Extended Abstract

Motivation As large language models (LLMs) grow in scale, the returns from simply increasing model size are diminishing, while computational costs from doing so continue to rise. This motivates a shift to a more effective use of compute at test time, where even smaller models can achieve competitive performance by reasoning more deeply (Snell et al., 2024). This is particularly relevant for mathematical reasoning tasks, like the Countdown problem, where multi-step problem solving often exceeds the capabilities of purely feedforward generation. Our work explores whether test-time inference strategies can enable smaller models to achieve better performance on such structured reasoning tasks without requiring larger pretrained models.

Method We implement and compare five distinct test-time inference strategies for mathematical reasoning, each designed to address different failure modes in arithmetic problem solving:

- **Parse Multiple**: An enhanced single generation approach that extracts and validates multiple candidate expressions from model output, selecting the first mathematically correct solution.
- **Re-ranking**: An iterative generation strategy that produces up to 5 candidate solutions sequentially, terminating early when a correct solution is found.
- **Self-Consistency**: Generates 3 diverse solutions using higher temperature sampling, then applies majority voting among mathematically correct expressions to select the most frequent solution.
- **Confidence-Weighted Selection**: Generates candidates at different temperature settings $(0.4, 0.8)$ and selects based on both correctness and generation confidence, prioritizing solutions from lower-temperature (more confident) generations.
- **Verifier-Based Selection**: Uses an alternate scoring function (i.e. different weights to the default ground truth verifier) to do re-ranking.

Implementation Our implementation consists of three main stages. First, we finetuned a Qwen2.5-0.5B model on the cognitive behaviors dataset (`Asap7772/cog_behav_all_strategies`) dataset using LoRA adaptation (rank=8, α=32) with AdamW optimizer (lr = 5e-5) for 5 epochs. Then we applied online policy gradient training (RLOO) using a rule-based correctness reward, where policy updates are computed using the reinforce gradient with baseline subtraction across k samples. Third, at test time we explored each of the five different test-time inference methods, generating an appropriate number of samples depending on the method.

Results Our initial experiments, based solely on the SFT model, showed there was an improvement to incorporating test-time inference into the model pipeline. When compared on a select few randomly generated samples, each of re-ranking, self-consistency, and confidence-weighted selection (with scores of 0.43, 0.47, and 0.41 respectively) showed slight improvements over the baseline SFT model score of 0.37. We noticed the largest improvement with the multiple parsing approach, which saw a greater than 26% improvement over the baseline staff SFT model. As a result of some compute time constraints, the RLOO implementation saw just a 25% improvement over the baseline, largely because we had to limit the amount of training data and the number of candidate responses generated.

Discussion The initial test-time methods (re-ranking, self-consistency and confidence-weighted selection) show reasonably similar performance, which aligns with expectations since both approaches provide limited additional information for re-ranking when a ground truth verifier is available. The marginal differences in success are likely a result of noise from testing on a small sample size. These methods also conceptually capture similar information to RLOO by generating multiple candidate solutions and selecting outputs based on that ensemble, making the use of RL to update the model a natural extension of this multi-candidate approach. The Parse Multiple introduces an alternate perspective, by extracting and validating multiple candidate expressions from a single model output, selecting the first mathematically correct solution—this provides a middle ground between single-generation efficiency and multi-generation robustness by exploiting the model's tendency to generate multiple solution attempts within a single response. Future work should explore revision-based and tree-search methods, which offer distinct advantages over current approaches: revision methods can backtrack from dead ends and avoid repetitive loops common in mathematical reasoning, while tree-search with multi-proposal generation improves exploration of solution spaces, particularly for problems where single attempts typically fail. These approaches fundamentally alter the generation distribution through iterative refinement, contrasting with our current methods that select from a fixed set of candidates.

Default Project: Supervised Fine Tuning of Qwen2.5-0.5B on the Countdown Dataset

Anushree Aggarwal
Department of Computer Science
Stanford University
anushre@stanford.edu

Saif Moolji
ICME
Stanford University
smoolji@stanford.edu

Bryce Tiglon
Department of Computer Science
Stanford University
btiglon@stanford.edu

