Extended Abstract

Motivation Reasoning models often produce a correct solution among multiple sampled attempts, even when single-sample decoding fails. This suggests that performance can be improved not only by further training the policy, but also by using additional test-time computation to search over candidate solutions. In this work, we study this idea on the Countdown arithmetic reasoning task, where the model must combine a given set of numbers into an expression that exactly matches a target. Our goal is to test whether a trained policy can be improved at inference time by sampling multiple candidate solutions and using a verifier to select the best one.

Method We use verifier-guided Best-of-K inference. For each Countdown problem, a trained policy samples K independent candidate solutions. A generative verifier then evaluates each candidate M times and produces binary correctness judgments. We average these judgments to obtain a verifier score for each candidate, and select the candidate with the highest average score as the final answer. This separates the problem into candidate generation and candidate selection: the policy proposes possible solutions, while the verifier selects the most promising one. We compare this procedure against single-sample decoding and an oracle Best-of-K upper bound, where the deterministic Countdown checker selects the best candidate from the same sampled set.

Implementation All experiments use Qwen2.5-0.5B as the base model. We evaluate two post-SFT candidate generators: an RLOO-trained policy and an IPO-trained policy. The generative verifier is trained on policy rollouts labeled by the deterministic Countdown checker, which assigns rewards in $\{0.0, 0.1, 1.0\}$. For binary verification, candidates with reward 1.0 are treated as correct, while all others are treated as incorrect. The main evaluation uses 50 held-out Countdown problems from `countdown_tasks_3to4`, with main GenRM setting $(K, M) = (8, 3)$ and ablations over K and M. We also study distillation by fine-tuning a policy on GenRM-selected Best-of-K outputs, and compare it with a random-correct control where the supervised target is chosen randomly from checker-correct samples rather than selected by the verifier.

Results Verifier-guided selection improves over single-sample decoding for both policy initializations. For the RLOO-initialized policy, accuracy improves from 0.52 under single-sample decoding to 0.70 with GenRM selection at $(K, M) = (8, 3)$, nearly matching the oracle Best-of-K accuracy of 0.72. This indicates that the RLOO policy often samples a correct solution and that the verifier is usually able to identify it. For the IPO-initialized policy, GenRM also improves accuracy, from 0.36 to 0.52 at $(K, M) = (8, 3)$, but remains farther from the oracle upper bound of 0.74. This larger oracle–GenRM gap suggests that IPO rollouts contain correct candidates that the current verifier does not always select. In the distillation experiment, the GenRM-distilled policy improves single-sample accuracy from 0.44 to 0.54, suggesting that some test-time gains can be partially amortized back into the policy. A random-correct control suggests that verifier-selected targets may better align with the verifier's selection behavior than arbitrary checker-correct targets.

Discussion The results show that test-time scaling is useful only when additional samples are paired with a strong selection mechanism. Increasing K can make correct solutions more likely to appear, but it also gives the verifier more incorrect distractors to rank. The contrast between RLOO and IPO highlights verifier alignment as a key bottleneck: RLOO is close to the oracle upper bound, while IPO has a much larger selection gap. Our study is limited by the small evaluation set, the controlled nature of Countdown, and the additional inference cost of running K policy samples and $K \times M$ verifier judgments per problem.

Conclusion Overall, our findings support verifier-guided Best-of-K inference as a simple and effective way to improve trained reasoning policies at test time without updating policy parameters. Future work should evaluate on larger test sets, improve verifier calibration across policy distributions, and further study distillation methods that transfer verifier-guided gains into single-sample policy performance.

Test-Time Best-of-K Selection with Generative Verification for Countdown Reasoning

Jingshu Liu
Department of Computer Science
Stanford University
jingshu7@stanford.edu

Abstract

Reasoning models often produce a correct solution among multiple sampled attempts, even when single-sample decoding fails. In this work, we study verifier-guided Best-of-K inference on the Countdown arithmetic reasoning task. For each problem, a trained policy samples multiple candidate solutions, and a generative verifier selects the candidate most likely to be correct using checker-derived supervision.

