ON THE LIMITS OF TEST-TIME COMPUTE FOR SELF-EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-verification and aggregation have emerged as popular approaches for bridging the gap between $\mathrm{pass@1}$ and $\mathrm{pass@}N$. However, increasing the number of candidates does not consistently narrow the gap between final-answer accuracy and $\mathrm{pass@}N$. To understand this limitation, we use ranking error to measure how reliably models distinguish their correct generations from incorrect ones. We show that ranking error admits a bias-variance decomposition, with both components shaped by algorithmic design choices and inference compute. Across our experiments, additional compute generally reduces variance, while bias initially decreases before approaching a plateau. Guided by this analysis, we introduce SCORPIO, a *simple* and *general* method that jointly scores randomly grouped candidates. Across math, code, and puzzle benchmarks, SCORPIO improves verification, tree search, and aggregation. For verification, SCORPIO improves accuracy over the strongest baseline by up to 7 percentage points while matching or exceeding that baseline's peak accuracy with 6–42% lower inference cost. For tree search, using SCORPIO to score partial solutions and select the final answer improves accuracy by up to 10 percentage points over prior methods. Finally, SCORPIO improves the scaling of aggregation-based methods by 12 percentage points on coding tasks.

1 INTRODUCTION

Scaling test-time compute has enabled substantial progress in the reasoning capabilities of large language models (LLMs) (OpenAI, 2026a; Anthropic, 2026). By spending more compute on a problem, models can explore alternate reasoning strategies (Wang et al., 2023), verify intermediate results (Yao et al., 2023), and revise earlier attempts before returning an answer (Madaan et al., 2023). These capabilities have enabled models to achieve gold-level performance in mathematics olympiads (Luong & Lockhart, 2025; Wei, 2025), complete long-horizon agentic tasks (Kwa et al., 2025), and more recently, propose a solution to an open Millennium Prize Problem (OpenAI, 2026b).

A common building block of these approaches is sampling multiple candidate solutions (Gu et al., 2026), which increases the chance of obtaining at least one correct answer compared to a single attempt. The resulting gap between the accuracy of a single attempt and the probability that at least one of N attempts succeeds, the $\mathrm{pass@1}$ versus $\mathrm{pass@N}$ gap, remains significant even for frontier models on agentic tasks. On Terminal-Bench 4, GPT-6 Astra gains 13 points and Fable 5.1 gains 20.9 points when moving from $\mathrm{pass@1}$ to $\mathrm{pass@5}$ (Fig. 1(a)). Several approaches leverage increased compute to attempt to bridge this gap, ranging from verification to identify a correct candidate (Singh et al., 2026; Kwok et al., 2026) to iteratively synthesizing better solutions from a pool of existing candidates (Venkatraman et al., 2025). However, the ability of these methods to *select* a good candidate does not consistently scale with the number of candidates. As shown in Fig. 1(b) and Fig. 1(c): the gap between selection accuracy and oracle accuracy widens with the number of candidate, while aggregation-based methods show a decline in accuracy.

We investigate this issue by adapting the pairwise ranking loss from the ranking literature (Shivaswamy & Chandrashekar, 2021) to measure how accurately an LLM ranks its own solutions. We show that this loss admits a bias-variance decomposition, where bias captures systematic errors in pairwise judgments and variance captures fluctuations across repeated evaluations. This decomposition allows us to study how design choices in test-time scaling algorithms, such as the number of

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

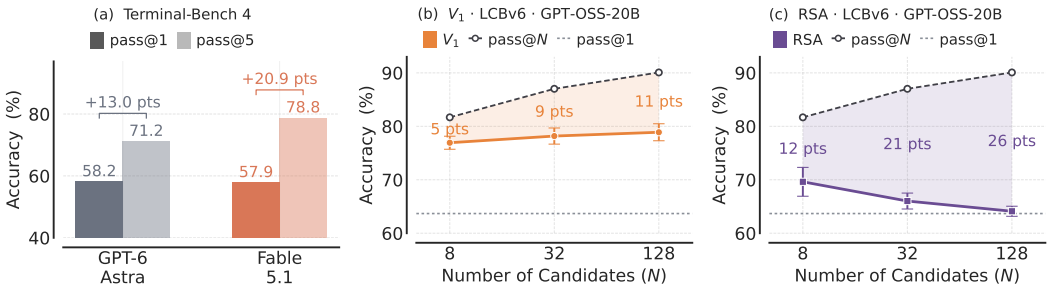

Figure 1: **Sampling more candidates improves the chance of a correct answer, but selecting it does not scale well with N. (a)** The gap between pass@1 and pass@N exists even for frontier models on agentic tasks.[1] **(b)** As N increases, the gap between pass@N and the selection accuracy of V_1 (Singh et al., 2026), a recent self-verification method for scaling test-time compute, widens. **(c)** A similar trend holds for Recursive Self-Aggregation (Venkatraman et al., 2025). For (b) and (c), we run each method until accuracy plateaus; refer App. E.1 for accuracy versus compute curves.

candidates evaluated in a single LLM call and how they are grouped, affect ranking quality. We find that variance generally decreases with inference compute, while bias appears to plateau within the evaluated budgets. This residual bias captures errors that persist even at high inference budgets, making bias reduction a key step toward closing the gap between selection accuracy and oracle performance.

Guided by this analysis, we introduce Scoring Candidates Over Random Partitions (SCORPIO), a simple scoring strategy that repeatedly partitions candidates into random groups and scores them jointly. This simple design reduces both bias and variance relative to prior scoring methods, resulting in lower ranking error and better scaling with test-time compute. Across math, code, and puzzle benchmarks, SCORPIO improves peak verification accuracy over the strongest baseline by up to **7** percentage points, while matching or exceeding that baseline's peak accuracy with **6–42%** lower total inference cost. Beyond verification, SCORPIO can also be applied to other test-time scaling methods. For tree search, it effectively identifies promising partial solutions to continue, improving over the strongest baseline by **10** percentage points on LiveCodeBench v6. Further, SCORPIO improves the test-time scaling of aggregation-based methods, reversing the decline in accuracy with larger candidate pools highlighted in Fig. 1(c). Despite these improvements, the residual bias identified above persists even under SCORPIO, leaving a gap between selection accuracy and pass@N at high inference budgets.

2 Related Work

LLMs as judges. LLMs are widely used to judge the quality of model-generated responses (Zheng et al., 2023). More recently, these judgments have provided a promising axis for scaling test-time compute on reasoning tasks, where repeated evaluations have been shown to improve the ability to select a correct solution from a set of generated candidates (Singh et al., 2026; Kim et al., 2026; Kwok et al., 2026). However, prior work has also documented limitations of LLM judges, including preferences for their own outputs (Panickssery et al., 2024; Yang et al., 2026), sensitivity to candidate order (Panickssery et al., 2024), and inconsistencies across repeated judgments (Haldar & Hockenmaier, 2025; Yagubyan, 2026; Norman et al., 2026). Our work examines both the potential and the limitations of self-verification through a bias–variance decomposition of ranking error. This decomposition offers a unified account of these limitations: inconsistency across repeated judgments corresponds to the variance term, while systematic preferences such as self-preference and order sensitivity plausibly manifest as bias that additional compute cannot remove. We find that a residual bias remains as inference compute increases, while variance decreases monotonically. While prior work has discussed variance reduction (Kwok et al., 2026), our work provides a more nuanced view by highlighting that the rate of variance reduction depends on algorithmic design choices. For instance, we observe slower variance reduction with inference compute when pairing similar candidates, as in V_1 (Singh et al., 2026), than when pairing candidates randomly.

[1] Accuracies from the Terminal-Bench 4 leaderboard.