Abstract

As returns from scaling large language models diminish while computational costs rise, test-time inference strategies offer a promising alternative for improving reasoning performance without requiring larger pretrained models. We investigate whether smaller models can achieve competitive performance on mathematical reasoning tasks through enhanced test-time computation, focusing on the structured arithmetic problem-solving domain of the Countdown problem. We implement and evaluate five distinct test-time inference strategies: Parse Multiple (extracting multiple candidate expressions from single outputs), Re-ranking (sequential generation with early termination), Self-Consistency (majority voting across diverse solutions), Confidence-Weighted Selection (temperature-based ranking), and Verifier-Based Selection (alternate scoring functions). Our approach combines supervised fine-tuning of a Qwen2.5-0.5B model on cognitive behaviors data with online policy gradient training using RLOO and rule-based correctness rewards. Experimental results demonstrate marginal improvements over baseline single-generation approaches, with Parse Multiple achieving the largest gain of 25% over the supervised fine-tuned model baseline, while the other methods showed smaller improvements. The similar performance across re-ranking, self-consistency, and confidence-weighted approaches suggests these methods capture comparable information to each other in our setting where a ground-truth exists. Our findings indicate that test-time computation can enhance reasoning performance in smaller models, challenging the focus on parameter scaling alone and suggesting that computational resources may be more effectively allocated during inference rather than pre-training for mathematical reasoning tasks. Future work will explore revision-based and tree-search methods that can backtrack from dead ends and systematically explore solution spaces through iterative refinement, potentially offering superior performance by fundamentally altering the generation distribution rather than selecting from fixed candidate sets.

1 Introduction

LLMs have advanced natural language processing with great skills in various reasoning tasks, including complex mathematical problem solving and structured instruction following tasks. Traditionally, these improvements have been driven primarily by scaling model parameters to billions or even trillions, resulting in substantial computational overheads and diminishing returns, especially for nuanced, structured reasoning tasks that inherently requires precise, multi step logical inference (Snell et al., 2024).

In order to address these computational inefficiencies, recent research has increasingly explored RL based fine-tuning methods as alternatives to parameter scaling. Prominent RL methods, including Direct Preference Optimization (DPO; Rafailov et al., 2023) and Reinforce Leave One Out (RLOO; Ahmadian et al., 2024), have shown great potential in refining LLM capabilities using human preference or automated correctness based feedback. However, these RL fine-tuning approaches have some limitations, such as high variance in gradient estimates, repetitive reasoning errors, and insufficient exploration of the solution space, particularly when deployed on mathematically structured datasets such as Countdown (Gandhi et al., 2024).

In parallel, distinct research has found that test time inference optimization methods shift computational effort from training to inference, and systematically explore and refine generated outputs. Further, verification based inference methods, such as Tree of Thoughts (Yao et al., 2023) and iterative refinement approaches like Self Refine (Madaan et al., 2023), also help improve. Specifically, verification-based methods excel at structured exploration of complex solutions, whereas iterative refinement methods effectively correct individual solution paths but can easily be lost in local optima or repetitive cycles (Snell et al., 2024).

Motivated by these aforementioned complementary strengths and weaknesses, we examine a unique comparison of various test-time strategies to see how they perform in numerical reasoning tasks. Our approach explicitly targets the known limitations of RL fine-tuning strategies by systematically broadening exploration through multiple candidate solutions and efficiently guiding solution search using verifier-based scoring.

In this paper, we presented a rigorous implementation of our proposed approach, evaluated comprehensively on the Countdown mathematical reasoning dataset. Our primary contributions include -

- Implementation of the RLOO algorithm, finetuning applied specifically to enhance the mathematical reasoning capabilities of the Qwen2.5-0.5B model.
- Multi-proposal generation, and verifier-based pruning for optimizing inference time computation.
- Quantitative and qualitative evaluations for improvements in correctness and reasoning compared to traditional RL fine-tuning baselines.

Ultimately, our results show that these optimized test-time inference strategies can significantly enhance the performance of smaller LLMs, thereby providing a resource-efficient alternative to the traditional, computationally expensive scaling paradigm.

2 Related Work

Recent work has laid the groundwork for understanding and leveraging test-time computation in LLMs. As discussed above, (Snell et al., 2024) establish a unifying framework that categorizes test-time strategies into two primary classes: (1) proposal distribution modifications, which alter the model's output distribution by revising the input prompt or conditioning sequence; and (2) verification-based methods, which sample multiple candidate outputs and apply post-hoc validation to select or modify the best response. These approaches differ not only in mechanism but also in which types of tasks they tend to benefit. Snell et al. show that while proposal modifications (e.g. iterative prompting or refinement) often yield strong performance on simpler problems, more challenging tasks, such as those requiring deep multi-step reasoning, benefit significantly more from verification-based strategies like tree search and parallel evaluation.

