find* $m + n$*.*

Robust Path Summary: The robust trajectory identifies central symmetry as the governing property. It pairs the opposite vertices $(10, 45)$ and $(28, 153)$, whose midpoint is $(19, 99)$, and verifies that the other opposite pair has the same midpoint. Congruent pieces must have equal area, and for any fixed direction the unique area-bisecting line of a parallelogram passes through its center. A half-turn about

this center maps the parallelogram to itself and exchanges the two sides of every line through the center, so such a line divides the figure into congruent polygons. Because the required line must also pass through the origin, its slope is uniquely

$$\frac{99}{19},$$

and the answer is $99 + 19 = 118$.

Brittle Path Summary: The brittle trajectory first pairs adjacent vertices as though they were opposite, producing incompatible midpoints and a provisional candidate center. It then conflates a side midpoint with the parallelogram's centroid, proposes the slope $129/38$, and explores several unrelated symmetry axes and vertex-based lines. After recomputing the average of all four vertices, it finally finds the actual center $(19, 99)$ and recovers the slope $99/19$.

Commentary: The key insight is not a generic search over cutting lines but the central symmetry of every parallelogram. The robust trajectory uses the vertex order and the shared diagonal midpoint to invoke that invariant directly. The brittle trajectory loses the adjacency structure, so incompatible midpoint calculations trigger a sequence of plausible-looking but geometrically unsupported candidates.

DeepScaleR Problem 8622:

> *Let x, y, and z be positive integers satisfying $(x + y)(y + z) = 2016$ and $(x + y)(z + x) = 1080$. Find the smallest possible value of $x + y + z$.*

Robust Path Summary: The robust trajectory introduces the pair sums

$$A = x + y, \qquad B = y + z, \qquad C = z + x.$$

The two equations become $AB = 2016$ and $AC = 1080$, so $B : C = 28 : 15$. Writing $B = 28k$ and $C = 15k$ gives $A = 72/k$. Recovering

$$x = \frac{A + C - B}{2} = \frac{72/k - 13k}{2}$$

and imposing $x > 0$ immediately restricts the positive integer parameter to $k = 1$ or $k = 2$. The parity condition eliminates $k = 1$, while $k = 2$ gives $(x, y, z) = (5, 31, 25)$ and hence the minimum $x + y + z = 61$. Although the trajectory performs some extra verification afterward, its central derivation remains stable and complete.

Brittle Path Summary: The brittle trajectory notices several useful factors but repeatedly changes notation and restarts the factor search without preserving the relations among the three pair sums. It alternates among divisor enumeration, quadratic reformulations, and partially substituted systems, occasionally mixing which product corresponds to which variable. Only late in the reasoning does it recover the ratio $28 : 15$, parameterize the pair sums consistently, and reach the same solution.

Commentary: The robust path succeeds by choosing variables that match the repeated algebraic structure and then propagating positivity before enumerating cases. This turns a potentially large divisor search into two candidates. The brittle path treats the equations more locally, so useful facts do not accumulate into a global constraint and many unnecessary branches remain open.

DeepScaleR Problem 12777:

> *There are 6 male doctors and 4 female doctors. From them, 3 male doctors and 2 female doctors are selected to provide medical services in 5 different regions. It is specified that male doctor A cannot go to region A. How many different assignment plans are there in total?*

Robust Path Summary: The model quickly recognizes a standard two-stage counting structure. It first selects the personnel, then assigns the chosen five doctors to the five regions. Once this decomposition is in place, the restriction on doctor A is handled cleanly by splitting on whether A is selected and subtracting the forbidden assignments when necessary. The entire solution stays at the level of structural counting principles rather than getting trapped in local assignment details.

Brittle Path Summary: The model spends a long prefix disentangling the wording of the task instead of extracting its latent combinatorial template. It circles around whether region assignment should be treated separately from doctor selection and delays committing to the standard "choose, then permute" structure. Although it eventually reaches the same final count, the reasoning allocates much more effort to local interpretive bookkeeping than to the underlying counting insight.

