Verifier-Guided Recombination Search for Token-Efficient Test-Time Compute in Countdown

Shyam Sai Bethina[1] Ananya Ganapathi[1] Sahil Koita[2]

[1]Department of Computer Science, Stanford University
[2]Department of Statistics, Stanford University

CS 224R: Deep Reinforcement Learning

Extended Abstract

Motivation and Problem Statement. Scaling test-time compute is one promising direction for improving reasoning in large language models [Snell et al., 2024]. A currently popular approach, Best-of-N (BoN) sampling, generates N complete solutions and returns the highest-scoring one according to a verifier. While it is effectice, BoN treats every rollout as an atomic object. That is, if a rollout fails, its entire chain-of-thought is discarded, including any arithmetic substeps that were individually correct and verifier-approved. In the Countdown arithmetic task, we observe that many incorrect rollouts contain sequences of valid operations that, if recombined with steps from other rollouts, would constitute a complete correct solution. As such, we pose the question question: can we recover useful reasoning fragments from failed rollouts and recombine them into correct solutions without spending additional inference tokens?

Method and Novelty. We introduce Verifier-Guided Recombination Search (VGRS), a post-hoc inference strategy that operates entirely on already-generated Best-of-N rollouts. VGRS mines every arithmetic expression of the form $aopb = c$ from all N rollouts using a regex parser, passes each candidate through a symbolic verifier that checks operand availability and arithmetic correctness against the exact Countdown state, and stores verified moves in a shared 'move pool'. It then runs a complete symbolic search restricted to moves in that pool. Because the search engine is purely symbolic (no language model calls), VGRS does not add generated tokens to the BoN budget. The key novelty is the recombination: the move pool aggregates correct reasoning fragments from all rollouts, enabling the construction of solutions that did not appear in any individual generation. We contrast VGRS with online verifier-guided search, which queries the policy model step-by-step during inference which requires additional token use.

Implementation Details and Headline Results. We implement a framework around a symbolic CountdownState that tracks remaining numbers as exact Fraction values (eliminating floating-point drift), immutable move application, and a batched beam search engine decoupled from any model dependency via a Proposer interface. We evaluate on a 50-problem Countdown benchmark using Qwen2.5-0.5B checkpoints fine-tuned with IPO and RLOO. Our headline results:

- VGRS (RLOO): improves over Best-of-N@16 accuracy from 0.733 to 0.780 at the same token budget (gain $+0.047 \pm 0.012$, consistent across 3 seeds), reaching BoN@16 accuracy using only $\sim$58% of the tokens.
- VGRS (IPO): matches Best-of-N@16 accuracy (0.807 ± 0.012 vs. 0.793 ± 0.012; statistically tied) at zero additional cost, reaching BoN@16 accuracy at $\sim$81% of the budget.
- VGRS lies on or above the Best-of-N accuracy-vs-token Pareto frontier for every value of k from 1 to 16 on both models.
- Online search underperforms Best-of-N at a matched token budget on both models (IPO 0.613 vs. 0.680 budget-matched BoN; RLOO 0.580 vs. 0.733), as its stepwise proposal prompt is out-of-distribution for the trained policies.

Discussion, Limitations, and Conclusion. VGRS demonstrates that Best-of-N rollouts contain substantially more exploitable reasoning than their final answers alone convey. The approach is training-free, adds no inference cost, and is guaranteed to be at least as good as BoN. Its main limitation is dependence on the structured arithmetic nature of Countdown. The extraction and verification strategy does not easily generalize to tasks where intermediate steps lack a symbolic verifier. Online search is additionally sensitive to the training objective of the base policy. RLOO-trained models generate fewer extractable step-level signals, causing early search starvation. Future work should explore learned heuristics for search, hybrid online-offline recombination, and extension

to broader mathematical reasoning domains.