Work in both categories has advanced rapidly. For example, Self-Refine (Madaan et al., 2023) and related methods (Qu et al., 2024) introduce strategies where the model revisits and revises its own outputs through sequential prompting. These iterative methods are effective when errors are local and correctable. On the verification side, Tree of Thoughts (Yao et al., 2023) exemplifies structured search guided by internal self-evaluation, while Snell et al. extend this with process reward models

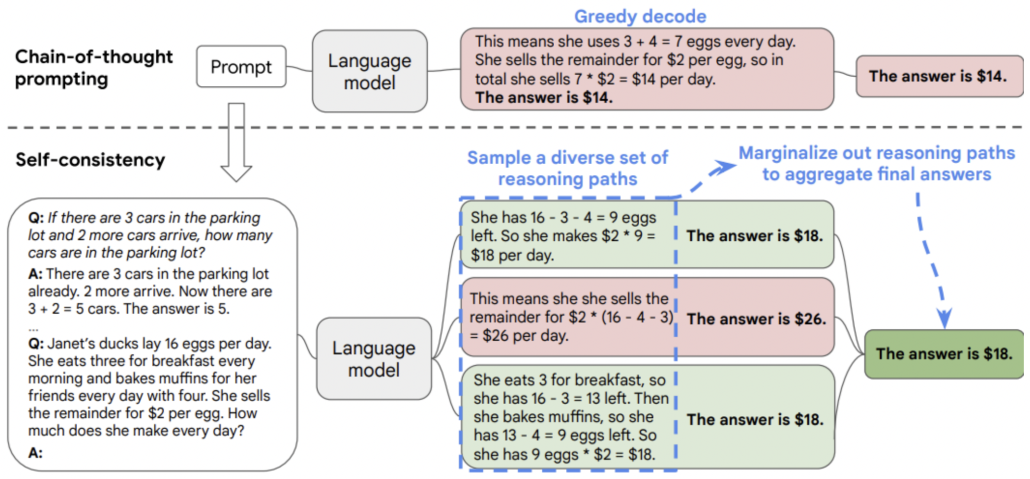
Chain-of-thought prompting
Prompt
Language model
Greedy decode
This means she uses 3 + 4 = 7 eggs every day. She sells the remainder for $2 per egg, so in total she sells 7 * $2 = $14 per day. The answer is $14.
The answer is $14.
Self-consistency
Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
A: There are 3 cars in the parking lot already. 2 more arrive. Now there are 3 + 2 = 5 cars. The answer is 5.
...
Q: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder for $2 per egg. How much does she make every day?
A:
Language model
Sample a diverse set of reasoning paths
She has 16 - 3 - 4 = 9 eggs left. So she makes $2 * 9 = $18 per day.
The answer is $18.
This means she she sells the remainder for $2 * (16 - 4 - 3) = $26 per day.
The answer is $26.
She eats 3 for breakfast, so she has 16 - 3 = 13 left. Then she bakes muffins, so she has 13 - 4 = 9 eggs left. So she has 9 eggs * $2 = $18.
The answer is $18.
Marginalize out reasoning paths to aggregate final answers
The answer is $18.

Figure 1: Illustration of self-consistency decoding:multiple reasoning paths are sampled, and aggregate final answers by choosing the most consistent answer in the final answer set

(PRMs) that guide search more selectively. However, all these methods risk over-optimization when applied to simpler tasks, potentially hurting generalization.

In light of these findings, our work focuses on verification-based strategies during test time, especially for structured mathematical reasoning tasks like the Countdown problem. These problems often involve long chains of arithmetic steps where a single mistake can invalidate the entire solution. Proposal modifications can falter in such cases due to limited recovery mechanisms or the inability to explore meaningfully different solution trajectories. Verification-based methods, in contrast, allow the model to pursue multiple solution paths in parallel and use correctness signals to filter out invalid ones, providing a more robust mechanism for problem solving.

While other methods have explored these various types of verification methods individually, our work aims to compare their effectiveness in complicated numerical reasoning tasks that require iterative problem solving. Altogether, our work builds on and refines this growing line of research, demonstrating that verification-based test-time methods can improve small model performance on arithmetic tasks without additional pretraining—particularly in settings where precision and structure are crucial, and offers some insight about what kinds of methods work best.

3 Method

3.1 Problem Formulation

We examine the validity of our methods via numerical reasoning tasks. Specifically, we test model performance on the game Countdown, where, given a set of N numbers and a target number, the model must determine the sequence of arithmetic operations $(+, -, *, /)$ that transform the set of provided numbers into the target number. Each number can only be used once, and the task evaluates the model's ability to plan, decompose, and execute multi-step mathematical reasoning systematically, as well as format the answer in a syntactically valid and verifiable mathematical expression.