Commentary: The key divergence is the ability to expose the correct combinatorial abstraction early. The robust path identifies that the problem should be factored into selection, permutation, and exclusion of a small forbidden set, which sharply reduces downstream complexity. The brittle path eventually recovers this structure, but only after wandering through more mechanical case-level reasoning. In this example, mathematical insight appears not as a sophisticated theorem but as the ability to uncover the latent counting architecture before brute-force assignment reasoning takes over.

DeepScaleR Problem 37594:

> *Find the difference between* 1000_7 *and* 666_7 *in base* 7.

Robust Path Summary: The model commits immediately to a low-complexity representation: convert both base-7 numerals into base 10, subtract there, and convert the result back to base 7. Once it sees that $1000_7 = 343_{10}$ and $666_7 = 342_{10}$, the computation collapses to $343 - 342 = 1$, so the answer is simply 1_7. The reasoning remains inside this simplified numerical frame throughout.

Brittle Path Summary: The model also discovers the same simplifying move at first, but it does not maintain it. After correctly obtaining the decimal subtraction, it reopens a more mechanical line of attack and starts reconsidering the problem through direct base-7 subtraction and borrowing. In effect, the trajectory briefly finds the right abstraction, then abandons it and drifts back toward a more local arithmetic procedure.

Commentary: The relevant insight here is not a deep theorem but the ability to choose and preserve the right representation. The robust path treats base conversion as a genuine simplification and stays with it until the end. The brittle path fails to trust that simplification and re-enters a more brute-force, digit-level computation mode after the problem has already been reduced. This example shows that robustness can come from maintaining a simplifying abstraction, not merely from discovering it momentarily.

DeepScaleR Problem 3754:

> *If Alex does not sing on Saturday, then she has a 70% chance of singing on Sunday; however, to rest her voice, she never sings on both days. If Alex has a 50% chance of singing on Sunday, find the probability that she sings on Saturday.*

Robust Path Summary: The robust trajectory ultimately translates the verbal conditions into a coherent event partition. Letting S and U denote singing on Saturday and Sunday, respectively, it recognizes that the exclusivity condition gives $P(U \mid S) = 0$, whereas the stated conditional probability is $P(U \mid \neg S) = 0.7$. The key structural observation is that all probability mass for Sunday singing must therefore come from the no-Saturday branch. The law of total probability gives

$$0.5 = 0 \cdot p + 0.7(1 - p),$$

where $p = P(S)$, yielding $p = 2/7$. The trajectory then verifies the result within the same probabilistic model.

Brittle Path Summary: The brittle trajectory failed to maintain a globally consistent event structure. It repeatedly confuses S with $\neg S$, treats the exclusivity of S and U as though it applied to $\neg S$ and U, and moves between conditional and joint probabilities without preserving their meanings. It eventually reaches the correct equation and obtains $2/7$, but that equation appears only after several mutually incompatible descriptions of the sample space have been introduced and discarded.

Commentary: The mathematical insight lies in identifying the correct event partition: Sunday singing can occur only on the no-Saturday branch, so its marginal probability is obtained by weighting that branch by the corresponding conditional probability. The robust trajectory organizes all conditions around this single structure and can derive and verify the answer within it. The brittle trajectory encounters the same formula only after repeatedly assigning exclusivity to the wrong pair of events;

its locally plausible steps are not grounded in a coherent sample space. The contrast is therefore one of stable structural understanding versus reasoning without a consistent mathematical representation.

D Assets and Licenses

The TREST implementation provided in the supplemental material is released under the Apache License 2.0. The code package includes the license text and copyright notice in `LICENSE` and `NOTICE`, respectively. Third-party assets remain subject to their respective licenses.

All existing datasets, models, and software frameworks utilized in this work are publicly available and properly credited. To comply with reproducibility guidelines, we detail the licenses for the primary assets used in our experiments in Table 7. We strictly respect the terms of use associated with these assets.

Table 7: Summary of primary assets and their corresponding licenses.