Bias-variance decomposition of losses. Bias-variance decompositions are a standard tool for analyzing the tradeoffs involved in training machine learning models (Geman et al., 1992). Such decompositions have been developed for squared loss (Bishop, 2006), zero-one loss (Kohavi & Wolpert, 1996; Domingos, 2000), and ranking objectives (Shivaswamy & Chandrashekar, 2021). We extend this perspective to test-time scaling by studying how bias and variance in rankings evolve as a function of inference compute. In this setting, stochasticity arises from temperature-based sampling by an LLM judge, rather than from training predictors on sampled training data. Finally, we use this decomposition to motivate simpler design choices in recent test-time algorithms, showing that random pairing outperforms tournament-based pairing at equal compute. App. B provides a more nuanced distinction from algorithmically similar approaches in the retrieval community.

3 Background

We consider three settings for scaling test-time compute: selecting among complete solutions, guiding search over partial solutions, and aggregating multiple solutions to generate new ones. Given a problem x, a language model π_θ generates a solution $y_{1:T} \sim \pi_\theta(\cdot \mid x)$, where T denotes its length in tokens. Sampling N such solutions produces a candidate pool $\mathcal{C} = \{y^{(1)}, \ldots, y^{(N)}\}$.

Candidate scoring. A scoring procedure f assigns a score to each candidate in $\mathcal{C}$. We focus on self-evaluation, where the generating model π_θ judges its own responses using a prompt P_{judge}. Candidates can be evaluated independently through pointwise scoring (Weng et al., 2023), yielding $s^{(i)} \sim \pi_\theta(\cdot \mid P_{\text{judge}}, x, y^{(i)})$, or jointly within a shared context (Singh et al., 2026), yielding $(s^{(1)}, \ldots, s^{(N)}) \sim \pi_\theta(\cdot \mid P_{\text{judge}}, x, \mathcal{C})$. In either case, more inference compute can be spent by repeating evaluations and combining the resulting scores (Kwok et al., 2026). For instance, in pointwise scoring, the final score can be computed as $\hat{s}^{(i)} = \frac{1}{M}\sum_{m=1}^{M} s_m^{(i)}$, where $s_m^{(i)}$ is the score from the m-th evaluation. This requires $M \cdot N$ LLM calls to evaluate all N candidates M times. The highest-scoring candidate is then selected, with $i^* = \arg\max_i \hat{s}^{(i)}$.

Candidate search. The same scoring procedure can be used during generation to evaluate partial solutions $y_{1:t}$ ($t < T$). Search proceeds in stages: at each stage, the retained partial solutions are extended, scored, and pruned. Beam search (Lowerre, 1976) retains a fixed number of top-scoring candidates, whereas we retain a fixed fraction ρ of them, as described in Sec. 5.2.

Candidate aggregation. Aggregation generates new solutions by conditioning on existing candidates. Given a subset $\mathcal{S} \subseteq \mathcal{C}$ and a prompt P_{agg} instructing the model to combine and improve these solutions, a new solution is sampled as $y' \sim \pi_\theta(\cdot \mid P_{\text{agg}}, x, \mathcal{S})$. Repeating this operation over N randomly sampled subsets produces a new pool of N candidates, which can be used in subsequent aggregation rounds (Venkatraman et al., 2025; Madaan et al., 2025).

Across these settings, effective test-time scaling requires accurately ranking candidates. In selection, a ranking error affects only the final choice. In search and aggregation, errors compound: pruning a promising partial solution or retaining a poor candidate for aggregation affects all subsequent generations. In the next section, we define the pairwise ranking loss to measure ranking quality and analyze how its bias and variance change with inference compute.

4 Decomposing Ranking Error Into Bias and Variance

We measure how accurately a language model ranks its own generations by examining whether correct solutions receive higher scores than incorrect ones. Let $\mathcal{C}^+$ and $\mathcal{C}^-$ denote the subsets of correct and incorrect solutions in $\mathcal{C}$, such that $\mathcal{C} = \mathcal{C}^+ \cup \mathcal{C}^-$. Following Cortes & Mohri (2003), we define the pairwise ranking error of the scoring procedure f as:

$$\mathcal{R}(f) = \begin{cases} \dfrac{1}{|\mathcal{C}^+|\,|\mathcal{C}^-|} \displaystyle\sum_{y^{(i)} \in \mathcal{C}^-} \sum_{y^{(j)} \in \mathcal{C}^+} \mathbb{1}\left[\hat{s}^{(i)} \geq \hat{s}^{(j)}\right], & \text{if } |\mathcal{C}^+| > 0 \text{ and } |\mathcal{C}^-| > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Algorithm 1 SCORPIO Selection

Inputs: problem x; pool $\mathcal{C} = \{y^{(1)}, \ldots, y^{(N)}\}$; policy π_θ; group size K; repeats M; groupwise scorer f.
$\hat{s} \leftarrow [\,]$ ▷ candidate scores
Score Candidates:
for $m \leftarrow 1$ to M **do**
  $\sigma_m \leftarrow$ random permutation of $[N]$ ▷ independent partition per repeat
  for $j \leftarrow 1$ to N/K **do**
    $\mathcal{G} \leftarrow \sigma_m[(j-1)K + 1 : jK]$ ▷ Take the next K candidates indices in the shuffled order
    $\big(r_m^{(i)}\big)_{i\in\mathcal{G}} \leftarrow f\big(x, (y^{(i)})_{i\in\mathcal{G}}\big)$ ▷ K candidates scored together
  end for
end for
Select Candidate:
$\hat{s}^{(i)} \leftarrow \frac{1}{M}\sum_{m=1}^{M} r_m^{(i)}$ ▷ average each candidate's M ratings
$i^* \leftarrow \arg\max_{i\in[N]} \hat{s}^{(i)}$
return $y^{(i^*)}$

In other words, the ranking error is the fraction of correct–incorrect pairs for which f assigns a higher score to the incorrect solution. This error is stochastic because the scores used to rank candidates are outputs of temperature-based sampling from the LLM. Evaluating candidates jointly can introduce further stochasticity when pairings are randomized, since the same candidate may receive different scores depending on its opponent. We therefore analyze the expected ranking error $\mathbb{E}[\mathcal{R}(f)]$, where the expectation is over randomness in the scoring procedure. For each correct–incorrect pair, we define $Z_{ij} = \mathbb{1}[\hat{s}^{(i)} \geq \hat{s}^{(j)}]$. Since Z_{ij} is binary, $Z_{ij}^2 = Z_{ij}$. Adding and subtracting $(\mathbb{E}[Z_{ij}])^2$ gives

$$\mathbb{E}[Z_{ij}] = \mathbb{E}[Z_{ij}^2] - \big(\mathbb{E}[Z_{ij}]\big)^2 + \big(\mathbb{E}[Z_{ij}]\big)^2 = \mathrm{Var}(Z_{ij}) + \big(\mathbb{E}[Z_{ij}]\big)^2. \tag{2}$$

Averaging over all correct–incorrect pairs, we obtain

$$\mathbb{E}[\mathcal{R}(f)] = \frac{1}{|\mathcal{C}^+|\,|\mathcal{C}^-|} \sum_{y^{(i)}\in\mathcal{C}^-} \sum_{y^{(j)}\in\mathcal{C}^+} \left[\underbrace{\big(\mathbb{E}[Z_{ij}]\big)^2}_{\text{Squared bias}} + \underbrace{\mathrm{Var}(Z_{ij})}_{\text{Variance}} \right], \tag{3}$$

Recall that $Z_{ij} = 1$ when the incorrect solution is scored higher than the correct one. Its expectation measures how often this error occurs, so $(\mathbb{E}[Z_{ij}] - 0)^2$ captures the squared bias relative to the desired value of zero. Its variance measures how consistently the scoring procedure orders the pair, regardless of whether the ordering is correct. Writing $p_{ij} = \mathbb{E}[Z_{ij}]$ for the misranking probability of a pair, the two terms are p_{ij}^2 and $p_{ij}(1 - p_{ij})$. Although both are determined by p_{ij}, they respond differently to changes in it: variance vanishes as p_{ij} approaches either 0 or 1, whereas squared bias vanishes only as p_{ij} approaches 0. Averaged over pairs, a decrease in variance without a corresponding decrease in bias therefore indicates that the scoring procedure orders pairs more consistently, but that some pairs are consistently misranked.