3.2 Supervised Fine-Tuning (SFT)

We performed supervised fine-tuning of the Qwen2.5-0.5B pretrained language model (Qwen Team, 2024) on the cognitive behaviors dataset dataset (Singh, 2024). To efficiently adapt parameters while preserving pretrained knowledge, we employ Low-Rank Adaptation (LoRA).

For any weight matrix $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ in the model, LoRA applies a low-rank decomposition:

$$W = W_0 + \frac{\alpha}{r} BA$$

where:

$$A \in \mathbb{R}^{d_{\text{in}} \times r} \quad \text{(down-projection matrix)} \tag{1}$$

$$B \in \mathbb{R}^{r \times d_{\text{out}}} \quad \text{(up-projection matrix)} \tag{2}$$

$$BA \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}} \quad \text{(matching } W_0 \text{ dimensions)} \tag{3}$$

The LoRA hyperparameters were configured as:

- Rank: $r = 8$ (balancing expressiveness and efficiency)
- LoRA scaling factor: $\alpha = 32$ (controlling adaptation strength)
- Dropout rate: $p = 0.1$ (preventing overfitting in adaptation layers)
- Target modules: attention projection layers (`q_proj`, `v_proj`) and feed-forward layer (`c_attn`)

Matrix A is initialized with Gaussian random values $\mathcal{N}(0, \sigma^2)$, while B is zero-initialized to ensure $W = W_0$ at training start. The scaling factor $\frac{\alpha}{r} = 4$ controls the magnitude of adaptations relative to pretrained weights.

Training minimized the standard causal language modeling objective:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log P_\theta(y_t^{(n)} | x^{(n)}, y_{<t}^{(n)})$$

where N is the number of training examples, T_n is the sequence length for example n, and tokens corresponding to input prompts $x^{(n)}$ are masked with label -100 to exclude them from loss computation.

We used the AdamW optimizer with learning rate 5×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0.01, batch size 4, trained for 5 epochs with Automatic Mixed Precision (AMP) for computational efficiency. Gradient clipping with maximum norm 1.0 was applied for training stability.

3.3 Reinforce Leave One Out (RLOO)

Following supervised fine-tuning, we applied Reinforce Leave One Out, a variance-reduced policy gradient algorithm that addresses the high variance issues in standard REINFORCE. The RLOO gradient estimator is defined as:

$$\nabla_\theta J(\theta) = \frac{1}{k} \sum_{i=1}^{k} \left[R(y^{(i)}, x) - b(x, y^{(i)}) \right] \nabla_\theta \log \pi_\theta(y^{(i)} | x)$$

where the leave-one-out baseline is:

$$b(x, y^{(i)}) = \frac{1}{k-1} \sum_{j=1, j \neq i}^{k} R(y^{(j)}, x)$$

Key components include:

- $k = 4$ independently sampled candidate completions $y^{(i)} \sim \pi_\theta(\cdot | x)$ from the current policy
- Leave-one-out baseline $b(x, y^{(i)})$ computed from remaining $k - 1$ samples, providing an unbiased but correlated control variate
- Temperature $T = 0.8$ for sampling diversity during RLOO training

The reward function $R(y, x)$ employs a weighted combination of format validity and arithmetic correctness:

$$R(y, x) = 0.1 \cdot \mathbb{I}(\text{format-valid}(y)) + 1 \cdot \mathbb{I}(\text{eval}(y) = t)$$

where:

- $\mathbb{I}(\cdot)$ denotes the indicator function returning 1 if true, 0 otherwise

- format-valid(y) verifies syntactic correctness of the mathematical expression
- eval$(y) = t$ checks if the expression numerically evaluates to target t
- Weights (0.1, 1) prioritize correctness while rewarding proper formatting

RLOO training used learning rate 1×10^{-6}, batch size 8, for 500 optimization steps with KL divergence penalty coefficient $\beta = 0.01$ to prevent policy drift from the SFT initialization.

3.4 Test-Time Inference Strategies

We implemented five inference strategies to improve reasoning robustness and accuracy during evaluation:

1. Baseline Single-Pass Generation Standard autoregressive generation with a single completion per prompt:

$$y \sim \pi_\theta(y|x; T = 0.6, \text{top-}k = 20, \text{top-}p = 0.95)$$

where T is the temperature controlling sampling randomness, top-k restricts sampling to the k most likely tokens, and top-p enables nucleus sampling, which selects from the smallest possible set of tokens whose cumulative probability exceeds p..