Asset Category	**Name**	**License**
Models & Baselines	DeepSeek-R1-Distill-Qwen-1.5B	MIT
	DeepScaleR-1.5B-Preview	MIT
	FastCuRL-V3	MIT
	Qwen3-1.7B	Apache 2.0
Datasets	DeepScaleR Training Pool	MIT
	MATH / MATH-500	MIT
	OlympiadBench	MIT
	SelfCodeAlign-74K	Tongyi Qianwen
Software Frameworks	vLLM	Apache 2.0
	veRL	Apache 2.0

Dataset Note: The DeepScaleR training pool and the evaluation suite released with DeepScaleR are separate collections. We access AIME'24, AMC'23, MATH-500, Minerva, and OlympiadBench through this evaluation setup. MATH-500 is derived from MATH, whereas Omni-MATH and MATH are distinct source collections within the DeepScaleR training pool. These problems originate from their respective competition or institutional sources, such as the Mathematical Association of America (MAA) and MIT OpenCourseWare (OCW). We acknowledge the original copyright holders and restrict our usage strictly to non-commercial academic evaluation.

Code-Generation Data: SelfCodeAlign-74K [29] is distributed under the Tongyi Qianwen License Agreement (`LICENSE-DATA` in its NeurIPS 2024 supplement).

E Iterative RFT under the 100×20 Initial-Sampling Configuration

Motivation and Design. The main comparison aligns the cumulative rollout budget while allowing Iterative RFT and TREST to use their respective data-collection procedures. To examine whether the different numbers of problems and initial samples per problem explain TREST's advantage, we additionally evaluate Iterative RFT with 100 problems per iteration and 20 samples per problem under the same 200,000-rollout cap. These counts match TREST's target problem quota and initial sampling multiplicity. The baseline retains the standard Iterative RFT procedure.

Implementation and Evaluation. To control for engineering differences, this experiment uses the same codebase and inference infrastructure as TREST. Each iteration samples trajectories from the current checkpoint, retains all verifier-correct trajectories, and performs full-parameter SFT on all answer tokens. Difficulty-based screening, evolutionary mutation, and robustness-guided parent selection are disabled. Apart from the sampling counts specified above, the settings follow Table 5, with a per-device batch size of 4 and 32 gradient accumulation steps to match TREST's SFT optimization settings. Training spans 100 iterations and 200,000 rollouts, with a checkpoint saved after every iteration. Evaluation follows the same five-benchmark protocol as the main comparison, with temperature $T = 0.6$, an 8,192-token output limit, and five stochastic evaluations of the selected checkpoint. The run uses the same hardware as the main experiments (Appendix A.6) and consumes approximately 118.74 GPU hours.

Table 8: Iterative RFT with two sampling configurations, compared with TREST. For Iterative RFT, $N \times M$ denotes problems per iteration times samples per problem. Scores are mean Pass@1 percentages over five stochastic evaluations of each selected checkpoint. **Avg.** reports the mean $\pm$ standard deviation of the per-run five-benchmark macro averages; **Cost** is cumulative training rollouts.

Method	AIME'24	AMC'23	MATH500	Minerva	Olympiad	Avg. (↑)	Cost (↓)
Iterative RFT ($2{,}048 \times 8$)	23.33	58.55	80.88	24.71	39.76	45.45±0.68	0.197M
Iterative RFT (100×20)	20.00	59.04	80.32	25.81	40.50	45.13±1.09	0.200M
TREST (Ours)	35.33	64.82	81.12	25.44	42.46	**49.83±1.45**	0.195M

Results. As shown in Table 8, the 100×20 Iterative RFT experiment achieves $45.13 \pm 1.09\%$, compared with $45.45 \pm 0.68\%$ for the $2{,}048 \times 8$ experiment in the main table. TREST retains a 4.70-percentage-point lead under the 100×20 configuration. Thus, matching TREST's initial problem and sample counts does not explain its advantage over Iterative RFT.