Abstract

Test-time scaling through Best-of-N (BoN) sampling improves reasoning accuracy in large language models but discards potentially useful intermediate computations from failed rollouts. We study the Countdown arithmetic task and observe that many incorrect rollouts contain individually correct, verifier-approved arithmetic substeps that are complementary across generations. We introduce Verifier-Guided Recombination Search (VGRS), a post-hoc inference strategy that mines verified arithmetic operations from all N BoN rollouts into a shared move pool and performs complete symbolic search over that pool, thus adding no additional generated tokens to the BoN budget. On a 50-problem Countdown benchmark with Qwen2.5-0.5B policies fine-tuned via IPO and RLOO (results averaged over 3 decoding seeds), VGRS strictly improves over BoN@16 for RLOO (0.780 vs. 0.733) at matched cost while matching BoN@16 for IPO (0.807 vs. 0.793), reaching BoN@16 accuracy with only $\sim$58% (RLOO) and $\sim$81% (IPO) of the token budget. VGRS lies on or above the BoN Pareto frontier for every rollout count k, demonstrating that recombining verified reasoning fragments is a strictly more token-efficient use of test-time compute than independent sampling alone.

1 Introduction

Improving the reasoning capabilities of large language models (LLMs) has emerged as a central challenge in modern NLP. One particularly effective strategy is to scale test-time compute. Rather than relying on a single decoding pass, the model generates multiple candidate solutions and uses a verifier to select among them [Snell et al., 2024]. Best-of-N (BoN) sampling is the canonical instantiation, generating N complete solutions, scoring each with a verifier, and returning the best. BoN is simple, parallelizable, and effective across many reasoning benchmarks.

However, BoN has a core inefficiency in that it treats each rollout as an atomic unit. A rollout that fails to reach the correct final answer is entirely discarded, even if it contains many correct intermediate reasoning steps. In compositional tasks like arithmetic reasoning, a model may correctly compute $44 + 19 = 63$ in one rollout and correctly continue from 63 in another, yet neither rollout independently produces the right answer. The correct solution is latent across the two rollouts but invisible to BoN.

The Countdown task is an ideal setting for studying this phenomenon. Given a set of input numbers and a target value, the model must find a sequence of binary arithmetic operations $(+, -, \times, \div)$ that uses each number exactly once and evaluates to the target. Countdown has a natural symbolic structure. Each intermediate step can be verified independently, partial solutions remain meaningful, and the full solution space is a tree over arithmetic states. This makes it possible to extract, verify, and recombine reasoning fragments across rollouts in a principled way.

Our project can be divided into three research questions:

1. Do failed BoN rollouts contain exploitable correct reasoning fragments, and how frequently?
2. Can verified arithmetic steps from multiple rollouts be recombined into correct solutions that appear in no individual rollout?
3. Does verifier-guided recombination search dominate BoN on the accuracy-vs-token Pareto frontier, and does this hold across different RL fine-tuning objectives?

We answer all three in the affirmative through VGRS and a companion online verifier-guided search method, evaluated on Qwen2.5-0.5B policies trained with IPO and RLOO.

2 Related Work

Test-Time Compute Scaling. Snell et al. [2024] show that scaling test-time compute can be more

effective than scaling model parameters, and characterize the compute-optimal strategy as a function of problem difficulty and the verifier quality. Our work operates in this framework, replacing independent BoN sampling with a structured search that reuses existing compute.

Verifier-Guided Search. Tree-of-Thought [Yao et al., 2023] and related work decompose reasoning into intermediate steps and use an LLM to evaluate partial solutions, enabling tree search over reasoning states. Our online search method is related but uses a symbolic verifier (exact arithmetic checking) rather than a model-based value function, which is cheaper and still exact for Countdown. Process Reward Models (PRMs) [Lightman et al., 2023] train separate verifiers to score intermediate reasoning steps. We instead use a no cost symbolic verifier specific to the task's arithmetic structure.