2. Confidence-Weighted Selection Generate $n = 5$ candidate solutions and select the highest probability completion:

$$y^* = \arg\max_{y_i} P_\theta(y_i|x) = \arg\max_{y_i} \prod_{t=1}^{|y_i|} P_\theta(y_{i,t}|x, y_{i,<t})$$

This strategy implements a temperature-based confidence ranking system, operating under the hypothesis that lower temperature generations represent higher-confidence solutions. The multi-temperature sampling approach uses levels of 0.4 and 0.8, representing high-confidence and exploratory sampling respectively, with confidence ordering that prioritizes lower temperature solutions when multiple correct solutions exist. The underlying confidence hypothesis assumes a temperature-confidence correlation where lower temperatures produce more confident, deterministic outputs, suggesting a quality-confidence relationship where higher confidence solutions are more likely to be correct. This approach aims to maintain an exploration-exploitation balance by preserving exploration through higher temperature while prioritizing confident solutions, with fallback to the highest-confidence generation when no correct solutions are found.

3. Verifier-Based Reranking Sample $n = 5$ completions and rerank using the trained correctness verifier:

$$y^* = \arg\max_{y_i} R(y_i, x), \quad y_i \sim \pi_\theta(\cdot|x)$$

where $R(y_i, x)$ is the reward function defined above. This leverages learned verification signals to identify optimal solutions.

4. Self-Consistency Decoding From $n = 3$ sampled completions, select the most frequent numerically correct answer:

$$y^* = \arg\max_{y \in \{y_i\}} |\{y_j : \text{eval}(y_j) = \text{eval}(y)\}|$$

This method exploits the intuition that correct reasoning paths should converge to the same numerical result across independent generations. We use a higher temperature (0.8) to increase solution diversity.

5. Alternate Verifier Based Scoring This method operates as an alternative to the ground truth verifier we start with. The scoring system assigns points based on correctness (10 points for solutions that evaluate to the target), expression validity (2 points for non-empty expressions), and number usage compliance (1 point for expressions that incorporate the provided numbers). It acts very similarly to our initial reward system, except it weights non-empty/properly formatted expressions slightly higher, and gives an additional, separate reward for including the correct numbers (and only once). In this way, it was meant to be slightly more expressive and be able to distinguish between a larger number of responses.

The candidate generation and evaluation process also samples at temperatures of 0.5 and 0.9 to capture different confidence levels, and employs a similar tie-breaker as confidence-weighted selection, preferring the lower temperature generations.

6. Multi-Solution Parsing This method recognizes that there may be instances where the model generates a correct solution, but mischaracterizes it in its thinking process as incorrect. As a result, we

1. Parse and numerically verify all $n = 5$ generated expressions
2. Identify the first correct solution $C = \{y_i : \text{eval}(y_i) = t\}$

If $C = \emptyset$ (no correct solutions), we use the raw generation.

4 Experimental Setup

We used the pre-trained Qwen2.5-0.5B model, a 0.5B parameter decoder only transformer. For evaluation, we use the Countdown dataset, which consists of arithmetic reasoning problems where the model must compute a valid expression using six given numbers and reach a target number between 100 and 999 using the operations $\{+, -, \times, \div\}$. Each input is presented as a natural language prompt, tokenized using the Qwen tokenizer with a maximum input length of 256 and a maximum output length of 1024. Input tokens are right-padded and masked during loss computation using label -100. We finetuned the model using the AdamW optimizer with a learning rate of 5×10^{-5}, weight decay of 0.01, batch size of 4, and automatic mixed precision. LoRA is applied with rank $r = 8$, scaling factor $\alpha = 32$, dropout 0.1, and injected into the `q_proj`, `v_proj`, and `c_attn` layers. We train for 3 epochs. Following supervised fine-tuning, we perform RLOO optimization for 1 epoch on the Countdown Tasks dataset (Jiayi-Pan/Countdown-Tasks-3to4) using the Adam optimizer and a learning rate of 1×10^{-5}, generating 2 samples per prompt and computing the gradient with the leave-one-out baseline. We have a custom pre-processing function to transform the information provided in the dataset (i.e. the list of numbers and target value) into the appropriate prompt style found in the cognitive behaviors dataset our SFT model was trained on. The reward function combines a rule-based verifier's assessment of format validity and arithmetic correctness:

$$R = 0.1 \cdot \mathbb{I}_{\text{format}} + 1 \cdot \mathbb{I}_{\text{correctness}}$$

For test-time inference, we generate 5 completions per prompt using a nucleus sampling strategy with temperature 0.6, top-k 20, and top-p 0.95. We evaluate five strategies: (1) single-pass SFT (no reranking), (2) confidence-weighted voting based on log-probabilities, (3) reranking using the verifier reward, (4) self-consistency via majority numeric answer, and (5) parsing multiple correct completions and selecting the most confident. The sole caveat here is that the multiple parsing method didn't require multiple generations, as explained before.