Estimating bias and variance. At a given inference budget, we repeat the scoring procedure f R times to obtain R scores for each candidate. These scores yield R rankings over N candidates. We then estimate the misranking probability for each correct-incorrect pair as $\hat{p}_{ij} = \frac{1}{R}\sum_{r=1}^{R} Z_{ij}^{(r)}$, where $Z_{ij}^{(r)} = 1$ when the incorrect solution is scored higher than the correct one in the r-th ranking. The pair's squared bias and variance are estimated as $\hat{p}_{ij}^2$ and $\hat{p}_{ij}(1 - \hat{p}_{ij})$. To reduce the bias and variance of these estimators, we use $R = 20$ repetitions throughout our experiments. Averaging these estimates over all correct-incorrect pairs gives estimates of the squared bias and variance terms in Eq. 3. App. C.1 provides further details on how bias and variance are estimated.

5 SCORPIO: SCORING CANDIDATES OVER RANDOM PARTITIONS

While previous methods have focused on evaluating candidates individually or in pairwise comparisons, we argue that contrasting a larger group inside a single LLM call can reduce bias and variance.

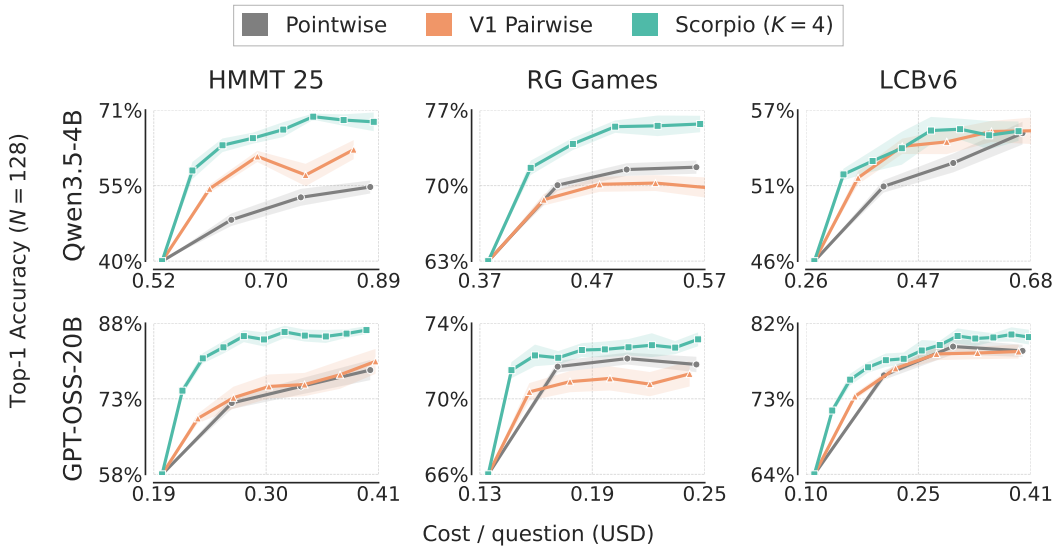

Figure 2: **SCORPIO scales better with inference compute than prior verification methods across math, code, and puzzle benchmarks.** We compare SCORPIO's scoring with V1 (Singh et al., 2026), a recent algorithm that scores candidates through pairwise comparisons between similar candidates, and pointwise scoring, where LLMs evaluate candidates in isolation. SCORPIO jointly scores candidates in groups of four. Inference compute is measured as the total dollar cost of all input and output tokens, and is scaled by repeating candidate evaluations. Results are averaged over 20 seeds, with shaded regions indicating 95% confidence intervals.

We propose SCORPIO (Algorithm 1), a simple instantiation of the scoring procedure f that randomly partitions a pool $\mathcal{C}$ of N candidates into groups of size K and *jointly* scores the candidates in each group in a single call. We run M rounds of scoring, creating fresh random partitions each round and using MN/K LLM calls in total.

Setup. We evaluate Qwen3.5-4B (Qwen Team, 2026) in instruct mode and GPT-OSS-20B (Agarwal et al., 2025) at medium reasoning effort on math with HMMT Feb 2025, code with LiveCodeBench (Jain et al., 2025), and general reasoning with Reasoning Gym games (Stojanovski et al., 2025). Following Hübotter et al. (2026), we consider the LCBv6 subset which contains 131 questions released between February and May 2025. For Reasoning Gym games, we use the 100-problem split from Venkatraman et al. (2025). Throughout our experiments, test-time compute is measured by the total dollar cost of all LLM calls. We use input and output prices of \$0.03 and \$0.10 per million tokens for GPT-OSS-20B, and \$0.20 per million tokens for both input and output for Qwen3.5-4B.[2] Further details on sampling parameters and cost calculations for each test-time scaling method are in App. D and evaluation prompts in App. F.

5.1 TEST-TIME SCALING FOR VERIFICATION

In self-verification, our goal is to select the single best candidate from a pool of $N = 128$ candidates generated by our model. The ultimate goal is to narrow the gap between pass@1 and $\text{pass@}N$. We compare against the standard pointwise approach, which generates a score for each sample $s^{(1)} \sim f(y^{(1)})$ and a recent state-of-the-art method, V1 (Singh et al., 2026) which combines pairwise comparisons $s^{(1)}, s^{(2)} \sim f(y^{(1)}, y^{(2)})$ with a tournament-style partitioning, iteratively comparing better-scoring candidates with each other. Following Singh et al. (2026), we cap the verification budget at the dollar cost of three rounds of pointwise scoring.

SCORPIO scales better than pointwise and tournament-based pairwise scoring. Fig. 2 compares SCORPIO with group size $K = 4$ against pointwise scoring and V_1. Across HMMT and Reasoning Gym games, SCORPIO improves peak accuracy over the strongest baseline by **1–7** percentage points and matches or exceeds that baseline's peak accuracy with **19–42%** lower inference

[2]Prices obtained from `https://pricepertoken.com` as of September 13, 2026.