F Inverse-Length Parent Selection

Motivation and Design. The negative correlation between trajectory length and robustness in Figure 1c ($r = -0.4394$) raises the question of whether directly favoring shorter trajectories could substitute for robustness-guided selection. We evaluate a control that replaces robustness-guided Thompson sampling with inverse-length parent selection while keeping the remaining TREST pipeline unchanged. For each correct candidate parent i with effective token length L_i, its selection probability is

$$p_i = \frac{1/L_i}{\sum_j 1/L_j},$$

where the sum runs over the correct candidate parents for the same problem and generation. Parent draws are made independently with replacement, so shorter correct trajectories receive proportionally higher selection probabilities. We compare this rule with uniform parent selection, which assigns equal probability to each candidate parent and serves as the no-preference control, as well as with full TREST.

Implementation and Evaluation. All remaining settings follow TREST's main-experiment configuration (Table 4), including the backbone, training data, entropy-guided mutation, and selective SFT. The experiment uses the same cumulative rollout cap of 200,000 and the same evaluation protocol as the main comparison (Section 4.1). The reported checkpoint uses 199,200 training rollouts, shown as 0.199M in Table 9. The TREST and uniform-selection results are the same experiments reported in Table 2.

Table 9: Comparison of parent-selection rules within TREST under the main-experiment setting. Scores are mean Pass@1 percentages over five stochastic evaluations of each selected checkpoint. **Avg.** reports the mean ± standard deviation of the per-run five-benchmark macro averages; **Cost** is cumulative training rollouts.

Method	AIME'24	AMC'23	MATH500	Minerva	Olympiad	Avg. (↑)	Cost (↓)
TREST (Ours)	35.33	64.82	81.12	25.44	42.46	**49.83±1.45**	0.195M
TREST (uniform selection)	26.67	60.72	80.40	26.47	42.19	47.29±2.02	0.198M
TREST (inverse-length selection)	20.00	55.90	76.52	18.01	37.39	41.57±0.68	0.199M

Results. Inverse-length selection obtains $41.57 \pm 0.68\%$, compared with $47.29 \pm 2.02\%$ for uniform selection and $49.83 \pm 1.45\%$ for full TREST. Its mean is 5.72 percentage points below the no-preference uniform control and lower on all five benchmarks. Thus, the tested inverse-length preference lowers performance in this setting and does not support shorter reasoning as the primary explanation for TREST's gain.

This outcome is consistent with length being an ambiguous proxy: shortness can arise from useful structural simplification or from omitted intermediate reasoning. A length-only rule treats these cases alike, whereas offspring correctness tests whether locally perturbed continuations remain correct.

G Length-Matched Insight Analysis and Problem-Type Stratification

Motivation and Design. We conduct a length-matched analysis to examine whether the insight–robustness association persists among nearly equal-length trajectories. This is a stringent residual test, not a conventional confounder control: mathematical insight can enable structural simplification that avoids lengthy brute-force calculation and shortens reasoning, so matching length also removes a visible manifestation of insight.

Starting from the same 200-problem pool used in Figure 1c, we consider pairs of correct trajectories for the same problem whose token lengths differ by at most 5%, offspring correctness rates differ by at least 10 percentage points, and posterior probability of the robustness ordering is at least 95%. For each of the 153 eligible problems, we uniformly sample one qualifying pair. We follow the Mathematical Insight criterion and voting procedure in Section 4.4, using the same judge and prompt (Appendix B) without revealing the trajectories' robustness identities. Each pair receives five judgments in each presentation order.

Length-Matched Results. The analysis produces 43 ties among 153 pairs (28.1%), compared with 4 ties among 102 pairs (3.9%) in the original analysis. This higher tie rate is consistent with insight becoming harder to distinguish after matching length. Among the 110 non-tied pairs, the more insightful trajectory is also more robust in 66 cases, versus 44 in the opposite direction, yielding 60.0% concordance (exact two-sided binomial $p = 0.045$). Thus, a positive association remains among nearly equal-length trajectories. The original 79.4% ($81/102$, including ties in the denominator) and length-matched 60.0% ($66/110$, excluding ties) answer complementary questions under different pair-selection rules: overall association among robustness-gap-selected pairs versus residual association after length matching.