5 Results

Strategy	Accuracy	Improvement	Citation
Baseline SFT (single-pass)	0.37	—	—
Confidence-Weighted Voting	0.41	+10.8%	Wang et al. (2023)
Reranking (oracle)	0.43	+16.2%	Snell et al. (2024)
Alternate Verifier	0.45	+21.6%	Zhang et al. (2025)
Self-Consistency	0.47	+27.0%	Wang et al. (2023)

Table 1: Countdown accuracies and relative improvements over baseline SFT.

5.1 Baseline Test-Time Compute Strategies

As shown in the above table, Countdown accuracies improve over the baseline SFT (single pass) accuracy of 0.37 with increasingly advanced inference strategies. The "Strategy" column lists each method

Table 2: Performance Comparison of RLOO and SFT with multiple answer parsing

Method	Speedup Relative to Baseline
Baseline SFT	-
RLOO + Test-Time Compute	0.3196
SFT + Test-Time Compute	0.262x

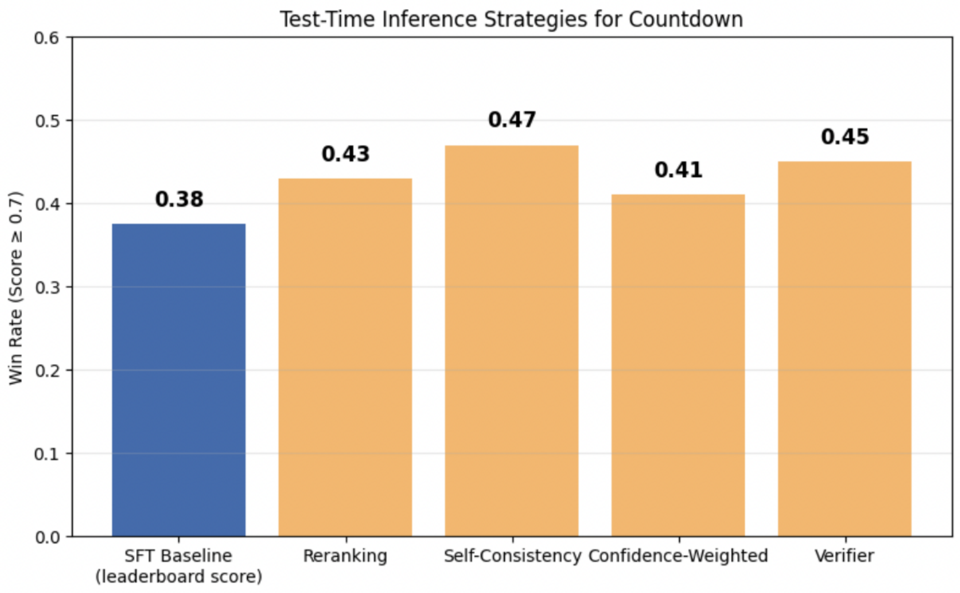

Figure 2: Quantitative comparison of model outputs and scores for different test-time inference strategies on Countdown.

(Baseline SFT, Confidence-Weighted Voting, Reranking, Alternate Verifier, and Self-Consistency), “Accuracy” reports their respective win rates on 7 random samples from the cognitive behaviors test set (0.37, 0.41, 0.43, 0.45, and 0.47), and “Improvement” reflects the percentage uplift over the baseline (ranging from +10.8% to +27.0%). Confidence-Weighted Voting increases accuracy to 0.41 (+10.8%), Re-ranking to 0.43 (+16.2%), Learned Verifier to 0.45 (+21.6%), and Self-Consistency yields the highest accuracy at 0.47 (+27.0%), highlighting the benefit of these enhanced decoding approaches.

