Extended Abstract

Test-Time Inference Scaling for RL Fine-Tuned Language Models

Leah Balakrishnan, Julian Rodriguez Cardenas, Ava Kouhana

Motivation. Most work on RL fine-tuning of language models focuses on improving training, but leaves inference largely unchanged. At test time, models typically generate a single response with no mechanism to recover from mistakes. This paper asks whether smarter inference strategies can extract more performance from already-trained models, and whether the training method affects how much a model benefits from additional compute at inference time.

Setting. We study the Countdown arithmetic reasoning task, where the model must generate an expression from a given set of numbers that evaluates to a target value. A rule-based verifier provides exact binary feedback with no noise, making it a clean setting for studying test-time scaling. We apply three test-time inference strategies to three checkpoints trained with SFT, IPO, and RLOO respectively.

Methods. Our first method, confidence-weighted Best-of-N, replaces random candidate selection with selection based on sequence log-probability. We hypothesize that better-trained models assign higher probability to correct responses, making confidence a useful selection signal. Our second method uses recursive decomposition to break Countdown problems into smaller subproblems, using brute-force enumeration over decompositions to improve coverage. Our third method is a verifier-guided critic-refiner loop, where a separate instruction-tuned model diagnoses failures using a structured Keep/Modify/Plan prompt and the original solver policy generates a refined answer conditioned on the repair plan.

Results. Confidence-weighted selection consistently outperforms random Best-of-N across all three checkpoints, with the largest gains for stronger models. At $N = 30$, IPO improves from 0.28 to 0.58 and RLOO improves from 0.56 to 0.66 under confidence-weighted selection, while random Best-of-N shows no consistent trend. Recursive decomposition with brute-force enumeration improved accuracy by up to 12 points on RLOO. The critic-refiner loop improved solve rate from 70% to 74% on RLOO using the pairwise prompt-v2 configuration. Across all three methods, better-trained models benefit more from inference-time compute.

Takeaway. Test-time compute scaling is not just about sampling more. The quality of candidate selection and refinement matters, and that quality depends on how well the underlying policy is trained. RL training appears to improve not just task accuracy but also the reliability of the model as a component in a larger inference pipeline.

Test-Time Inference Scaling for RL Fine-Tuned Language Models

Leah Balakrishnan
Department of Computer Science
Stanford University
leahtb@stanford.edu

Julian Rodriguez Cardenas
Department of Computer Science
Stanford University
julianrc@stanford.edu

Ava Kouhana
Department of Computer Science
Stanford University
akouhana@stanford.edu

Abstract

Standard RL fine-tuning methods like SFT, IPO, and RLOO improve model behavior during training, but at inference time most systems still rely on a single sampled response. This paper studies whether smarter sampling strategies at test time can improve performance on the Countdown arithmetic reasoning task, where a rule-based verifier provides exact feedback with no noise. We investigate three test-time inference strategies applied to SFT, IPO, and RLOO checkpoints: confidence-weighted Best-of-N selection, recursive problem decomposition, and a verifier-guided critic-refiner loop. Our results show that better-trained models benefit more from inference-time compute, and that structured feedback during refinement produces more consistent gains than simply sampling more responses.

1 Introduction

Large language models trained with reinforcement learning have shown strong performance on reasoning tasks like math and code generation. Most of this progress comes from improvements to the training procedure itself, whether through better reward signals, more stable policy gradient estimators, or preference optimization objectives. Once a model is trained, however, inference is typically just a single forward pass with greedy or temperature sampling. This leaves a lot of potential performance on the table.

Test-time compute scaling has recently emerged as a complementary direction. Rather than improving the model itself, the idea is to spend more computation at inference time to get better answers out of an already-trained model. Simple approaches like Best-of-N sampling have shown that even weak models can achieve high accuracy if given enough chances to try. More structured approaches use verifiers, critics, or search procedures to guide the model toward correct solutions.