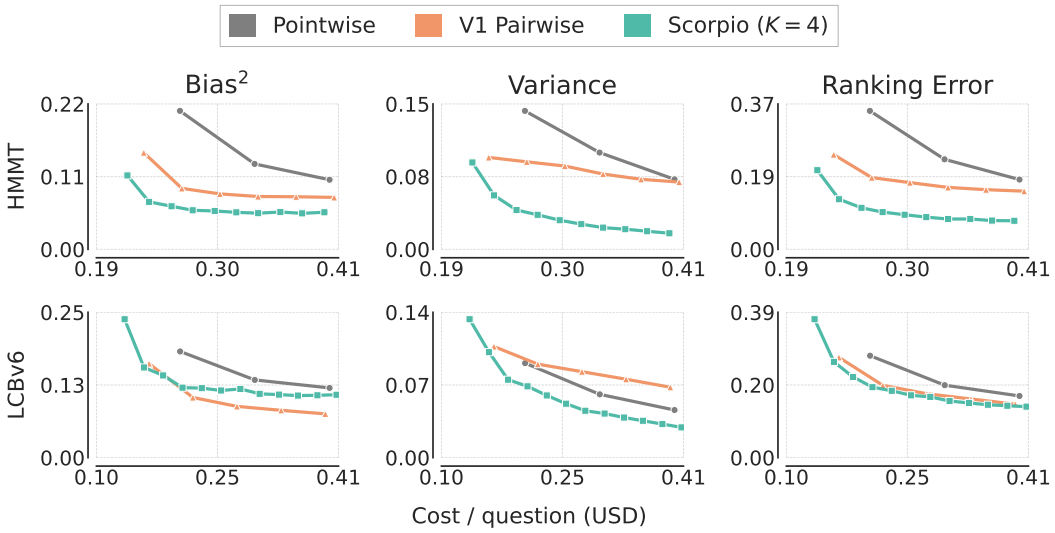

Figure 3: **Decomposition of ranking error into bias and variance.** Results for GPT-OSS-20B on HMMT (math) and LCBv6 (code). As inference compute increases, variance generally decreases, while bias decreases initially before approaching a plateau (see App. C.3). SCORPIO reduces variance faster than pointwise scoring and V1's (Singh et al., 2026) pairwise scoring, while achieving similar or lower ranking error. Similar trends are observed for Qwen3.5-4B (see Fig. 14).

cost. On LCBv6, all methods attain similar peak accuracy, but SCORPIO matches or exceeds the strongest baseline's peak accuracy with **6–28%** lower inference cost. We attribute these gains to (1) LLM's ability to better compare candidates *within* a context than between separate LLM calls and (2) more efficient use of LLM context and calls by combining multiple inputs. SCORPIO also compares favorably with Probabilistic Pivot Tournament (PPT) (Kwok et al., 2026) (see Fig. 12).

Variance reduces with compute, but ranking bias plateaus. Despite the improvements provided by SCORPIO, its selection accuracy plateaus well below pass@128, leaving an average gap of **16.4** percentage points across the six settings in Fig. 2. To better understand why additional verification compute does not close this gap, we revisit the ranking error and its bias-variance decomposition introduced in Sec. 4. Fig. 3 compares SCORPIO, pointwise scoring, and V_1 across the two datasets using GPT-OSS-20B. Across the evaluated methods, variance generally decreases as inference compute increases, while bias typically decreases initially before approaching a plateau.[3] In other words, additional compute makes rankings more stable, but incorrect solutions can still receive higher scores than correct ones.

Random grouping reduces variance faster while maintaining comparable bias. While SCORPIO achieves similar or lower ranking error in Fig. 3, its bias-variance decomposition differs significantly from that of V_1. By pairing candidates based on score similarity, V_1 consistently shows higher variance than SCORPIO, likely because these comparisons are harder than random comparisons and yield less consistent rankings across evaluations. At the same time, the two methods achieve broadly comparable bias, with V_1 attaining slightly lower bias on LCBv6. To isolate the effect of the pairing strategy, we compare V_1 with SCORPIO at $K = 2$ and observe similar trends (Fig. 13), showing that the faster variance reduction is not solely due to larger group sizes.

Joint evaluation reduces ranking error, but larger groups can increase bias. Consistent with Singh et al. (2026) and Kwok et al. (2026), we find that evaluating candidates together improves ranking quality over evaluating them in isolation. In Fig. 4, we vary the group size $K \in \{1, 2, 4, 8\}$ on HMMT and LCBv6 while keeping the grouping strategy random. Pointwise scoring ($K = 1$) has higher bias than joint evaluation, suggesting that comparisons help the model distinguish correct solutions from incorrect ones (see App. C.2 for an example). Having multiple candidates in the context also reduces variance because fewer calls are needed per scoring round, allowing each candidate's score to be averaged over more evaluations within the same cost budget. However,

[3] See App. C.3 for a discussion of how ranking bias can decrease with inference compute.

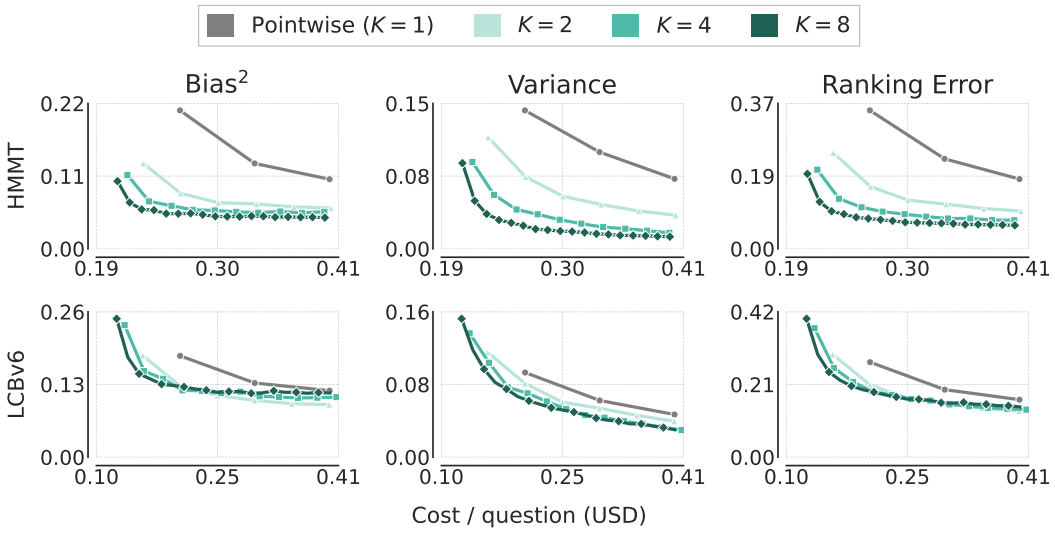

Figure 4: **Larger groups reduce variance faster but can increase bias.** Results for GPT-OSS-20B on HMMT (math) and LCBv6 (code). Increasing the group size allows more evaluations within the same cost budget, reducing variance faster. However, these gains can come at the expense of increased bias: on LCBv6, $K = 8$ yields higher bias than smaller groups. Similar trends are observed for Qwen3.5-4B (see Fig. 15).

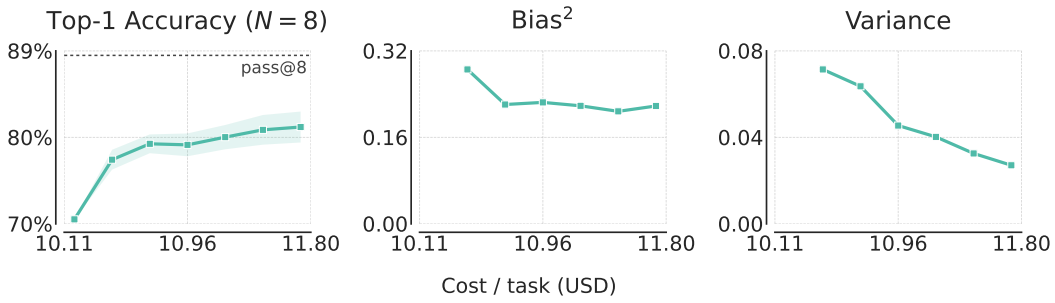

Figure 5: **Residual bias persists at scale with Gemini-3.1-Pro on Terminal-Bench-2.1.** We generate $N = 8$ candidates and score them with SCORPIO. Top-1 accuracy for SCORPIO ($K = 4$) plateaus about 8 percentage points below pass@8. SCORPIO's bias plateaus with compute, while its variance continues to decrease. Results are averaged across 5 seeds.

these variance reductions can come at the expense of bias: on LCBv6, increasing the group size to $K = 8$ raises bias, potentially because comparing more candidates in a single context is harder. We therefore use $K = 4$ as the default to balance these effects.