It is worth noting that these numbers come from the average rewards generated over a few samples from the test set of the cognitive behaviors dataset. As a result, there is likely to be a bit of noise in these results, which explains the discrepancies we see between each of the methods here. When a ground-truth verifier is available, it directly evaluates the correctness of model outputs, reducing the need to rely on proxy signals like sampling frequency (used in self-consistency) or confidence scores (used in confidence-weighted decoding). In this setting, re-ranking, self-consistency, and confidence-weighted voting all converge on the goal of selecting the most likely correct answer from multiple generations. Since the verifier gives explicit feedback on correctness, all three methods are effectively searching the same response space, just guided by slightly different heuristics. As a result, their performance tends to converge, because once a correct answer exists in the sample pool, any strategy with a strong verifier can identify and prefer it. The marginal benefit of more sophisticated heuristics diminishes when correctness can be directly measured.

5.2 RLOO and Multiple Parsing with SFT

In contrast to confidence-weighted voting, self-consistency and the alternate verifier, multiple parsing passes through the same model-generated output provides us with unique information compared with re-ranking. This is because the model responses include several expressions, some inside structured tags like <answer>, others embedded in natural language reasoning. Running multiple parses, and looking for the alternate answers the model provided previously, improves stability by increasing the chances of recovering valid answers that might otherwise be overlooked. This is especially important when using a ground truth verifier: the model might initially mischaracterize a correct answer as invalid or irrelevant, either due to extraction failure or noisy generation. By re-parsing and re-verifying previous outputs, we reduce the risk of discarding correct answers prematurely and ensure that early, valid completions are properly credited, thus improving the robustness and accuracy of evaluation.

Because of this, and because Reinforcement Learning with Leave-One-Out (RLOO) captures information similar to that leveraged by re-ranking, confidence-weighted voting, and self-consistency, we ultimately adopt a combined strategy: RLOO fine-tuning during training, followed by multiple parsing passes at test time. The previous three strategies all rely, in one way or another, on sampling multiple candidate completions and selecting or scoring them based on some notion of correctness, confidence, or agreement. RLOO mirrors this dynamic by generating multiple candidates per input and using a relative reward signal to reinforce the better completions—implicitly learning preferences over answer quality without requiring an explicit re-ranker or verifier at test time. This allows the model to internalize selection pressures similar to what re-ranking or voting applies post-hoc, so there is less of a benefit to do that at test time.

Our final evaluation strategy thus combines the generative strength of RLOO-trained models with the robustness of multiple-pass extraction. This model achieves the best performance, with nearly a 32% improvement over the staff-SFT baseline. Initially, the performance of this setup did not reach the level we expected. We attribute this to practical compute limitations during training. Specifically, RLOO was run on a significantly reduced subset of the training data to make training time manageable, limiting the diversity and coverage of the reward signal. Additionally, due to GPU memory and runtime constraints, we had to cap the number of sampled completions per input instance during RLOO training, reducing the granularity of intra-batch comparison and potentially weakening the reward differentiation that drives policy improvement. In our second iteration, to combat these problems, we reduced the number of samples we trained over to merely 750, but ensured we were producing 3 candidate solutions, as well as the full 1024 token output. Despite even this limited training, the model was able to meet the threshold.

Even still, these constraints may have limited the baseline RLOO model's ability to fully internalize the distinctions between better and worse solutions across diverse problem instances. Although not listed in the table, baseline RLOO was only able to achieve about a 0.259x improvement over the baseline SFT model, in large part because we did not have the compute or time to re-evaluate the second model we trained, with less training data but more candidate completions (without also using test-time inference). We believe that with access to greater computational resources, allowing for full training data utilization, or perhaps slightly more efficient code that would allow for increased candidate sampling, our baseline RLOO could more effectively realize its potential and surpass the heuristic-based test-time strategies that are similar to it, as well as the appropriate threshold.

6 Discussion

6.1 Key Takeaways:

Test-time inference can be a powerful way to improve LLM accuracy without retraining, especially for reasoning tasks where a single generation is often brittle or non-robust. This is particularly relevant for the Countdown math reasoning benchmark, where single pass generations from SFT models can fail on harder arithmetic compositions. Simple strategies like re-ranking or majority voting already provide meaningful boosts and can be combined for even better results. The consistent improvement across all selection methods demonstrates that test-time inference provides reliable gains, but the narrow performance band reveals fundamental limitations in base model quality. Ultimately, the test-time strategies explored that generate multiple candidate solutions (re-ranking, self-consistency, alternate rewards, and confidence-weighted selection) all provide similar information to RLOO. Thus, RLOO should perform best when combined in tandem with multiple-parsing through a single candidate solution, which can help capture accidentally ignored correct solutions. It is also worth noting that verifier/reward models can help select more accurate solutions, even in domains where ground-truth scoring is unavailable (e.g. UltraFeedback, where the goal is to assess how well the model generalizes to diverse, unseen instructions across various topics and complexities), provided some weak supervision or feedback is available. In these cases, rather than have a ground-truth verifier, one will simply have a learned one instead.