Generative verification substantially improves test-time reasoning performance, increasing accuracy from 0.52 under single-sample decoding to 0.70 in our main setup, nearly matching the 0.72 oracle Best-of-K upper bound. We further ablate the effects of policy initialization, the number of candidate samples, and verifier sampling, and study a distillation stage that uses verifier-selected solutions to train a new policy. Overall, our results suggest that generative verification is an effective way to scale reasoning at inference time, while distillation provides a path for amortizing these gains into the policy and verifier calibration remains a key bottleneck for future improvement.

1 Introduction

Reasoning performance is not determined only by how a policy is trained, but also by how it is used at inference time. In mathematical reasoning tasks, a model may fail under single-sample decoding even though another sampled trajectory from the same policy contains a correct solution. This motivates test-time methods that sample multiple candidates and use a verifier to select the final answer.

We study this idea on Countdown arithmetic reasoning, where the model must combine a given set of numbers into an arithmetic expression that exactly reaches a target. The task is useful for studying verifier-guided inference because candidate solutions can be automatically checked, allowing us to measure both the quality of the sampled candidate set and the quality of the verifier's selection.

In this work, we apply generative-verifier-guided Best-of-K inference to trained Countdown policies. For each problem, the policy samples multiple candidate solutions, a generative reward model (GenRM) scores each candidate, and the highest-scoring candidate is selected as the final answer. We compare two post-SFT candidate generators, RLOO and IPO, and evaluate three quantities: single-sample accuracy, GenRM-selected Best-of-K accuracy, and oracle Best-of-K accuracy. The oracle uses the deterministic Countdown checker to select the best sampled candidate, providing an upper bound for a perfect verifier on the same candidate set.

Our main result shows that verifier-guided inference substantially improves over single-sample decoding. For the RLOO-initialized policy, GenRM selection improves accuracy from 0.52 to 0.70, nearly matching the oracle Best-of-K accuracy of 0.72. For the IPO-initialized policy, GenRM also

improves performance, but a larger gap to the oracle remains, indicating a stronger verifier-selection bottleneck.

We further study whether test-time gains can be amortized back into the policy through distillation, using verifier-selected solutions as training targets for a new policy. In addition, we perform ablations over policy initialization, the number of candidate samples, and verifier sampling budget.

Our contributions are:

- We implement generative-verifier-guided Best-of-K inference for Countdown arithmetic reasoning.
- We show that GenRM selection improves over single-sample decoding and can nearly match the oracle upper bound for the RLOO-initialized policy.
- We compare RLOO and IPO candidate generators, separating candidate-generation failures from verifier-selection failures using the oracle Best-of-K benchmark.
- We study distillation and ablations over sampling and verifier settings, characterizing when test-time verification improves reasoning and where its bottlenecks remain.

2 Related Work

Post-training objectives for reasoning. Post-training methods improve reasoning through either online rewards or preference supervision. RLOO uses multiple responses per prompt with a leave-one-out baseline to reduce policy-gradient variance (Ahmadian et al., 2024), while DPO and IPO learn from pairwise preferences (Rafailov et al., 2024; Azar et al., 2023). In this work, we use RLOO- and IPO-trained policies as candidate generators and study how much test-time verification can further improve them.

Test-time computation for reasoning. Additional inference-time computation can improve reasoning by sampling multiple candidate trajectories. Wang et al. (2023) introduce self-consistency, which aggregates answers across sampled reasoning paths. Best-of-K inference similarly samples multiple completions and selects one using a scoring rule. More recently, Snell et al. (2024) show that test-time compute can substantially improve reasoning when paired with an effective verifier or reward model. Our work follows this direction by sampling multiple Countdown solutions from a fixed policy and using a verifier to select the final answer.