Reasoning Recombination. Chen et al. [2025] study whether LLMs can collaborate on reasoning trajectories by sharing intermediate outputs. VGRS is related in spirit but operates post-hoc over verified symbolic steps rather than requiring online inter-model communication, and adds zero inference cost. Beam search decoding and diverse beam search explore multiple partial hypotheses but do not mine and recombine fragments from independent complete rollouts.

Generative Verifiers. Zhang et al. [2024] propose using LLMs themselves as verifiers through next-token prediction, showing that generative reward models can provide richer supervision signals than discriminative verifiers. Our approach uses a symbolic verifier that is exact and free, which is possible precisely because Countdown's arithmetic structure admits ground-truth checking at every step.

3 Method

3.1 Problem Formulation

A Countdown instance is a pair $(\mathcal{N}, t)$ where $\mathcal{N} = \{n_1, \ldots, n_k\}$ is a multiset of integers and t is a target integer. A solution is a sequence of binary operations that uses each element of $\mathcal{N}$ exactly once and evaluates to t. We represent the search state as a CountdownState: a tuple $(\mathbf{v}, \mathbf{e}, t, H)$ where $\mathbf{v} \in \mathbb{Q}^m$ is the multiset of remaining values (stored as exact Fractions), $\mathbf{e}$ is the corresponding tuple of expression strings, t is the target, and H is the step history. A state is solved when $m = 1$ and $v_1 = t$.

A move $\mu = (i, j, \texttt{op})$ selects two distinct slots $i \neq j$ and an operator $\texttt{op} \in \{+, -, \times, \div\}$. Applying μ to state s produces a child state s' with $m - 1$ remaining values: the two consumed slots are replaced by a single slot holding $v_i \,\texttt{op}\, v_j$ and its corresponding expression string. Division by zero returns None (invalid move). Commutative operators $(+, \times)$ are de-duplicated by requiring $i < j$.

3.2 Symbolic Per-Step Verifier

The verifier is the core of the method. Given a candidate move $(a, \texttt{op}, b, c)$ parsed from model text and a current state s, it checks:

1. a is available as slot i in $\mathbf{v}$ (exact Fraction comparison),
2. b is available as a distinct slot $j \neq i$ in $\mathbf{v}$,
3. $a \,\texttt{op}\, b$ is well-defined (no division by zero),
4. if the model stated a result c, then $a \,\texttt{op}\, b = c$ exactly.

This check runs in $O(m)$ per candidate and is called after every proposed step, enabling immediate pruning of invalid branches.

3.3 Text-to-Move Extraction

Rather than constraining the model to a rigid output format, we mine candidate moves from free-form text using two regex patterns:

- With result: matches a `op` $b = c$,
- Without result: matches a `op` b (fallback).

Each match is passed to the verifier and survivors are de-duplicated by the value multiset of the resulting child state. This design does not penalize the model for including natural language alongside arithmetic.

3.4 Online Verifier-Guided Search

Online search replaces one-shot full-solution decoding with an interactive loop. At each node in the search tree the policy is prompted with the current partial state (the remaining numbers and target) and is asked to propose next arithmetic steps. The prompt deliberately does not constrain output format. The text-to-move extractor handles the rest. Surviving children are scored by a heuristic:

$$h(s') = w_{\text{prog}} \cdot \text{progress}(s') + w_{\text{prox}} \cdot \text{proximity}(s') + w_{\text{lik}} \cdot \text{likelihood}$$

where progress rewards reducing the number of remaining slots (always increases by 1 per move), proximity $= 1/(1 + |v_{\text{last}} - t|)$ rewards the new slot value being close to the target, and likelihood is the model's log-probability of the proposed step. The search supports both beam search (keep top-B states at each depth) and best-first search (priority queue over all frontier states).