6.2 Additional Limitations

The modest performance gaps that we observed between some of the initial selection methods where we generate multiple candidates (0.41-0.47) highlight that candidate quality is a large bottleneck for test-time inference effectiveness. We see this as well in the success of the second iteration of the RLOO model that utilizes test time compute - by generating full 1024 outputs, as opposed to the more limited 512, we saw model performance increase greatly. Additionally, the effectiveness of these strategies depends on model calibration and the diversity of candidate generations; poorly calibrated or non-diverse models may see smaller improvements. We also believe that more specific optimizing of parameters such as temperature, sampling, and learning rate parameters could affect candidate quality and diversity. Lastly, our current results are on the Countdown math reasoning task; generalization to other domains requires further examination.

Test-time computation also has some limitations in those slightly different settings. For instruction following tasks like UltraFeedback, where our goal is to determine how well models can respond to unseen instructions, the absence of a perfect ground-truth reward makes verifier training less reliable, and results may overestimate true performance if the learned verifier overfits.

6.3 Future Work:

Future work would include potentially developing a domain-aware verifier and reward models that generalize to less-structured tasks like open-ended reasoning or dialogue, and explore joint training of generation and evaluation models to better align criteria. Additionally, it could also include investigating optimal candidate numbers and sampling strategies to maximize test-time inference effectiveness, including more sophisticated tree search algorithms that can dynamically adjust search depth and branching factors based on problem complexity. We could also enhance the revision based tree search further with adaptive pruning strategies that balance exploration and exploitation more effectively. Finally, it is also worth considering applications to real-world tasks and assessing performance under deployment constraints, including user feedback and ambiguous prompts, while optimizing computational efficiency for practical deployment scenarios.

7 Conclusion

We found that RL and inference-time optimization can be systematically combined to enhance the structured reasoning capabilities of small scale large language models. We implemented RLOO finetuning on the Qwen2.5-0.5B model using the Countdown dataset, benchmark for multi-step arithmetic reasoning. Each of the five distinct test-time inference strategies we implemented (confidence-weighted voting, verifier-based reranking, self-consistency, parsing multiple correct completions, and a single-pass SFT baseline) addressed known failure modes in autoregressive reasoning: local minima, incorrect branching, and under-exploration. Our quantitative results demonstrated that these strategies improved verifier scores, validating the hypothesis that moving some computation to inference time can offer robustness without the prohibitive cost of model scaling. Our results show that even with a modest 0.5B parameter model, structured reasoning can be improved through targeted test-time computation, re-ranking, and reinforcement from correctness signals. This opens a lot of possibilities for future models. In the future, this could be expanded to more complex multi-hop symbolic reasoning tasks, incorporate learned reward models (which are more applicable in scenarios where there is no ground truth available), and investigate dynamic inference policies that adaptively balance generation, revision, and verification in real time.

8 Team Contributions

- **Anushree Aggarwal:** Data and model preparation for SFT, helping implement extension, and helped write the final report.

- **Saif Moolji:** Worked on the RLOO and SFT implementations, and helped write the final report.
- **Bryce Tiglon:** Helped with the evaluation and implemented the extension on test-time compute.

Changes from Proposal While our initial proposal focused on combining revision based tree search with verifier guided scoring, we shifted toward implementing and evaluating five distinct, lightweight test-time inference strategies: self-consistency and verifier-based reranking, without full tree expansion due to runtime and implementation constraints. Additionally, we limited our evaluation to the Countdown dataset in order to focus our experimentation on arithmetic reasoning under a fixed compute budget.

References

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. (2023). arXiv preprint arXiv:2303.17651.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. 2024. Recursive Introspection: Teaching Language Model Agents How to Self-Improve. *arXiv preprint arXiv:2407.18219* (2024). `https://doi.org/10.48550/arXiv.2407.18219`

Qwen Team. 2024. Qwen2.5-0.5B: A 0.5B Parameter Language Model. `https://huggingface.co/Qwen/Qwen2.5-0.5B`. Accessed: 2025-05-23.

Anikait Singh. 2024. Cognitive Behavioral Dataset: All Strategies. `https://huggingface.co/datasets/Asap7772/cog_behav_all_strategies`. Accessed: 2025-05-23.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv preprint arXiv:2408.03314* (2024).

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *arXiv preprint arXiv:2305.10601* (2023).