Verifiers and reward models. A key challenge in Best-of-K inference is selecting the best candidate from a set of generated solutions. Cobbe et al. (2021) show that verifier models can improve mathematical reasoning by ranking candidate solutions generated by a policy. Later work studies outcome-level and process-level supervision for mathematical reasoning, including process reward models that supervise intermediate steps (Lightman et al., 2023). These methods suggest that generation and verification are complementary: the policy explores possible solutions, while the verifier selects among them.

Generative verifiers. Traditional reward models usually assign a scalar score to a prompt-response pair. Generative reward models instead formulate verification as a next-token prediction problem. Zhang et al. (2025) propose Generative Verifiers (GenRM), which judge candidate solutions using a generative objective. We adopt this approach by querying the verifier multiple times for each Countdown candidate and averaging its judgments to obtain a candidate score.

Countdown reasoning environments. Countdown is a compact arithmetic reasoning task with automatically checkable solutions, making it useful for studying rule-based rewards and test-time search. Pan et al. (2025) study reinforcement learning on Countdown-style tasks and show that rule-based rewards can induce reasoning behavior in small language models. Our work is complementary: we focus on how much accuracy can be recovered at inference time using generative verification.

Position of this work. Compared with prior verifier-based reasoning work, our study focuses on a controlled arithmetic setting where oracle correctness is available. This lets us compare single-sample accuracy, GenRM-selected Best-of-K accuracy, and the oracle Best-of-K upper bound. By evaluating

both RLOO- and IPO-initialized candidate generators, we distinguish failures caused by insufficient candidate quality from failures caused by imperfect verifier ranking.

3 Method

Overview. We study whether inference-time search can improve a trained reasoning policy without additional policy updates. Given a Countdown problem, the policy samples K candidate solutions, and a generative verifier scores each candidate for correctness. The final answer is chosen as the candidate with the highest verifier score.

This decomposes reasoning into candidate generation and candidate selection: the policy proposes possible solutions, while the verifier selects the most promising one.

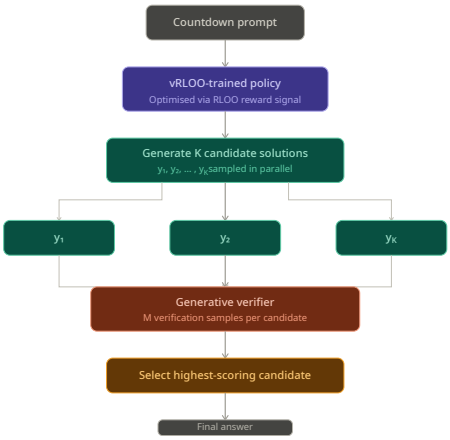

Figure 1: Test-time Best-of-K inference with a generative verifier. For each Countdown problem, the policy samples K candidate solutions. The verifier evaluates each candidate M times, and the candidate with the highest average verifier score is selected as the final answer.

Countdown task. Each input problem x consists of a target number and a set of input numbers. The model must produce an arithmetic expression that uses the given numbers and evaluates exactly to the target. A response y contains both the model's reasoning trajectory and its final expression.

We use the deterministic Countdown checker to assign a reward

$$R(x, y) \in \{0.0, 0.1, 1.0\}.$$

A reward of 1.0 indicates that the final expression is correct. A reward of 0.1 indicates a partially valid response, such as one that follows the expected format or uses the given numbers but does not reach the target. A reward of 0.0 indicates an invalid or incorrect response. During evaluation, we report accuracy using the strict correctness criterion $R(x, y) = 1.0$.

Candidate generation. Let π_a denote a trained policy, where

$$a \in \{\text{RLOO}, \text{IPO}\}.$$

For each held-out Countdown problem x, we sample K independent candidate solutions from the policy:

$$y_1, y_2, \ldots, y_K \sim \pi_a(\cdot \mid x).$$

Each candidate is generated independently using stochastic decoding. When $K = 1$, this reduces to standard single-sample inference. Larger values of K increase the chance that at least one sampled trajectory contains a correct solution, but they also require a mechanism for selecting among the candidates.