What is less well understood is how the training method interacts with test-time inference. If a model is trained with a stronger RL objective, does it benefit more from test-time search? Does a better-trained policy produce better-calibrated probabilities, making confidence-based selection more reliable? These questions motivate our work.

We study three test-time inference strategies on the Countdown task, a controlled arithmetic reasoning benchmark where the model must generate an expression from a set of numbers that evaluates to a given target. A rule-based verifier scores each output with no ambiguity, making it ideal for studying

test-time scaling in isolation. We apply each strategy to three checkpoints trained with progressively stronger objectives: SFT, IPO, and RLOO.

Our contributions are:

- A confidence-weighted Best-of-N method that uses sequence log-probabilities to select the most likely correct response from N candidates, and an analysis of how this interacts with training method quality.
- A recursive decomposition approach that breaks Countdown problems into smaller subproblems and uses brute-force enumeration over decompositions to improve accuracy.
- A verifier-guided multi-model refinement loop that uses a separate critic model to diagnose failures and propose structured repairs, improving solve rate on RLOO from 70% to 74% under matched inference budgets.

2 Related Work

RL fine-tuning of language models. Reinforcement learning from human feedback has become a standard part of language model post-training. Early work used PPO with an explicit reward model (Ouyang et al., 2022), while more recent methods like DPO (Rafailov et al., 2023) and IPO (Azar et al., 2023) avoid reward model training entirely by directly optimizing on preference data. IPO relaxes the Bradley-Terry assumption used in DPO to reduce overfitting to preference labels. REINFORCE-style methods have also seen renewed interest: Ahmadian et al. (2024) propose RLOO, which uses a leave-one-out baseline to reduce gradient variance and has shown strong results on reasoning tasks without the complexity of PPO.

Test-time compute scaling. Snell et al. (2024) showed that scaling compute at inference time can be more efficient than scaling model parameters under matched FLOPs budgets. Best-of-N sampling is the simplest instantiation of this idea: sample N responses and select the best one using a verifier. Chain-of-thought prompting (Wei et al., 2023) similarly improves reasoning by encouraging intermediate steps during decoding.

Verifier-guided and iterative refinement. Zhang et al. (2024) cast reward modeling as next-token prediction, enabling self-correction during inference. Work on iterative refinement (Madaan et al., 2023) has shown that models can improve their outputs by conditioning on feedback from a critic, though gains are sensitive to the quality of the feedback. Our verifier-guided refinement loop builds on this idea, using a structured Keep/Modify/Plan prompt format to give the critic a clearer role than unstructured self-repair.

Recursive decomposition. Prior work has studied how LLMs can collaborate on reasoning trajectories by breaking problems into subgoals that can be solved independently. Our recursive decomposition approach is similar in spirit, applied specifically to the Countdown task where exact arithmetic verification is possible at each step.

3 Extension

3.1 Confidence-Weighted Best-of-N

Standard Best-of-N sampling generates N candidate solutions independently and picks one at random, or picks the first verified correct answer. This treats all samples as equally likely to be correct, regardless of what the model itself thinks about them.

We hypothesize that a well-trained model should assign higher sequence log-probability to correct solutions. In other words, the model's own confidence is a signal worth using. Our method replaces random selection with confidence-weighted selection: for each prompt we sample N responses, compute the cumulative log-probability of each full response sequence, and return the response with the highest log-probability.