Residual bias persists at frontier scale. To examine whether the limitations of self-verification persist in a more capable model, we evaluate Gemini-3.1-Pro (Google DeepMind, 2026) on Terminal-Bench-2.1 (Merrill et al., 2026). We generate $N = 8$ responses per task and use SCORPIO to select the best candidate by comparing candidate summaries. As shown in Fig. 5, SCORPIO's selection accuracy plateaus 8 percentage points below pass@8. The middle and right panels show a similar trend: increasing inference compute reduces variance, while a residual bias remains. These results show that simple scoring methods can improve selection accuracy even at frontier scale and suggest that reducing residual ranking bias could help close the remaining gap.

5.2 TEST-TIME SCALING FOR TREE SEARCH

We apply SCORPIO to tree search to score partial solutions and identify promising trajectories. We use DeepConf's search procedure (Fu et al., 2025), replacing its confidence-based evaluation of partial trajectories with SCORPIO's scoring. The model evaluates groups of partial solutions and

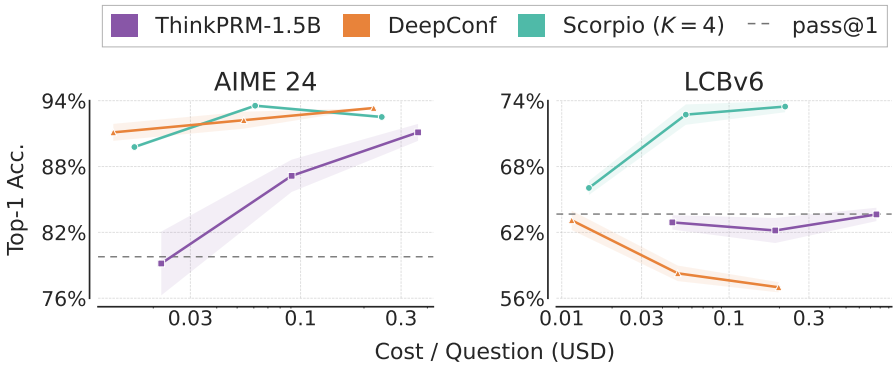

Figure 6: **Test-time search using SCORPIO matches or outperforms prior methods without any training.** Results for GPT-OSS-20B on AIME 2024 and LCBv6. We compare SCORPIO, which jointly scores partial solutions during search, with ThinkPRM-1.5B (Khalifa et al., 2025), which uses a trained process reward model, and DeepConf, which uses confidence scores derived from token probabilities. We vary the number of initial solutions: $N \in \{8, 32, 128\}$ for SCORPIO and ThinkPRM, and $N \in \{16, 64, 256\}$ for DeepConf to keep total costs comparable. Top-1 accuracy is plotted against total inference cost. Similar trends are observed for Qwen3.5-4B (see Fig. 17).

scores how likely each is to lead to a correct final answer (see Algorithm 3). We apply SCORPIO every 2,048 tokens. We compare SCORPIO with ThinkPRM (Khalifa et al., 2025), which uses a trained process reward model to score partial solutions pointwise, and with DeepConf.

Compute can be scaled along two axes: by increasing the number of initial solutions and by increasing the number of evaluations of each partial solution before pruning. For SCORPIO, we vary the initial pool size $N \in \{8, 32, 128\}$, use groups of size $K = 4$, and repeat the random grouping and scoring process three times ($M = 3$). At each pruning stage, the top $\rho = 0.75$ fraction of partial solutions is retained, while the rest are pruned. For each baseline, we follow the final-answer selection strategy reported in its original paper, with any adaptations described in App. D.4.

SCORPIO improves test-time search on math and code. Fig. 6 compares SCORPIO, DeepConf, and ThinkPRM on AIME 2024 and LCBv6 using GPT-OSS-20B. We vary $N \in \{8, 32, 128\}$ for SCORPIO and ThinkPRM, while allowing DeepConf larger pools of $N \in \{16, 64, 256\}$ to keep total costs in comparable ranges. On AIME 2024, where the model already achieves a high pass@1, DeepConf and SCORPIO attain similar peak accuracy, suggesting that inexpensive confidence estimates can be effective when models are already good at solving the task. On LCBv6, where pass@1 is lower, SCORPIO scales well with compute, exceeding the strongest baseline's peak accuracy by **10** percentage points. We believe that ThinkPRM's lower accuracy on code is primarily due to a distribution mismatch between its mathematical training data and the coding tasks evaluated here.

Repeated evaluation of partial solutions improves search accuracy. We now fix the number of starting solutions at $N = 128$ and vary the number of times each partial solution is scored, $M \in \{1, 2, 3\}$. As shown in Fig. 18, across three datasets and two models, spending more compute on state evaluation improves the accuracy of the selected answer by **2–6** percentage points.

These results show that search with SCORPIO can match or outperform ThinkPRM and DeepConf in our experiments, and that additional compute spent evaluating partial solutions can improve search accuracy. However, as in verification, a gap to pass@N persists after search.

5.3 TEST-TIME SCALING FOR AGGREGATION

Finally, we evaluate test-time scaling via aggregation, which iteratively generates N candidates and uses them as prompt-conditioning to generate N new, improved candidates. Following Venkatraman et al. (2025), we scale across $L = 1 \rightarrow 10$ rounds of generation and aggregation, and further scale compute by varying the number of candidates generated at each round, $N \in \{8, 32, 128\}$

We compare a standard aggregation method RSA (Venkatraman et al., 2025) with RSA augmented by SCORPIO scoring. Following Singh et al. (2026), we score the N candidates after generation with

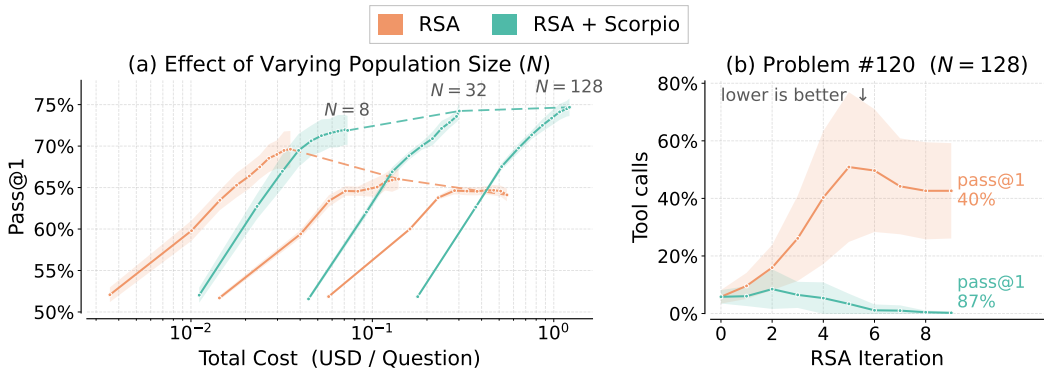

Figure 7: **Pruning candidates with SCORPIO allows RSA to scale with pool size N.** Using GPT-OSS-20B on LCBv6, we plot **(a)** accuracy across $L = 1 \rightarrow 10$ aggregation rounds for different pool sizes N. Increasing N degrades RSA performance, whereas RSA + SCORPIO improves with larger pools. One failure mode contributing to this degradation is shown in **(b)**: RSA amplifies rare incorrect responses over successive rounds on problem 120. LCBv6 requires a complete Python solution in a single turn and does not permit tool calls. However, RSA amplifies rare incorrect responses containing tool calls over successive rounds, with the effect becoming more severe as N increases (see Fig. 19). After each round, SCORPIO scores all N candidates, and only the top $N/2$ are used to construct aggregation sets for the next round, limiting this bad-mode amplification. Results are averaged over 8 seeds, with 95% CI.