Generative verifier. To select among the sampled candidates, we train a generative verifier V_ϕ. The verifier receives both the original Countdown problem x and a candidate solution y_i as input. It is prompted to judge whether the candidate solution is correct.

Unlike a standard scalar reward model, the verifier is generative: it produces a textual judgment that is parsed into a binary correctness score. For each candidate y_i, we sample the verifier M times:

$$z_{i,1}, z_{i,2}, \ldots, z_{i,M} \sim V_\phi(\cdot \mid x, y_i),$$

where each parsed judgment satisfies

$$z_{i,j} \in \{0, 1\}.$$

The verifier score for candidate y_i is the average of these M judgments:

$$s_i = \frac{1}{M} \sum_{j=1}^{M} z_{i,j}.$$

A larger value of s_i means that the verifier more consistently judges the candidate as correct.

Best-of-K selection. After scoring all candidates, we select the candidate with the highest average verifier score:

$$\hat{y} = \arg \max_{y_i \in \{y_1, \ldots, y_K\}} s_i.$$

The selected answer $\hat{y}$ is then evaluated using the deterministic Countdown checker. Importantly, the checker is not used to select the answer during GenRM inference; it is only used to compute the final evaluation metrics.

Verifier training data. The verifier is trained on prompt-candidate pairs collected from policy rollouts. For each training problem, we sample candidate solutions from the policy and label them using the deterministic Countdown checker. This produces examples of the form

$$(x, y_i, R(x, y_i)).$$

For verifier training, candidates with reward 1.0 are treated as correct, while candidates with reward below 1.0 are treated as incorrect for the purpose of binary correctness judgment.

Baselines and oracle comparison. We compare verifier-guided Best-of-K inference against two reference methods. First, the *single-sample baseline* evaluates the first sampled candidate from the policy, corresponding to standard test-time sampling without verification. Second, the *oracle Best-of-K* baseline selects the best candidate according to the deterministic checker:

$$y_{\text{oracle}} = \arg \max_{y_i \in \{y_1, \ldots, y_K\}} R(x, y_i).$$

The oracle is not a deployable method because it uses ground-truth correctness at selection time. Instead, it serves as an upper bound on how well any verifier could perform given the same set of sampled candidates.

We also report pass@K, which measures whether at least one of the K sampled candidates is correct:

$$\text{pass@}K = \mathbb{I}\left[\max_{i \in \{1, \ldots, K\}} R(x, y_i) = 1.0\right].$$

The gap between GenRM accuracy and oracle Best-of-K accuracy measures the remaining verifier-selection bottleneck. A small gap indicates that the verifier usually identifies the correct candidate when one is present, while a large gap indicates that better candidate ranking could unlock further gains.

4 Experimental Setup

Table 1 summarizes the experimental setup. The main experiment evaluates whether GenRM selection improves over standard single-sample decoding at fixed test-time compute $(K, M) = (8, 3)$. We compare the GenRM-selected output against both the first sampled candidate and an oracle Best-of-K candidate selected by the deterministic checker.

We also include a distillation experiment to test whether the gains from verifier-guided inference can be amortized into the policy. In this setting, GenRM-selected Best-of-K outputs are used as supervised fine-tuning targets for a new policy checkpoint, which is then evaluated using single-sample decoding.