Algorithm 1 Online Verifier-Guided Beam Search

Require: Problem $(\mathcal{N}, t)$, proposer π, beam width B, max expansions E
$s_0 \leftarrow \text{InitState}(\mathcal{N}, t)$; frontier $\leftarrow [s_0]$; expanded $\leftarrow 0$
while frontier $\neq \emptyset$ and expanded $< E$ do
  texts $\leftarrow \pi(\text{frontier})$ ▷ Sample proposals for all frontier states
  children $\leftarrow \emptyset$
  for each (s, text) in zip(frontier, texts) do
    for each valid move $\mu \in \text{ExtractValidMoves}(s, \text{text})$ do
      $s' \leftarrow \text{ApplyMove}(s, \mu)$
      if s'.IsSolved() then return s'
      end if
      children $\leftarrow$ children $\cup \{(h(s'), s')\}$
    end for
  end for
  frontier $\leftarrow$ top-B states from children by $h(\cdot)$
  expanded $\leftarrow$ expanded + |frontier|
end whilereturn None

3.5 Verifier-Guided Recombination Search (VGRS)

VGRS is a post-hoc method that requires no additional model inference. Given N complete rollout texts from Best-of-N generation, it:

1. Extracts all arithmetic triples $(a, \texttt{op}, b, c)$ from every rollout using the text-to-move extractor.
2. Verifies each triple against the root state s_0 and all states reachable by previously verified moves, building a move pool $\mathcal{P} \subseteq \{(s, \mu)\}$.
3. Searches over states using only moves in $\mathcal{P}$, running a complete symbolic DFS restricted to pooled moves.

Algorithm 2 Verifier-Guided Recombination Search (VGRS)

Require: Problem $(\mathcal{N}, t)$, rollout texts $\{r_1, \ldots, r_N\}$, max nodes M
$s_0 \leftarrow \text{InitState}(\mathcal{N}, t)$
$\mathcal{P} \leftarrow \bigcup_{i=1}^{N} \text{ExtractValidMoves}(s_0, r_i)$
frontier $\leftarrow [s_0]$; visited $\leftarrow \{s_0.\text{Key}()\}$; nodes $\leftarrow 0$
while frontier $\neq \emptyset$ and nodes $< M$ do
    $s \leftarrow$ frontier.pop()
    for each $\mu \in \mathcal{P}$ applicable to s do
        $s' \leftarrow \text{ApplyMove}(s, \mu)$
        if s'.IsSolved() then return s'
        end if
        if $s'.\text{Key}() \notin$ visited then
            visited $\leftarrow$ visited $\cup \{s'.\text{Key}()\}$; frontier.append(s')
        end if
    end for
    nodes $\leftarrow$ nodes +1
end whilereturn None

Why VGRS is provably $\geq$ BoN. If any single rollout r_i is correct, its sequence of moves is fully in $\mathcal{P}$ (each step verified against the exact state it was applied to). VGRS will therefore rediscover this solution during search. VGRS can additionally find solutions by recombining moves from different rollouts, so its coverage is a superset of BoN's.

Token budget. VGRS reuses the exact rollout texts generated by BoN. No new tokens are sampled. The token budget for VGRS equals the token budget for BoN@N exactly.

4 Experimental Setup

Task and Dataset. We evaluate on the Countdown task using the asingh15/countdown_tasks_3to4 dataset, which contains problems with 3–4 input numbers. We use the test split, evaluating on 50 problems. Each problem is scored by evaluation/countdown.py, which extracts the expression from <answer>...</answer> tags and checks it evaluates to the target using all input numbers exactly once.

Policies. We use Qwen2.5-0.5B fine-tuned checkpoints trained with two RL objectives:

- IPO: Identity Preference Optimization, which minimizes a pairwise preference loss with a direct regularization term, encouraging step-level reasoning quality.
- RLOO: REINFORCE Leave-One-Out, which optimizes expected trajectory-level return using a leave-one-out baseline for variance reduction.

Baselines.