Problem-Type Stratification. Problem type is held fixed by construction because each pair contains two trajectories for the same problem. As an additional consistency check, we categorize the 110 non-tied problems by primary solution method (Table 10). Concordance denotes agreement between the judge's insight preference and the robustness ordering, not answer accuracy. It exceeds 50% in four categories; the Calculus & Analysis subset contains only three pairs and is inconclusive.

Table 10: Problem-type breakdown of the length-matched insight analysis. Counts and concordance rates use non-tied pairs only.

Problem Type	**Concordant / Non-tied**	**Concordance**
Algebra	22/36	61.1%
Geometry	17/29	58.6%
Number Theory	11/16	68.8%
Counting & Probability	15/26	57.7%
Calculus & Analysis	1/3	33.3%
Overall	**66/110**	**60.0%**

H Supplementary Code-Generation Evaluation

Setup and Design. We use Qwen3-1.7B [17] with its native thinking template on Python tasks from the SelfCodeAlign dataset [29]. After filtering invalid tasks, we use 55,202 training tasks and a fixed, disjoint 1,500-task held-out evaluation set. Correctness requires passing all frozen executable tests, and offspring correctness provides the robustness feedback. We compare TREST with its ablation without robustness-guided parent selection, which selects parents uniformly while retaining the remaining components.

Implementation. Both conditions target 100 selected problems per iteration and draw 20 initial samples per problem. We perform one epoch of full-parameter selective SFT per iteration with a learning rate of 4×10^{-5} and an effective batch size of 128. The output limit is 32,768 tokens, with sampling temperatures of 1.1 for training and 0.6 for evaluation. We conduct the experiments using eight NVIDIA Hopper-architecture GPUs.

Budget and Evaluation. Both conditions use the same 25,000-training-rollout budget, counting initial samples and mutation-generated offspring. For each condition, we select the checkpoint with the highest single-run held-out Pass@1 and perform five stochastic evaluations of the selected checkpoint.

Table 11: Supplementary code-generation results. Mean $\pm$ sample standard deviation over five stochastic evaluations of each fixed checkpoint on the same 1,500-task held-out set. Both trained variants use the same 25K-rollout budget. Differences are computed before rounding.

Method	**Held-out Pass@1 (%)**
Qwen3-1.7B (Base Model)	39.85 ± 0.52
TREST	$\mathbf{51.91 \pm 0.45}$
– w/o Robustness-Guided Parent Selection	50.53 ± 0.46

Results and Analysis. Both trained variants improve over the initial model, while robustness-guided parent selection provides an additional 1.37-percentage-point gain over uniform selection (Table 11). This provides supplementary evidence that using robustness feedback to allocate mutation rollouts can also benefit training in this code-generation setting.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the proposal of the TREST framework and its specific benefits (superior reasoning performance under aligned budgets and favoring mathematical insight over mechanical calculation). These claims are directly supported by the experimental results in Section 4, including the main performance comparison (Table 1), ablation studies (Table 2), and qualitative analysis (Figure 4 and Table 3). The evaluation centers on mathematical reasoning, supplemented by code generation, and uses 1.5B and 1.7B parameter models; Section 6 acknowledges that broader cross-domain validation and large-model scaling remain future work.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a dedicated "Limitations and Future Work" section (Section 6). The authors explicitly acknowledge two main limitations: first, the empirical validation focuses primarily on mathematical reasoning, with supplementary code-generation evidence, while broader cross-domain validation remains future work; second, the experiments were conducted on 1.5B and 1.7B parameter architectures, and the scaling behavior of the framework on massive language models remains unverified.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper focuses on proposing an empirical and methodological framework (TREST) rather than theoretical contributions. While it uses mathematical formulations to describe its algorithmic components (such as the offspring correctness rate, token-level entropy, and Thompson sampling mechanics), it does not present any formal theoretical results, theorems, lemmas, or proofs.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides the information needed to reproduce the reported experiments. Section 3 and Algorithm 1 define TREST, Section 4.1 specifies the data, base model, evaluation protocol, and budget for the main comparison, and Section 4.3 evaluates controlled component variants under that same main-experiment setting. Appendices A.1–A.6 provide implementation, hyperparameter, and compute details, while Appendix B provides the qualitative-evaluator prompt. Appendix H describes the supplementary code-generation setup and evaluation protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The complete source code for our proposed TREST framework is provided in the supplemental material. This includes all necessary Python implementation files, environment configurations, and shell scripts (`.sh`) to initiate the training and evaluation pipelines. We use the publicly available DeepScaleR training pool and the separate five-benchmark evaluation suite released with DeepScaleR. Detailed instructions on how to access, download, and prepare these resources are included in the `README.md` file to ensure faithful reproduction of our results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 specifies the shared training data, test benchmarks, base model, budget, and evaluation protocol for the main comparison, and Section 4.3 holds this main