Component	Setting
Train and test data	We use `countdown_tasks_3to4`[1] from the default project evaluation setup. Verifier and distillation data are constructed from training prompts using policy rollouts labeled by the deterministic checker. Final evaluation is performed on a held-out test split of 50 Countdown problems unless otherwise stated.
Base model	Qwen2.5-0.5B (Yang et al., 2024; Team, 2024). All candidate generators start from the same supervised fine-tuned checkpoint.
Policy initializations	RLOO-trained and IPO-trained post-SFT policies. RLOO uses online rollouts with checker rewards, while IPO uses preference pairs constructed from checker-derived rewards.
Verifier-data generation	We construct the verifier training set from automatically labeled policy rollouts. For each training prompt, we sample 8 candidate solutions using stochastic decoding with temperature 1.0, top-$p = 1.0$, top-k disabled, and min-$p = 0.0$. We generate rollouts for 2000 Countdown prompts and label each prompt-candidate pair using the deterministic checker.
Main GenRM setting	Best-of-K inference with $K = 8$ candidate solutions per problem and $M = 3$ stochastic verifier judgments per candidate. The final answer is selected by the highest average verifier score.
Verifier decoding	During verifier evaluation, we use stochastic decoding with temperature 0.6, top-$k = 20$, top-$p = 0.95$, and min-$p = 0.0$. Each candidate receives the average score over M verifier samples.
GenRM ablations	We evaluate multiple candidate budgets $K \in \{1, 2, 4, 8, 16\}$ and verifier sample counts $M \in \{1, 3, 5\}$. The main reported setting is $(K, M) = (8, 3)$.
Metrics	Single-sample accuracy, GenRM-selected accuracy, oracle Best-of-K accuracy, mean checker reward, and pass@K.
Distillation dataset	We construct a distillation dataset by running GenRM Best-of-K selection on 3,000 training prompts. For each prompt, the policy samples candidate solutions, the verifier selects the highest-scoring candidate, and the selected response is used as the supervised fine-tuning target for a new policy checkpoint.
Distillation setting	Unless otherwise stated, distillation uses the same Best-of-K selection setting as the main GenRM experiment, $(K, M) = (8, 3)$. The distilled policy is evaluated with single-sample decoding, so improvements measure whether verifier-guided test-time gains can be amortized into the policy.

Table 1: Summary of experimental setup.

5 Results

As shown in Figure 2, verifier-guided Best-of-K inference improves over standard single-sample decoding for both RLOO- and IPO-initialized Countdown policies. In the main RLOO setting, GenRM selection improves accuracy from 0.52 to 0.70 at $(K, M) = (8, 3)$, nearly matching the oracle Best-of-K accuracy of 0.72. The IPO-initialized policy also improves under GenRM selection, although a larger gap remains between GenRM and oracle performance.

These results suggest that test-time sampling can expose correct solutions that are missed by single-sample decoding, while the oracle gap measures how much performance is still limited by verifier selection. We analyze these effects in more detail in the quantitative and qualitative evaluations below.

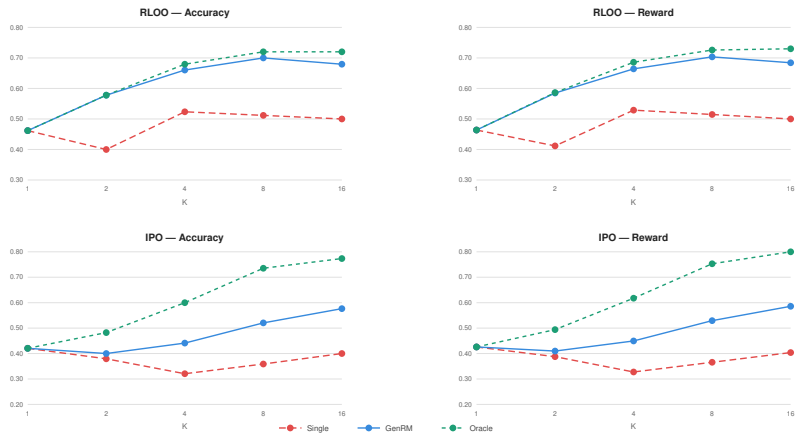

Figure 2: Verifier-guided Best-of-K inference results on Countdown for RLOO- and IPO-initialized policies. GenRM selection improves over single-sample decoding across policy initializations, while the gap to oracle Best-of-K indicates the remaining verifier-selection bottleneck.