- Single sample: one full solution per problem, greedy-ish decoding (temperature 0.6).
- Best-of-N@16: 16 full solutions per problem, select highest-scoring by verifier. Temperature 0.6, top-p 0.95, top-k 20.
- Online verifier-guided search: beam width 4, max 48 expansions, 4 proposal samples per node, proposal temperature 0.8, max 128 proposal tokens.
- VGRS: applied post-hoc to BoN@16 rollouts, max 5,000 symbolic nodes.

Metrics.

- Accuracy: fraction of problems where the returned answer passes the official verifier.
- Average generated tokens / problem: total tokens generated by the LM divided by number of problems. VGRS inherits BoN's token count (zero additional).
- Budget-matched BoN accuracy: for each problem, give BoN exactly the token budget spent by online search, consume samples in order, and check if any affordable sample was correct. Averaged across problems — the headline comparison for online search.
- Pass@k / coverage curve: for $k = 1, \ldots, 16$, accuracy of BoN@k vs. VGRS@k using the first k rollouts. Plotted against average tokens to produce the Pareto frontier figure.

5 Results

5.1 Quantitative Results

Table 1: Accuracy on the 50-problem Countdown benchmark (mean ± std over 3 decoding seeds). Best result per model in bold.

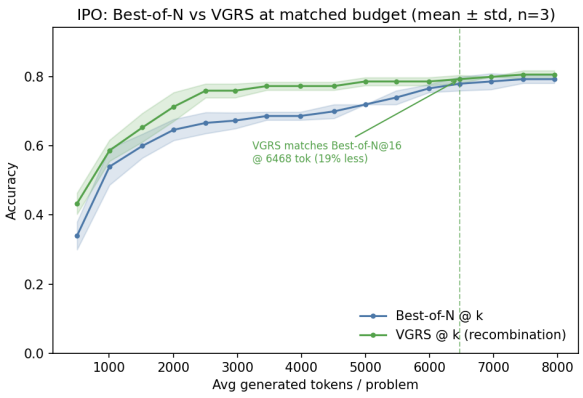
Method	IPO	RLOO
Single Sample	0.360 ± 0.020	0.287 ± 0.042
Best-of-N@16	0.793 ± 0.012	0.733 ± 0.031
Online Verifier Search	0.613 ± 0.031	0.580 ± 0.069
VGRS (Ours)	$\mathbf{0.807 \pm 0.012}$	$\mathbf{0.780 \pm 0.035}$

Main accuracy comparison. Table 1 shows the main accuracy results (mean ± std over 3 decoding seeds). For RLOO, VGRS exceeds BoN@16 by nearly 5 points (0.780 vs. 0.733; gain $+0.047 \pm 0.012$) at the same budget, and reaches BoN@16 accuracy using only ∼58% of the tokens (see Figure 1). For IPO, VGRS and BoN@16 are statistically tied (0.807 vs. 0.793), with VGRS reaching BoN@16 accuracy at ∼81% of the budget. Online search underperforms budget-matched BoN for both models (IPO 0.613 vs. 0.680; RLOO 0.580 vs. 0.733), and degrades most for RLOO, a discrepancy we analyze in Section 5.2.

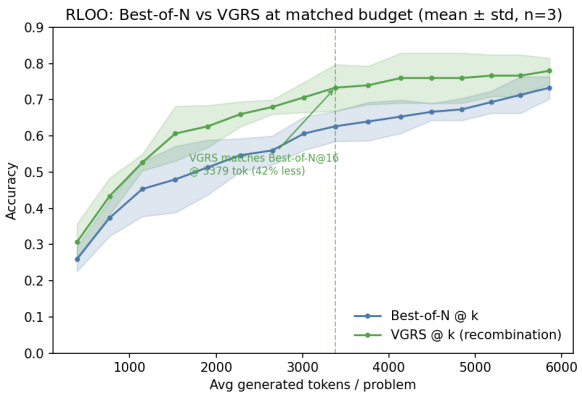
Table 2: Average generated tokens per problem (mean ± std over 3 decoding seeds). VGRS shares BoN's budget exactly.