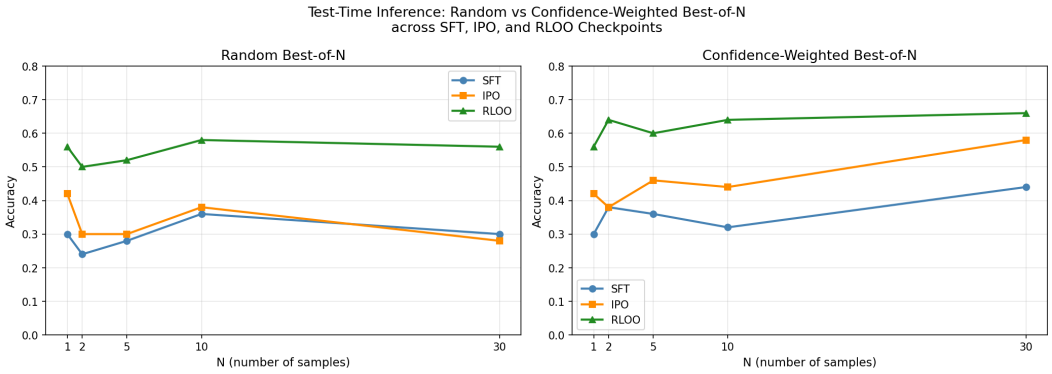

Figure 1: Random Best-of-N (left) vs. confidence-weighted Best-of-N (right) across SFT, IPO, and RLOO checkpoints for $N \in \{1, 2, 5, 10, 30\}$.

Formally, given N sampled responses $y^{(1)}, \ldots, y^{(N)}$ for a prompt x, we select:

$$\hat{y} = \arg\max_i \sum_t \log \pi_\theta(y_t^{(i)} \mid x, y_{<t}^{(i)}) \tag{1}$$

We evaluate this across $N \in \{1, 2, 5, 10, 30\}$ for all three checkpoints and compare against random Best-of-N as a baseline. Log-probabilities are obtained directly from vLLM during sampling with `logprobs=1`.

3.2 Recursive decomposition through self-prompting and naive decomposition

Our second extension involves two separate attempts at recursively decomposing the problem. The Countdown is well-suited for recursive solutions by its own nature. If a solution to a specific countdown problem exists, a solution can be found by listing all pairs of numbers and operations with them, and constructing a solution recursively.

However, the basic approach the model takes at finding a solution does not inherently decompose the problem. Given an example of the Countdown problem, it will attempt to output a solution all in one go. For problems that involve many numbers, the model struggles to find the corresponding solution, naturally performing better at smaller problems.

Thus, we wondered if we could leverage the model we already had (including the IPO and RLOO checkpoints), providing scaffolding for it to try finding solutions that used some decomposition as part of the solution-finding process (the implementations for the two parts of this extension were completed with AI assistance, following course policy).

Our first approach consists of using new prompts to have the model engage with itself in two modes (with no further training). The prompt for planning asks the model to suggest a single pair of numbers and an operation to do with them. Then, a script computes that and prompts the model again with the remainder of the problem. This planning model is used to decompose the problem into easier subproblems, possibly in multiple levels of decomposition. For example, the problem [44, 19, 35] -> 98 might be solved in the following way: 1. the planner picks 44 + 19 = 63, and then the solver finalizes the solution on [63, 35], with 63 + 35 = 98.

While the planner approach is promising, and our scripting made it possible to do multiple rounds of planning before solving a solution, ultimately this method failed because the model was not able to produce correct outputs with the new prompt. For example, instead of producing outputs with <step> </step> structure, it still outputted with the <think> and <answer> structures (more information and examples are included in the results section).

Thus, we tried a second approach at decomposing the problem that only required the LLM to work with the prompt and formatting it had been trained with. The second approach enumerates all the moves possible at a given level (i.e. every pair of numbers and operations possible with them), completes those moves, and outputs a smaller sub-problem. Then, either we repeat this process, or

ask the model to complete a solution from there. Essentially, this naively tries moves to simplify the problem, repeated for a variable number of steps, after which the model searches for a solution in the same way it was trained to do.

3.3 Verifier-guided multi-model refinement loop

Our final method is a verifier-guided multi-model refinement loop. We use a trained RLOO policy as the solver, a separate instruction-tuned model (`Qwen/Qwen2.5-3B-Instruct`) as the critic, and the symbolic Countdown checker already provided in the codebase as the verifier.