5.1 Quantitative Evaluation

Table 2 summarizes the main quantitative results. For each policy initialization and inference setting, we report single-sample accuracy, GenRM-selected accuracy, and oracle Best-of-K accuracy.

Policy	(K, M)	**Accuracy**			**Reward**		
		Single	**GenRM**	**Oracle**	**Single**	**GenRM**	**Oracle**
RLOO-init	$(1, 1)$	0.46	0.46	0.46	0.47	0.47	0.47
RLOO-init	$(2, 3)$	0.40	0.58	0.58	0.40	0.58	0.59
RLOO-init	$(4, 3)$	0.54	0.66	0.68	0.54	0.66	0.68
RLOO-init	$(\mathbf{8}, \mathbf{3})$	**0.52**	**0.70**	**0.72**	**0.52**	**0.70**	**0.73**
RLOO-init	$(8, 5)$	0.44	0.68	0.70	0.46	0.68	0.71
RLOO-init	$(16, 3)$	0.50	0.68	0.72	0.50	0.68	0.73
IPO-init	$(1, 1)$	0.42	0.42	0.42	0.43	0.43	0.43
IPO-init	$(2, 3)$	0.38	0.40	0.48	0.39	0.41	0.49
IPO-init	$(4, 3)$	0.32	0.44	0.60	0.33	0.45	0.62
IPO-init	$(\mathbf{8}, \mathbf{3})$	**0.36**	**0.52**	**0.74**	**0.37**	**0.53**	**0.75**
IPO-init	$(16, 3)$	0.30	0.60	0.78	0.40	0.58	0.80

Table 2: Main results and experimental findings. Accuracy and reward are computed using the deterministic Countdown checker. The RLOO-init $(8, 5)$ row is an additional verifier-sampling run.

For the RLOO-initialized policy, GenRM selection closely tracks the oracle upper bound for moderate values of K. At $K = 8$, GenRM accuracy is 0.70, compared with an oracle accuracy of 0.72. This small gap indicates that the verifier is usually able to identify a correct candidate when one appears in the sampled set. However, increasing to $K = 16$ does not further improve GenRM accuracy, even though the oracle remains high. This suggests that larger candidate sets may introduce more difficult negative examples or increase the burden on the verifier's ranking ability.

For the IPO-initialized policy, the oracle accuracy increases substantially with larger K, reaching 0.78 at $K = 16$. GenRM accuracy also improves over the single-sample baseline, increasing from 0.30 to 0.60 in the $K = 16$ setting. However, the GenRM–oracle gap remains larger than in the RLOO setting. This indicates that the IPO policy can generate correct solutions, but the current verifier is less reliable at selecting them. One possible explanation is that the verifier training distribution is better aligned with RLOO-style rollouts than with IPO-style rollouts.

These results suggest that Best-of-K sampling can expose reasoning ability that is missed by single-sample decoding, but only when paired with a reliable selection mechanism. The small GenRM–oracle

gap for RLOO indicates strong verifier alignment, while the larger gap for IPO shows that additional samples alone are not sufficient when candidate ranking remains imperfect.

Table 3 evaluates whether verifier-selected Best-of-K outputs can be distilled back into the policy. In this separate distillation evaluation run, the RLOO baseline obtains 0.44 single-sample accuracy, while the GenRM-distilled policy reaches 0.54. This suggests that some inference-time gains can be amortized into the policy through supervised fine-tuning. However, the gain is smaller than direct GenRM Best-of-K selection, indicating that distillation does not fully recover the benefit of test-time verification.

Policy	**Distillation Target**	**Single Acc.**	**Oracle/pass@8**
RLOO-init baseline, distill eval	No distillation	0.44	0.66
GenRM-distilled RLOO	GenRM-selected Best-of-8 outputs	**0.54**	**0.70**

Table 3: Distillation results on Countdown. The distilled policies are evaluated with single-sample decoding to test whether GenRM Best-of-K gains can be amortized into the policy. Oracle/pass@8 measures whether at least one of eight sampled candidates is correct.