Method	IPO (tokens)	RLOO (tokens)
Single Sample	499.0 ± 23.9	376.5 ± 20.1
Best-of-N@16	7946.6 ± 27.4	5858.3 ± 33.7
Online Verifier Search	3604.5 ± 10.2	3485.0 ± 23.6
VGRS (Ours)	7946.6 ± 27.4	5858.3 ± 33.7

Token efficiency. Online search for IPO achieves 0.613 accuracy at ∼3,600 tokens/problem, but budget-matched BoN at the same budget reaches 0.680 — i.e. online search does not beat independent sampling at matched cost, and the gap is far larger for RLOO (0.580 vs. 0.733). The token-efficiency win therefore comes from VGRS, which reuses BoN's rollouts at zero extra cost rather than spending tokens on an out-of-distribution stepwise prompt.



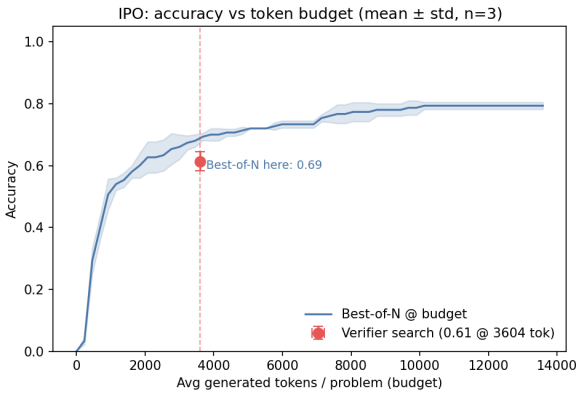
(a) IPO: VGRS lies above BoN@k throughout and reaches BoN@16 accuracy at ∼81% of the compute.



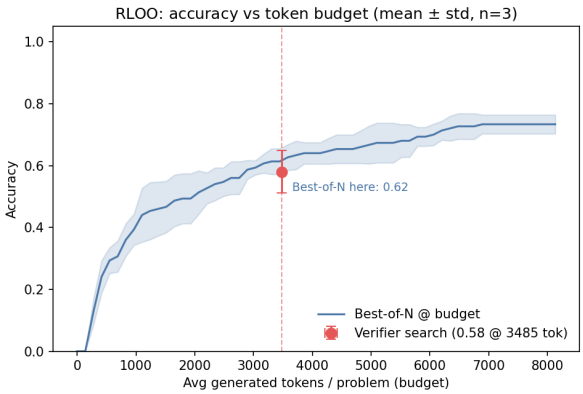
(b) RLOO: VGRS exceeds BoN@16 (0.780 vs. 0.733) and reaches BoN@16 accuracy at ∼58% of the compute.

Figure 1: VGRS vs. BoN@k at matched token budgets. VGRS lies on or above the BoN Pareto frontier for every k on both models.

VGRS Pareto frontier. Figure 1 shows the coverage curves with ±std bands over seeds. On IPO, VGRS@k lies above BoN@k across the budget range and reaches BoN@16 accuracy at $k \approx 13$ (∼81% of tokens, ∼19% saving). On RLOO, VGRS@k strictly dominates BoN@k for every k, reaching BoN@16 accuracy at $k \approx 9$ (∼58% of tokens), and the final VGRS@16 point lies clearly above BoN@16.



(a) IPO: online search operating point vs. BoN budget curve.



(b) RLOO: online search underperforms BoN at matched budget.

Figure 2: Best-of-N accuracy as a function of token budget (blue curve, mean ± std over seeds) with the online verifier search operating point overlaid (red, with error bars). At a matched budget online search sits on or below the BoN curve for IPO and well below it for RLOO.