The refiner reuses the solver policy. For each problem, the solver first samples a small set of candidate equations. The symbolic verifier checks each candidate for arithmetic validity, number usage, and whether the expression reaches the target. If any candidate is correct, the search terminates early. Otherwise, the system selects the top failed candidates and asks the critic to compare them and produce a short structured repair plan.

The critic prompt uses a three-part format: Keep, Modify, and Plan. "Keep" identifies useful partial structure from a failed trajectory, "Modify" states the main error diagnosed by the verifier, and "Plan" proposes a concrete revision strategy. The solver then conditions on this repair plan to generate a refined answer. This loop repeats for a small number of rounds, and we keep the highest-scoring verified answer across all rounds.

Inference budget. We compare against a one-shot baseline that samples 20 answers and takes the best verified one (*best-of-20*). For the refinement methods, we use 4 initial samples, beam width 2, and up to 3 refinement rounds. This gives a comparable maximum inference budget, while allowing early stopping when a correct answer is found.

We evaluate several variants: (i) a single-model verifier-guided repair loop, (ii) a pairwise critic that compares two failed trajectories before refinement, (iii) a structured prompt-v2 critic using **Keep/Modify/Plan**, and (iv) a dual-solver debate variant where two differently prompted solvers generate initial trajectories and the critic chooses or merges them.

4 Experimental Setup

All experiments use the Countdown arithmetic reasoning task. Each problem gives the model a set of numbers and a target value; the model must generate an arithmetic expression using those numbers that evaluates to the target. A rule-based verifier scores each output as correct (1.0), correctly formatted but wrong (0.1), or invalid (0.0).

We fine-tune Qwen 2.5 0.5B using three training objectives of increasing strength:

- **SFT** – supervised fine-tuning on expert demonstrations from the warm-start dataset, using next-token prediction loss applied only to completion tokens.
- **IPO** – preference optimization trained on pairwise preference data, using the squared margin objective from Azar et al. (2023) with the SFT model as the reference policy.
- **RLOO** – online policy gradient with a leave-one-out baseline (Ahmadian et al., 2024), trained using rollouts scored by the rule-based verifier. Responses are sampled with vLLM and importance weights are applied to correct for numerical differences between the sampling and update policies.

For evaluation we use the test split of `asingh15/countdown_tasks_3to4`, which contains 50 held-out problems. All three test-time inference methods are evaluated on the same set of problems using the same checkpoints, with no additional training. For sampling we use temperature 0.6, top-k 20, and top-p 0.95 following the project evaluation protocol.