Table 4 provides an additional control by comparing GenRM-distilled outputs with a random-correct distilled policy. The random-correct policy has comparable oracle/pass@8, but lower GenRM-selected accuracy, suggesting that the verifier-selected distillation targets may better align the policy with the verifier's selection behavior.

Evaluated Policy	**Single Acc.**	**GenRM Acc.**	**Oracle/pass@8**
GenRM-distilled RLOO	**0.54**	**0.70**	0.72
Random-correct distilled RLOO	0.50	0.64	0.72

Table 4: Verifier-selection checks on distilled and control policies. All rows use Best-of-8 inference with $M = 3$ verifier judgments per candidate. The random-correct distilled policy is a control trained on randomly selected checker-correct outputs rather than GenRM-selected outputs.

5.2 Qualitative Analysis

The main qualitative distinction is between *candidate-generation failures* and *verifier-selection failures*. Candidate-generation failures occur when none of the K sampled solutions is correct; these are measured by $1 -$ Oracle Acc. Verifier-selection failures occur when a correct candidate is present, but GenRM selects an incorrect one; these are measured by the gap between oracle accuracy and GenRM accuracy. In the main RLOO setting $(K, M) = (8, 3)$, oracle accuracy is 0.72 and GenRM accuracy is 0.70, leaving a selection gap of only 0.02. Since evaluation uses 50 examples, this corresponds to roughly one example where a correct solution was available but not selected. Thus, most remaining RLOO errors come from candidate-generation failures rather than verifier-selection failures.

The IPO policy shows a different error pattern. At $(K, M) = (8, 3)$, IPO oracle accuracy reaches 0.74, but GenRM accuracy is only 0.52, leaving a much larger selection gap of 0.22. On 50 examples, this corresponds to about 11 problems where a correct candidate was present but not selected. This indicates that IPO candidate generation is not the only bottleneck: the policy often samples a correct solution, but the verifier does not rank IPO-generated candidates as reliably as RLOO-generated candidates. This may reflect distribution mismatch between the verifier's training data and IPO rollouts, or a calibration issue where fluent but incorrect reasoning traces receive high verifier scores.

Increasing K further separates candidate availability from candidate ranking. Larger candidate sets improve the chance that at least one correct solution appears, but they also introduce more incorrect distractors for the verifier to rank. For RLOO, moving from $K = 8$ to $K = 16$ keeps the oracle accuracy roughly unchanged at 0.72, while GenRM accuracy slightly decreases from 0.70 to 0.68. This suggests that once the candidate set already contains correct solutions, additional samples do not necessarily improve final accuracy unless the verifier can reliably rank the larger set. For IPO, the

oracle remains well above GenRM accuracy at larger K, again indicating that verifier selection is the main bottleneck.

The distillation results provide a complementary view of whether verifier-guided test-time gains can be transferred back into the policy. The GenRM-distilled RLOO policy improves single-sample accuracy from 0.44 to 0.54, suggesting that some of the benefit of Best-of-K selection can be amortized through supervised fine-tuning. However, the gain is much smaller than direct GenRM inference, indicating that distillation only partially recovers the test-time advantage.

The random-correct distilled control is also informative: although it has the same oracle/pass@8 value as the GenRM-distilled policy in this verifier-check run, its GenRM-selected accuracy is lower (0.64 vs. 0.70). This suggests that verifier-selected distillation targets may better align the policy with the verifier's selection behavior than randomly selected checker-correct targets.

6 Discussion

Our experiments suggest that verifier-guided Best-of-K inference can improve a trained reasoning policy at test time, but they also reveal several limitations of the current approach.

Limitations. First, our main evaluation uses only 50 held-out Countdown problems, so each example changes accuracy by 0.02. Small differences between settings may therefore reflect sampling variance, and larger evaluations with multiple seeds are needed.