SCORPIO and select the top $N/2$ for future prompt-conditioning. In other words, at each round, we build N inputs to the LLM using subsets sampled from top $N/2$ candidates from the previous round.

Standard RSA can degrade as the number of candidates N increases. In Fig. 7(a), we scale RSA over $L = 1 \rightarrow 10$ rounds and candidate pool sizes $N \in \{8, 32, 128\}$. Performance improves over rounds L, but scaling compute by increasing N from 8 to 128 lowers final accuracy by **6** percentage points despite using around $16\times$ more compute. To illustrate how this decline can arise, we examine a single LCBv6 problem (#120). Although the model is instructed to provide a complete Python solution in a single turn, it occasionally attempts tool calls to test its logic instead. These responses fail because the evaluation harness expects a complete program and does not support tool calls. When these tool-call responses are passed into subsequent aggregation rounds, this behavior becomes more common in subsequent rounds, despite instructions to provide a Python program. With $N = 128$, the proportion of responses containing tool calls grows from 5.8% of the initial pool to 40% of the final pool, shown in Fig. 7(b).For this problem, tool-call responses remain rare with $N = 8$ but become substantially more common with $N = 128$ (see Fig. 19 in Appendix).

RSA + SCORPIO improves scaling with N, limits the amplification of undesirable responses. RSA + SCORPIO selects the top $N/2$ candidates at each round to maintain a high-quality pool for the next round. For the problem examined in Fig. 7(b), this selection reduces tool-call responses to nearly zero and improves final accuracy from 40% to 87%. SCORPIO's scoring procedure incurs extra compute cost but allows RSA to scale well with N. As shown in Fig. 7(a), RSA + SCORPIO achieves a better compute–accuracy tradeoff and higher peak accuracy within the evaluated budgets.

6 CONCLUSION

In this work, we introduced a bias-variance decomposition to analyze how well LLMs evaluate their own solutions. We found that increasing test-time compute generally reduces variance, while residual ranking bias limits further improvements. Using this decomposition, we identified strengths and weaknesses of test-time scaling algorithms and introduced SCORPIO, a simple strategy that jointly scores candidates over random partitions. This strategy is effective across multiple test-time scaling settings, models, and benchmarks. More broadly, the decomposition provides a novel perspective on compute and algorithmic design choices and encourages future test-time scaling methods to improve judgments by reducing bias, variance, or both.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing and improve the style of plots. We did not use these tools to generate ideas or propose experiments. We reviewed all AI-assisted work and verified that grammatical changes preserved the intended meaning. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

App. D provides details of our method and the baselines used throughout the paper. App. F provide details of the prompts used for self-evaluation.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Anthropic. Introducing Claude Fable 5.1 and Claude Mythos 5.1, 2026. URL https://www.anthropic.com/claude-fable-and-mythos-5-1.

Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

Corinna Cortes and Mehryar Mohri. Auc optimization vs. error rate minimization. In S. Thrun, L. Saul, and B. Schölkopf (eds.), *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003. URL https://proceedings.neurips.cc/paper_files/paper/2003/file/6ef80bb237adf4b6f77d0700e1255907-Paper.pdf.

Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th international conference on machine learning*, pp. 231–238, 2000.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence. *arXiv preprint arXiv:2508.15260*, 2025.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

Google DeepMind. Gemini 3.1 Pro Model Card, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Xiangming Gu, Soham De, Larisa Markeeva, Petar Veličković, and Razvan Pascanu. Understanding performance gap between parallel and sequential sampling in large reasoning models. *arXiv preprint arXiv:2604.05868*, 2026.

Rajarshi Haldar and Julia Hockenmaier. Rating roulette: Self-inconsistency in LLM-as-a-judge frameworks. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 24986–25004, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1361. URL https://aclanthology.org/2025.findings-emnlp.1361/.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think. *arXiv preprint arXiv:2504.16828*, 2025.

Joongwon Kim, Wannan Yang, Kelvin Niu, Hongming Zhang, Yun Zhu, Eryk Helenowski, Ruan Silva, Zhengxing Chen, Srinivasan Iyer, Manzil Zaheer, et al. Scaling test-time compute for agentic coding. *arXiv preprint arXiv:2604.16529*, 2026.

Ron Kohavi and David Wolpert. Bias plus variance decomposition for zero-one loss functions. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pp. 275–283, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1558604197.

Anton Korikov, Pan Du, Scott Sanner, and Navid Rekabsaz. Batched self-consistency improves LLM relevance assessment and ranking. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 32687–32703, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1661. URL https://aclanthology.org/2025.emnlp-main.1661/.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long software tasks. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. Llm-as-a-verifier: A general-purpose verification framework, 2026. URL https://arxiv.org/abs/2607.05391.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Bruce T. Lowerre. *The HARPY speech recognition system*. PhD thesis, Carnegie Mellon University, 1976.

Thang Luong and Edward Lockhart. Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad. Google DeepMind Blog, July 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Lovish Madaan, Aniket Didolkar, Suchin Gururangan, John Quan, Ruan Silva, Ruslan Salakhutdinov, Manzil Zaheer, Sanjeev Arora, and Anirudh Goyal. Rethinking thinking tokens: Llms as improvement operators. *arXiv preprint arXiv:2510.01123*, 2025.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

Justin D Norman, Michael U Rivera, and D Alex Hughes. Reliability without validity: A systematic, large-scale evaluation of llm-as-a-judge models across agreement, consistency, and bias. *arXiv preprint arXiv:2606.19544*, 2026.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026a. URL https://openai.com/index/gpt-6-astra/.

OpenAI. On the Navier–Stokes millennium prize problem. OpenAI blog, https://openai.com/index/navier-stokes-solution/, September 2026b.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations. In *Advances in Neural Information Processing Systems*, volume 37, pp. 68772–68802, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Pannaga Shivaswamy and Ashok Chandrashekar. Bias-variance decomposition for ranking. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp. 472–480, 2021.

Harman Singh, Xiuyu Li, Kusha Sareen, Monishwaran Maheswaran, Sijun Tan, Xiaoxia Wu, Junxiong Wang, Alpay Ariyak, Qingyang Wu, Samir Khaki, et al. V_1: Unifying generation and self-verification for parallel reasoners. *arXiv preprint arXiv:2603.04304*, 2026.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, et al. Recursive self-aggregation unlocks deep thinking in large language models. *arXiv preprint arXiv:2509.26626*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Alexander Wei. OpenAI IMO 2025 proofs. GitHub repository, 2025. URL https://github.com/aw31/openai-imo-2025-proofs.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 2550–2575, 2023.

Abel Yagubyan. The coin flip judge? reliability and bias in llm-as-a-judge evaluation. *arXiv preprint arXiv:2606.13685*, 2026.

Jinming Yang, Zheng Hu, Chuxian Qiu, Zhenyu Deng, Xinshan Jiao, and Tao Zhou. Quantifying and mitigating self-preference bias of llm judges. *arXiv preprint arXiv:2604.22891*, 2026.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Peiwen Yuan, Shaoxiong Feng, Yiwei Li, Xinglin Wang, Boyuan Pan, Heda Wang, Yao Hu, and Kan Li. BatchEval: Towards human-like text evaluation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15940–15958, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.846. URL https://aclanthology.org/2024.acl-long.846/.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

A LIMITATIONS

Despite SCORPIO's improved test-time scaling, it does not eliminate bias in rankings, as a result of which a significant gap remains between its selection accuracy and the oracle accuracy, $\mathrm{pass@}N$. Addressing this residual bias remains an important direction for realizing the full benefit of sampling multiple candidates.