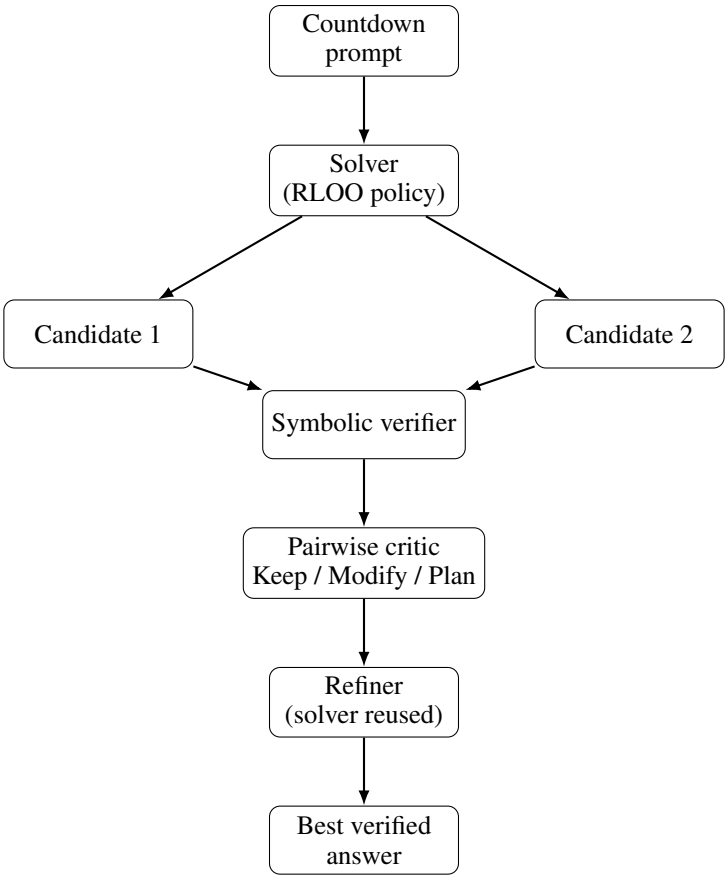

Figure 2: Overview of the best-performing test-time inference method. The solver proposes candidates, the symbolic verifier diagnoses failures, the critic writes a structured repair plan, and the refiner produces a revised answer.

Model	Method	N=1	N=2	N=5	N=10	N=30
SFT	Random BoN	0.30	0.24	0.28	0.36	0.30
SFT	Confidence-Weighted	0.30	0.38	0.36	0.32	0.44
IPO	Random BoN	0.42	0.30	0.30	0.38	0.28
IPO	Confidence-Weighted	0.42	0.38	0.46	0.44	0.58
RLOO	Random BoN	0.56	0.50	0.52	0.58	0.56
RLOO	Confidence-Weighted	0.56	0.64	0.60	0.64	0.66

Table 1: Accuracy of random vs. confidence-weighted Best-of-N across checkpoints and N values.

5 Results

5.1 Quantitative Evaluation

5.1.1 Confidence-Weighted Best-of-N

Table 1 and Figure 1 show the accuracy of random and confidence-weighted Best-of-N across all three checkpoints.

5.1.2 Recursive Decomposition

We evaluated both recursive decomposition methods on the same 50-problem test split of `asingh15/countdown_tasks_3to4` used for the milestones.

Table 2 shows the results of the self-prompting method. On both checkpoints (IPO and RLOO), it either underperforms or basically ties flat Best-of-N at a matched compute budget.

Checkpoint	Configuration	Decomp accuracy	Flat BoN@16 (same run)
IPO	default ($n_{\text{plan}} = 4, n_{\text{solve}} = 8$)	0.62	0.76
IPO	equal-budget ($n_{\text{solve}} = 16$)	0.72	0.76
RLOO	equal-budget ($n_{\text{solve}} = 16$)	0.72	0.70

Table 2: Self-prompting decomposition results

Table 3 shows the naive (brute-force) results. The method outperforms flat Best-of-N on both checkpoints. We compare against Best-of-128 since at $n_{\text{solve\_per\_branch}} = 16$ the brute-force method uses about 368 samples per problem, roughly 2.9 times the compute of Best-of-128.

Checkpoint	Flat BoN@16	Flat BoN@128	Brute-force ($n = 16$/branch)	Gain over BoN@128
IPO	0.76	0.86	**0.92**	**+6**
RLOO	0.71	0.74	**0.86**	**+12**