Figure 2 overlays the online search operating point on the Best-of-N budget curve. At matched spend, online search sits on or below BoN for IPO and well below it for RLOO.

5.2 Qualitative Analysis

Why online search works for IPO but not RLOO. IPO is trained with step-level preference supervision, which encourages the model to write explicit intermediate arithmetic steps in its chain-of-thought. When online search prompts the IPO model with a partial state (e.g., remaining numbers $[3, 7, 50]$, target 56), it reliably produces lines like $3 + 7 = 10$ or $50 - 7 = 43$, which the text-to-move extractor mines as valid moves. The search tree is well-populated at every depth.

RLOO, by contrast, is trained to maximize final answer reward over complete trajectories. The model learns to write fluent full solutions with implicit reasoning. This means rather than $44 + 19 = 63$, it might write "I can combine the numbers to approach 56 by using subtraction and addition". When prompted mid-trajectory for a next step, RLOO produces similarly vague proposals. ExtractValidMoves finds few or no valid moves at the root, starving the search frontier immediately. This is a training objective mismatch, not necessarily a verifier issue.

Why VGRS recovers for RLOO. VGRS never prompts the model in an unfamiliar way. Each RLOO rollout is a complete solution attempt, sampled under the model's training distribution. Even if a given rollout's final answer is wrong, it typically contains one or two correct intermediate steps embedded in the prose. Across 16 rollouts, the root move pool accumulates roughly 10–20 verified moves (mean ∼15 on our benchmark). The symbolic search then combines these fragments freely, recovering solutions that no individual rollout assembled correctly.

Recombination example. Consider numbers $\{3, 4, 6, 8\}$ and target $\mathbf{t} = 24$. Across 16 rollouts, no single attempt assembles the full solution, but verified steps appear in different rollouts:

- Rollout 1 writes $3 \times 4 = 12$, then continues incorrectly ($12 + 6 = 18$, $18 + 8 = 26$).
- Rollout 2 writes $8 - 6 = 2$, then continues incorrectly ($2 + 3 = 5$).
- Rollout 7 mentions $12 \times 2 = 24$ in its reasoning but emits a wrong <answer>.

Best-of-N scores each rollout independently and never finds a perfect answer. VGRS mines $3 \times 4 = 12$, $8 - 6 = 2$, and $12 \times 2 = 24$ into the move pool and symbolically chains them into $\big((3 \times 4) \times (8 - 6)\big) = 24$—a solution no individual rollout assembled correctly.

Failure cases. VGRS fails when the move pool is too sparse to cover any complete solution path. This occurs when: (a) the problem requires an unusual sequence that no rollout approximated even partially; (b) all rollouts make the same systematic error (e.g., all attempt the same incorrect first step); or (c) the problem is genuinely hard and BoN@16 itself would fail. On our 50-problem benchmark, VGRS reaches 0.807 for IPO (vs. BoN@16 0.793) and 0.780 for RLOO (vs. 0.733): cross-rollout recombination adds a small amount of coverage beyond individual rollouts for IPO and a larger, seed-consistent gain ($+0.047 \pm 0.012$) for RLOO, where rollouts more often contain complementary-but-incomplete reasoning fragments.

6 Discussion

Limitations. VGRS's core limitation is its dependence on a symbolic verifier and structured state representation. Countdown is unusual in that every intermediate step has a ground-truth checkable answer. Extending VGRS to open-ended mathematical reasoning (where intermediate steps may be valid but not mechanically verifiable) requires either a learned process reward model or a more constrained task format. The method also inherits any systematic biases in the base policy. That is, if all rollouts make the same early error, the move pool will not contain the correct first step and VGRS will fail.

Broader impacts. Reducing the token cost of accurate reasoning has direct practical implications for LLM deployment efficiency. VGRS offers a principled way to extract more value from already-generated rollouts without incurring additional inference costs, which is directly relevant to production systems where inference budget is a primary constraint. The technique is most naturally applicable to any task with a cheap symbolic verifier, like formal verification or code execution.