B EXTENDED RELATED WORK

Repeated Groupwise Comparisons using LLM Judges. Prior work has studied joint scoring and repeated evaluation for text quality assessment (Yuan et al., 2024) and passage relevance ranking (Korikov et al., 2025), finding improvements in judgment quality and efficiency. SCORPIO shares this principle of joint scoring with randomized contexts. However, our work differs in its objective, analysis, and applications. First, we study whether a model can identify correct solutions among its own generations, rather than rank passages by relevance to a query. Second, we use a bias and variance decomposition of ranking error to analyze repeated evaluation and grouping strategies. This analysis sheds light on how inference compute affects both the consistency and correctness of self-evaluation, and how these effects depend on algorithmic design choices. Third, we use SCORPIO to guide subsequent generation, beyond ranking a fixed candidate pool: during search, it selects partial solutions to continue; during aggregation, it selects solutions to condition the next round of generation.

C RANKING ERROR AND BIAS VARIANCE DECOMPOSITION

C.1 ESTIMATION OF BIAS AND VARIANCE FOR DIFFERENT METHODS

To estimate the bias and variance from $R = 20$ rankings at a given evaluation budget. For each incorrect–correct pair (i, j), we first estimate the probability of an incorrect ordering, then average the resulting squared bias and variance over all such pairs:

$$\hat{p}_{ij} = \frac{1}{R}\sum_{r=1}^{R} \mathbb{1}\left[\hat{s}^{(i,r)} \geq \hat{s}^{(j,r)}\right], \qquad y^{(i)} \in \mathcal{C}^-, \quad y^{(j)} \in \mathcal{C}^+. \tag{4}$$

$$\widehat{\mathrm{Bias}^2} = \frac{1}{|\mathcal{C}^+||\mathcal{C}^-|}\sum_{y^{(i)}\in\mathcal{C}^-}\sum_{y^{(j)}\in\mathcal{C}^+} \hat{p}_{ij}^2, \tag{5}$$

$$\widehat{\mathrm{Variance}} = \frac{1}{|\mathcal{C}^+||\mathcal{C}^-|}\sum_{y^{(i)}\in\mathcal{C}^-}\sum_{y^{(j)}\in\mathcal{C}^+} \hat{p}_{ij}(1-\hat{p}_{ij}). \tag{6}$$

where $\hat{s}^{(i,r)}$ is the score of candidate $y^{(i)}$ in the r-th ranking at the evaluation budget under consideration. Algorithms 2 describe how these rankings are obtained.

C.2 GROUPWISE COMPARISONS HELPS WITH LOWERING BIAS

Here, we examine responses to a single question to illustrate the benefit of SCORPIO's groupwise comparisons over pointwise evaluations. All excerpts below are from GPT-OSS-20B generations for problem index 0 in the HMMT-Feb 2025 dataset. In the verification setup, we start with $N = 128$ generations for this question. Of these, 45 yield the correct answer, while 67 yield the same incorrect answer. Thus, majority voting also does not select the correct answer.

Question and Answer

Given that x, y and z are positive real numbers such that

$$x^{\log_2(yz)} = 2^8 \cdot 3^4, \qquad y^{\log_2(zx)} = 2^9 \cdot 3^6, \qquad z^{\log_2(xy)} = 2^5 \cdot 3^{10},$$

compute the smallest possible value of xyz.

Algorithm 2 Estimating bias and variance

Require: Problem x, labeled candidate pool $\mathcal{C}$ of size N, scorer f, maximum evaluation level $L_{\max}$, target level $1 \le L \le L_{\max}$

Set $R = 20$ independent runs
Initialize score array $S \in \mathbb{R}^{R \times L_{\max} \times N}$
for $r = 1, \ldots, R$ **do**
    Initialize a fresh run of f on $(x, \mathcal{C})$
    for $\ell = 1, \ldots, L_{\max}$ **do**
        Advance f to evaluation level ℓ ▷ Continue the same run using its scoring history
        Store the current candidate scores in $S[r, \ell, :]$
    end for
end for
for $r = 1, \ldots, R$ **do**
    $(\hat{s}^{(1,r)}, \ldots, \hat{s}^{(N,r)}) \leftarrow S[r, L, :]$ ▷ Retrieve scores at the target level
end for
Compute $\hat{p}_{ij}$ using equation 4
Compute squared bias and variance using equation 5 and equation 6
return Estimated squared bias and variance

Answer: $\frac{1}{576}$.

Failure mode. Two solutions satisfy the constraints, one yielding $\frac{1}{576}$ (correct as it is smaller) and the other 576 (incorrect). They are related by reversing the signs of $\log_2 x$, $\log_2 y$, and $\log_2 z$. The incorrect reasoning begins by assuming $x, y, z > 1$, even though the question only requires x, y, z to be *positive*. This assumption excludes the negative logarithmic branch and overlooks the solution that yields $\frac{1}{576}$.

Pointwise evaluation does not spot the mistake. We track one such incorrect generation and score it 20 times under each scheme. Scored on its own it receives a rating of 10 on 19 of 20 evaluations (mean 9.95); scored inside random groups of $K = 4$ it receives a 1 or a 2 on 17 of 20 (mean 2.60).

Groupwise evaluation spots the mistake. We now show one SCORPIO call in which this generation is scored alongside three others. Solutions 1, 2 and 3 all answer 576 (incorrect one), while Solution 4 answers $1/576$ (correct). The verifier opens by comparing the final answers, and identifies the faulty assumption directly:

Verifier reasoning

"We need to see whether the smallest possible product is 576 or $1/576$. … But x, y, z must be positive real numbers. Negative exponents produce positive numbers: e.g., if $X = -2$, then $x = 2^{-2} = 1/4$ positive. So that's okay. … but the requirement says x, y, z positive real numbers, *not necessarily* > 1. So negative X allowed."

It then verifies that the negative branch satisfies the original equations, and assigns a rating of 1 to Solutions 1-3 and 10 to Solution 4. In other words, performing groupwise comparisons help reduce the bias while evaluating solutions.

C.3 WHY DOES BIAS INITIALLY DECREASE WITH COMPUTE?

As seen in Figure 3, bias initially decreases as inference compute increases. To understand why, consider a correct and an incorrect candidate whose score distributions overlap, with the correct candidate receiving a higher expected score, as illustrated in Fig. 8. At a low evaluation budget, fluctuations in these scores can cause the incorrect candidate to be ranked above the correct one, despite its lower expected score.

As the evaluation budget increases, each candidate's final score averages more judgments and concentrates around its mean. This makes an incorrect ordering less likely, reducing the pair's contri-

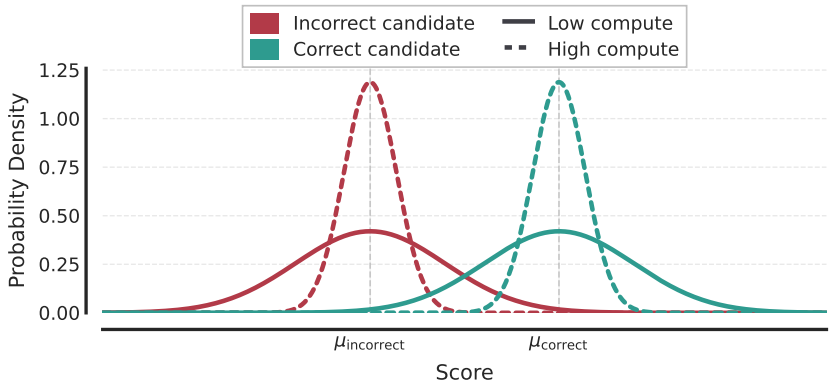

Figure 8: **Increasing compute does not change the mean of the score distribution, but can reduce the bias in rankings.** At low compute (solid curves), overlapping score distributions allow the incorrect candidate to receive a higher score than the correct one. Averaging more evaluations (dotted curves) concentrates the scores around their respective means. Since the correct candidate has a higher mean score in this example, this makes an incorrect ordering less likely, reducing the pair's squared bias $(\mathbb{E}[Z_{ij}])^2$.

bution $(\mathbb{E}[Z_{ij}])^2$ to the bias term even though the expected scores remain unchanged. Averaging does not, however, correct pairs for which the incorrect candidate has a higher expected score. For these pairs, concentrating the scores makes the incorrect ordering more likely, contributing to the bias floor observed at higher budgets.