Table 3: Brute-force (also referred to as naive) decomposition results

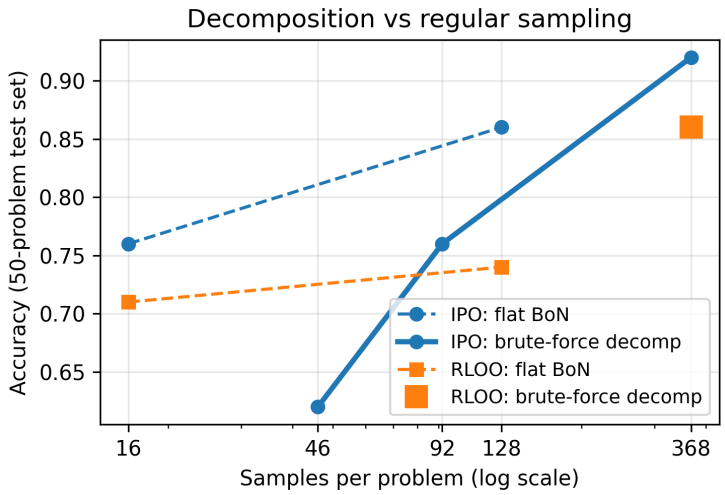

Figure 3: Brute-force decomposition (solid) vs. flat Best-of-N (dashed) on IPO and RLOO checkpoints, across increasing samples per problem.

5.1.3 Verifier-Guided Refinement

We evaluate on 50 Countdown examples from `asingh15/countdown_tasks_3to4`. We report the mean verifier score and solve rate, where solve rate is the fraction of examples with score 1.0. All methods are compared under the same maximum inference budget as the best-of-20 baseline.

5.2 Qualitative Analysis

5.2.1 Confidence-Weighted Best-of-N

Confidence-weighted selection consistently outperforms random sampling across all three models and all values of N. The gap is largest for stronger models: at $N = 30$, IPO improves from 0.28 to 0.58 and RLOO improves from 0.56 to 0.66. Random Best-of-N shows no consistent trend with increasing N, while confidence-weighted accuracy trends upward. This pattern suggests that RL training improves not just raw task accuracy but also the reliability of the model's probability estimates as a signal for correctness.

Inference method	Mean score	Solve rate	Avg. candidates
Best-of-20 baseline	0.730	0.70	20.00
Single-model TTO v1	0.730	0.70	8.80
Pairwise critic v1	0.730	0.70	8.80
Single-critic prompt-v2	0.748	0.72	8.96
Pairwise critic prompt-v2	**0.766**	**0.74**	**8.64**
Dual-solver debate	0.730	0.70	8.80

Table 4: Verifier-guided refinement results on RLOO checkpoint

Interestingly, the improvement is more pronounced for IPO than RLOO at larger N. One possible explanation is that RLOO is already strong enough at $N = 1$ that there is less room to improve, while IPO benefits more from having additional candidates to choose from.

5.2.2 Recursive Decomposition

Trace analysis confirms the failure mode of the self-prompting method: on many problems, the model failed to produce an output that our parser script could use, which then defaulted to the unreduced problem, essentially working without any extension. Another issue is that even when the planner is functional, the first move it chooses does not always make the resulting subproblem easier to solve.

For the brute-force method, when compute is lower per branch, it underperforms flat Best-of-16 because each subproblem gets too few shots to reliably find a solution. As per-branch compute grows, the method scales past flat sampling on both checkpoints. The size of the decomposition gain differs between algorithms. For IPO, flat Best-of-N goes from 0.76 at N=16 to 0.86 at N=128, while naive decomposition adds 6 points. On RLOO, flat Best-of-N improves less, from 0.71 to 0.74, while naive decomposition adds 12 points on top.

5.2.3 Verifier-Guided Refinement

Naive test-time optimization did not help on its own: both the single-model repair loop and the first pairwise critic variant exactly matched the baseline. The improvement came from the structured critic prompt. The best configuration, pairwise critic prompt-v2, increased mean score from 0.730 to 0.766 and solve rate from 70% to 74%.

Although the gain is modest in absolute terms, it is consistent with the hypothesis that structured verifier-guided refinement is more effective than simple reranking or unstructured self-repair. The prompt-v2 design gave the critic a clearer role: preserve useful partial arithmetic, explicitly identify the failure mode, and propose a targeted correction. This produced the only setting that reliably outperformed the baseline at matched budget.

The dual-solver debate variant did not improve over baseline. This suggests that adding more trajectories alone is not sufficient; the quality of the critique and repair interface matters more than simply increasing diversity.