Difficulties encountered. The hardest systems issue was turning messy 0.5B rollouts into reliable symbolic moves. The policy was SFT'd to emit long <redacted_thinking> chains and a single <answer>, not a rigid per-step API, so early parsers that expected clean line-by-line outputs missed most usable steps. The final design mines every a op b = c triple from free-form prose with regex and then lets the symbolic verifier (match_triple_to_move) reject bad candidates: operands not currently available, double-spent numbers, or arithmetic that does not match the claimed result. This still cannot recover steps written with Unicode operators ($\times$, $\div$) or approximate equalities ($\approx$), which appear frequently in real rollouts; those lines are silently dropped rather than trusted.

A second difficulty was that online verifier-guided search is out-of-distribution for the trained policy. Stepwise proposal prompts differ from the full-solution format seen during SFT/IPO/RLOO, and the model often rambles without proposing valid next moves. On our 50-problem benchmark, beam search at matched compute underperformed Best-of-N: IPO reached 0.613 vs. 0.793 (BoN@16), and RLOO only 0.580 vs. 0.733 despite spending similar token budgets ($\sim$3.6k vs. $\sim$5.9k tokens/problem). This pushed us toward VGRS, which reuses the same in-distribution BoN rollouts and only adds offline symbolic recombination, recovering cases where the model wrote correct intermediate steps but produced a malformed or incorrect final <answer>.

Finally, fair comparison required careful token accounting. Search batches proposal calls across problems, so we attribute each batched vLLM cost evenly across contributing states;

VGRS adds zero new generation by copying each problem's BoN generated_tokens into its record (run_recombination.py). On the training side, RLOO also required stability work: noisy policy-gradient loss, version-dependent vLLM logprob extraction for importance weights, and tuning entropy/KL so rollout accuracy improved without collapsing pass@16 coverage.

7 Conclusion

We introduced Verifier-Guided Recombination Search (VGRS), a training-free, zero-additional-cost inference strategy that extracts and recombines verified arithmetic reasoning fragments from Best-of-N rollouts. Our central finding is that BoN rollouts collectively contain substantially more correct reasoning than their final answers convey. VGRS mines this latent signal and constructs solutions that never appeared in any individual generation. On Countdown with Qwen2.5-0.5B, VGRS strictly improves over BoN@16 for RLOO (0.780 vs. 0.733, gain $+0.047 \pm 0.012$ across seeds) and matches BoN@16 for IPO at zero additional token cost, reaching BoN@16 accuracy with $\sim$58% (RLOO) / $\sim$81% (IPO) of the budget. VGRS lies on or above the BoN Pareto frontier for every rollout count on both models.

More broadly, the results suggest that what to do with failed rollouts is an underexplored axis of test-time compute scaling. Independent sampling treats each rollout as a fresh attempt. VGRS treats the collection of rollouts as a shared knowledge base of verified reasoning fragments. Future work should explore extending this paradigm to tasks with learned (rather than symbolic) verifiers, combining online and offline recombination, and applying reasoning recombination to broader mathematical and code reasoning domains.

8 Team Contributions

Note that all group members assisted primary implementers.

- Shyam Sai Bethina: Primary implementer of the extension
- Sahil Koita: Primary implementer of the IPO/RLOO milestone
- Ananya Ganapathi: Primary implementer of the SFT checkpoint and poster layout

References

Xinyun Chen et al. Can LLMs collaborate on reasoning trajectories? arXiv preprint arXiv:2510.06410, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. arXiv preprint arXiv:2305.20050, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 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. Tree of thoughts: Deliberate problem solving with large language models. arXiv preprint arXiv:2305.10601, 2023.

Lunjun Zhang, Hritik Bansal, Mehran Kazemi, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. arXiv preprint arXiv:2408.15240, 2024.