Second, Countdown has a deterministic checker, which makes verifier training and evaluation straightforward but limits generality. It remains unclear how well the same method transfers to open-ended reasoning tasks without automatic correctness labels.

Third, the verifier is not equally reliable across policy initializations. For RLOO at $(K, M) = (8, 3)$, the oracle–GenRM gap is only 0.02, while for IPO it is 0.22. This suggests that IPO failures are often selection failures rather than generation failures.

Finally, the method is computationally expensive. The main setting requires 8 policy rollouts and 24 verifier judgments per problem, compared with one rollout for single-sample decoding.

Future work. Future work should first evaluate the method on larger test sets, multiple random seeds, and larger candidate budgets to better understand how performance scales with K and M.

A second direction is improving verifier calibration. Policy-specific verifiers, mixed RLOO/IPO verifier training data, or verifier ensembles could reduce the distribution mismatch observed in the IPO setting. Richer verifier targets, such as reward-level prediction or error-type classification, may also improve arithmetic checking.

Finally, distillation remains an important way to amortize test-time gains. Verifier-selected Best-of-K outputs can be used to train a policy that performs better under single-sample decoding, reducing the need for expensive verifier calls at deployment time.

7 Conclusion

We studied verifier-guided Best-of-K inference for the Countdown arithmetic reasoning task. Given trained RLOO- and IPO-initialized policies, we sample multiple candidate solutions and use a generative verifier to select the final answer. This simple test-time procedure improves over single-sample decoding without updating the policy parameters.

For the RLOO-initialized policy, GenRM selection improves accuracy from 0.52 to 0.70 at $K = 8$, nearly matching the oracle Best-of-K accuracy of 0.72. The IPO-initialized policy also improves, but leaves a larger gap to the oracle, indicating that verifier ranking remains a key bottleneck. Overall, our results show that test-time sampling combined with generative verification can improve reasoning performance, while the GenRM–oracle gap helps diagnose whether future gains require better candidate generation or better candidate selection.

8 Team Contributions

- **Jingshu Liu** led the research extension and completed the implementation, experimentation, analysis, and write-up for the verifier-guided Best-of-K inference study. This included working on the SFT baseline, RLOO training and evaluation, generative verifier construction, GenRM inference experiments, distillation experiments, result analysis, and paper preparation. Jingshu also assisted with the IPO-related evaluation and comparison.

Changes from Proposal Compared with the original proposal, we expanded the experimental analysis in three ways. First, we added a distillation experiment to test whether verifier-selected Best-of-K outputs can be amortized back into the policy. Second, we included a random-correct selection baseline to compare GenRM-selected targets against randomly selected checker-correct samples. Third, we performed additional ablations over policy initialization, candidate budget K, and verifier sampling budget M.

Use of AI Assistance. OpenAI ChatGPT was used to assist with wording, grammar, organization, and LaTeX polishing during the preparation of this report.

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. In *Annual Meeting of the Association for Computational Linguistics*.

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:2310.12036 [cs.AI] `https://arxiv.org/abs/2310.12036`

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168 [cs.LG] `https://arxiv.org/abs/2110.14168`

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step. arXiv:2305.20050 [cs.LG] `https://arxiv.org/abs/2305.20050`

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. 2025. TinyZero. `https://github.com/Jiayi-Pan/TinyZero`. Accessed: 2026-06-07.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] `https://arxiv.org/abs/2305.18290`

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:2408.03314 [cs.LG] `https://arxiv.org/abs/2408.03314`

Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. `https://qwenlm.github.io/blog/qwen2.5/`

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL] `https://arxiv.org/abs/2203.11171`

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan

Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671* (2024).

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2025. Generative Verifiers: Reward Modeling as Next-Token Prediction. arXiv:2408.15240 [cs.LG] `https://arxiv.org/abs/2408.15240`