6 Discussion

Across all three test-time inference strategies, a consistent pattern emerges: smarter use of inference-time compute outperforms naive resampling, and the gains are larger for better-trained models.

For confidence-weighted Best-of-N, the results suggest that RL training makes model probabilities more informative. SFT models show modest gains from confidence weighting, while RLOO models show larger and more consistent improvements. This is consistent with the idea that online RL, by repeatedly sampling and updating on correctness feedback, teaches the model to be better calibrated about which of its outputs are likely correct.

For recursive decomposition, brute-force enumeration over subproblems was the more reliable approach. The LLM-planner approach failed because the model was not trained to produce outputs in the structured format the planner expected. This highlights a general challenge with test-time methods that rely on out-of-distribution prompts: the model needs to have seen similar formats during training to respond usefully.

For the verifier-guided refinement loop, the key finding is that the quality of the critic prompt matters more than the number of refinement rounds. The structured Keep/Modify/Plan format gave the critic a specific role to play, which produced more targeted repairs than unstructured self-feedback. The dual-solver debate variant did not help, suggesting that diversity of initial candidates is less important than the quality of the feedback used to refine them.

One limitation across all three methods is the small test set size of 50 problems. Some of the observed differences between conditions may reflect variance rather than true signal. A larger evaluation would give more reliable estimates of the gains from each approach.

7 Conclusion

This paper studied whether smarter test-time inference strategies can improve performance of RL fine-tuned language models on the Countdown task, and whether training method quality affects how much a model benefits from additional inference-time compute.

We found that all three methods produced gains over naive single-sample inference, and that better-trained models benefited more. Confidence-weighted Best-of-N showed that RLOO and IPO checkpoints have more reliable probability estimates than SFT, making confidence a useful selection signal. Recursive decomposition showed that brute-force enumeration over subproblems is more reliable than prompting the model to plan, at least without task-specific training. The verifier-guided critic-refiner loop showed that structured feedback from a separate critic model can improve solve rate modestly but consistently, and that the format of the critique matters more than the number of refinement rounds.

Together these results suggest that the gains from test-time compute are not just about sampling more — the quality of how candidates are selected or refined matters, and that quality depends on the underlying policy. A well-trained policy produces better-calibrated probabilities, generates better candidates to refine, and responds more reliably to structured feedback.

Future work could explore combining these three strategies into a unified pipeline, or studying whether the same patterns hold on harder reasoning tasks beyond Countdown.

8 Team Contributions

- **Leah Balakrishnan:** SFT implementation, Modal training setup, evaluation pipeline, IPO/RLOO milestone report, confidence-weighted Best-of-N extension implementation and analysis, poster and final report writing.
- **Julian Rodriguez Cardenas:** IPO implementation, recursive decomposition extension implementation and analysis, evaluation pipeline, final report subsection writing, poster printing logistics, poster presentation.
- **Ava Kouhana:** RLOO implementation, verifier-guided critic-refiner loop extension implementation and analysis, related work, LaTeX report writing, poster presentation.

Changes from Proposal. Our original proposal focused on Best-of-N sampling and recursive decomposition as the two main test-time inference strategies. During implementation we found that standard Best-of-N alone was not novel enough, so we extended it to confidence-weighted selection using sequence log-probabilities. We also added the verifier-guided critic-refiner loop as a third strategy, which was not in the original proposal but emerged naturally from Ava's work on iterative refinement. The core research question remained the same throughout.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs. *arXiv preprint arXiv:2402.14740* (2024).

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. 2023. A General Theoretical Paradigm to Understand Learning from Human Preferences. *arXiv preprint arXiv:2310.12036* (2023).

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. *arXiv preprint arXiv:2303.17651* (2023).

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *arXiv preprint arXiv:2305.18290* (2023).

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).

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903* (2023).

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024. Generative Verifiers: Reward Modeling as Next-Token Prediction. *arXiv preprint arXiv:2408.15240* (2024).