setting fixed while changing only the indicated mechanism(s). Appendix A.5 tabulates the corresponding optimization settings, and Appendix A.6 reports compute information. Appendix H describes the supplementary code-generation setup and evaluation protocol.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Tables 1 and 2 report mean Pass@1 over five stochastic evaluation runs of each selected checkpoint; the reported $\pm$ is the standard deviation of the five per-run, equally weighted five-benchmark macro averages and therefore characterizes stochastic autoregressive-evaluation variability rather than training-seed variability. Table 11 reports mean Pass@1 and sample standard deviation over five stochastic evaluations of each selected checkpoint. The qualitative analyses report their own sample counts, voting procedures, and statistical tests.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A.6 reports the hardware and GPU-hour totals for the main comparison, while Table 2 reports the cumulative rollout cost of each main-setting ablation. Appendix A.4 details the host/GPU memory-management strategies. Appendix H reports the hardware and rollout budget for the supplementary code-generation experiment. The total research compute was higher because it also included preliminary exploration and debugging.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This research focuses fundamentally on improving the mathematical reasoning capabilities and algorithmic efficiency of large language models. The experiments use publicly available mathematical and code-generation data, together with open-weight base models. The work does not involve human subjects, personally identifiable information (PII), or sensitive data, and it does not introduce any foreseeable ethical risks or direct societal harms.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper focuses entirely on foundational algorithmic research—specifically, developing a post-training framework to improve the mathematical reasoning performance of language models. As noted in the guidelines for foundational research, our general algorithm is not tied to particular downstream deployments, nor does it present a direct path to negative applications (such as disinformation, privacy violations, or surveillance). Therefore, a dedicated discussion of broader societal impacts is not applicable to the scope of this algorithmic contribution.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not release models or datasets with a high risk for misuse. The experiments use publicly available mathematical and code-generation tasks and existing open-weight base models. Generated programs are evaluated in an isolated execution sandbox.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly credit the creators of all primary assets (models, datasets, and software) via formal citations. For evaluation sets accessed through the suite released with DeepScaleR (e.g., AIME'24, AMC'23, and Minerva), Appendix D provides a dedicated note that explicitly acknowledges their institutional origins (e.g., MAA and MIT OCW). This section tabulates the licenses for all primary distribution sources and confirms our adherence to established academic research practices and terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Our primary new asset is the implementation code for the TREST framework. We provide comprehensive documentation within the supplemental material, including a detailed `README.md` that specifies installation procedures, dependency versions, and usage instructions. The code package includes `LICENSE` (Apache License 2.0) and `NOTICE` (copyright attribution). Appendix D documents the TREST code license separately from third-party asset licenses and institutional origins.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not involve crowdsourcing or research with human subjects. Mathematical and code-generation outputs are evaluated automatically.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This paper does not involve research with human subjects; therefore, IRB approval and risk disclosure to participants are not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.

- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Large language models are core to TREST: they generate reasoning trajectories and serve as the base models in the evolutionary self-training pipeline. DeepSeek-R1-Distill-Qwen-1.5B is used for the main comparison and main-setting ablations (Sections 4.1 and 4.3), and Qwen3-1.7B is used for the supplementary code-generation experiment (Appendix H). DeepSeek-v3.2 serves as the mathematical-insight evaluator (Section 4.4; prompt in Appendix B). We do not use LLMs for original idea generation or core theorem proving.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.