D Experimental Details

D.1 Sampling Parameters and Budgets

We leverage vllm (Kwon et al., 2023) for LLM inference.

Sampling parameters. Following Singh et al. (2026), we set the temperature for GPT-OSS-20B to $T = 0.6$ for code generation and $T = 1.0$ for mathematics and Reasoning Gym tasks, with top-p sampling at $p = 0.95$ throughout. For Qwen3.5-4B, we use the developer-recommended sampling parameters for instruct mode across all tasks: $T = 1.0$, top-$p = 0.95$, top-$k = 20$, and a presence penalty of 1.5.

Budgets for the Judge. When scoring candidates, all methods are provided a maximum generation budget of 32,000 tokens per LLM call.

Prompts for scoring responses are provided in App. F.

D.2 Cost Calculation

In this section, we describe how the cost per question is calculated for each test-time scaling method discussed in the paper.

Verification. The reported cost includes both generating the N candidate solutions and verifying them through repeated evaluations. Let $\bar{g}_{\text{in}}$ and $\bar{g}_{\text{out}}$ denote the average input and output token counts per generation call, and let $\bar{v}_{\text{in}}$ and $\bar{v}_{\text{out}}$ denote the corresponding averages per verification call. For Scorpio with group size K, M scoring rounds require MN/K verification calls. The total cost per question is therefore

$$\text{Cost} = \frac{1}{10^6}\left[N\left(p_{\text{in}}\bar{g}_{\text{in}} + p_{\text{out}}\bar{g}_{\text{out}}\right) + \frac{MN}{K}\left(p_{\text{in}}\bar{v}_{\text{in}} + p_{\text{out}}\bar{v}_{\text{out}}\right)\right],$$

where p_{in} and p_{out} are the input and output prices in dollars per million tokens. Verification input counts include the judging prompt, the problem, and the candidate solutions presented in each call.

For other scoring methods, we use the actual number of verification calls in place of MN/K and compute $\bar{v}_{\text{in}}$ and $\bar{v}_{\text{out}}$ from that method's input and output token counts.

Search. For SCORPIO and ThinkPRM, cost accounting follows the same principle as verification, but also includes evaluations of partial solutions during search. As described in Sec. 5.2, trajectories are progressively pruned so that only a subset continues to completion. For a pruned trajectory, we count all generation and evaluation costs incurred before it is stopped. The total cost includes generation, intermediate evaluations, and verification of the completed solutions. For ThinkPRM, we use the same input and output token prices as the generating model. For DeepConf, judging intermediate states requires no additional model calls, so the cost includes only generation.

Aggregation. The total cost includes generating the initial pool of N solutions and the input and output token costs of every aggregation call across all rounds. When SCORPIO is used between rounds, we also include the cost of scoring candidates for selection.

D.3 Implementation Details of Verification Methods

V_1 (Singh et al., 2026). We use the authors' implementation. Following their setup, for GPT-OSS-20B, the judge receives the final-channel output on math and Reasoning Gym tasks, while on LCBv6, the code block is extracted and provided as input. For Qwen3.5-4B (in instruct mode), which does not separate reasoning traces into distinct channels, we summarize the outputs using the *same* model and provide these summaries to the judge. The additional summarization cost is *included* in *all* figures reported in the main and supplementary text.

Pointwise scoring. We use the implementation from the V_1 codebase, replacing a single evaluation with repeated evaluations. Each response is scored multiple times by the judge, and its final score is the average of the resulting scores.

LLM as a Verifier (PPT). (Kwok et al., 2026) For a fair comparison, we fix the evaluation criteria to be the same as those in our prompt ($C = 1$, as we use only a single evaluation criterion). We use the same rating scale described in their paper. In our setting, the generating and evaluating models are the same.

SCORPIO. Our implementation is based off the V_1 code base, with the only difference being we replace V_1's Swiss Tournament strategy with random grouping strategy, and instead of putting two elements in context together we put 4 elements in context. Prompts for scoring complete responses are provided in App. F.1

D.4 Implementation Details of Tree Search Methods

In this section, we describe the search algorithm used to obtain the results in Sec. 5.2. We adapt the search algorithm of Fu et al. (2025) to perform value estimation: intermediate states are provided as inputs to a scorer, which reasons about them and outputs a score for each state. This reasoning incurs additional evaluation costs, but accurate scores can reduce generation costs by allowing unpromising trajectories to be pruned well before completion. We use the same sampling parameters for tree search as outlined in App. D.1

Algorithm 3 describes the procedure. Starting from N trajectories, we alternate generation and state evaluation for $D = 4$ stages, generating up to $L = 2{,}048$ tokens per trajectory at each stage. Completed solutions are set aside, while the remaining partial solutions are scored and the lowest-scoring 25% are pruned ($\rho = 0.75$). After the final stage, surviving trajectories continue until completion or until the generation budget of $64{,}000$ tokens per trajectory is exhausted. The completed solutions are then scored to select the final answer. We now discuss details specific to each method below. We retain each baseline's original final-answer selection strategy to preserve its strengths.

SCORPIO. We use group size $K = 4$ and $M = 3$ scoring rounds for our main results. Intermediate state evaluations use the prompt in App. F.2, while final verification uses the prompt in App. F.1. We scale inference compute by increasing the initial pool size $N \in \{8, 32, 128\}$.

DeepConf (Fu et al., 2025). We use the authors' implementation in the online setup, with the generation budget and sampling parameters outlined above. Test-time compute is scaled by increasing the number of candidates $N \in \{16, 64, 256\}$. These larger values of N were chosen to roughly match

Algorithm 3 Tree Search with Intermediate Scoring

Inputs: problem x; policy π_θ; initial pool size N; scorer f; group size K; scoring repeats M; tree depth D; tokens per step L; retention fraction ρ.
$\mathcal{C} \leftarrow N$ empty trajectories
$\mathcal{C}_{\text{complete}} \leftarrow \varnothing$
for $d \leftarrow 1$ to D **do**
    Extend each trajectory in $\mathcal{C}$ by at most L tokens using π_θ
    Move completed solutions from $\mathcal{C}$ to $\mathcal{C}_{\text{complete}}$
    if $\mathcal{C} = \varnothing$ **then**
        break
    end if
    Score partial solutions in $\mathcal{C}$ using f with (K, M) and the prompt in App. F.2
    $\mathcal{C} \leftarrow$ the top $\lceil \rho |\mathcal{C}| \rceil$ partial solutions by score
end for
Continue the remaining trajectories in $\mathcal{C}$ until completion or the generation budget is exhausted
Add the resulting solutions to $\mathcal{C}_{\text{complete}}$
Score solutions in $\mathcal{C}_{\text{complete}}$ App. F.1
return the highest-scoring solution

the total costs of ThinkPRM and SCORPIO. The number of warmup traces is 4 for $N = 16$ and 16 for $N = 64$ and $N = 256$. We find that aggressive pruning ($\eta = 10$) works better for Qwen-3.5-4B, while conservative pruning ($\eta = 90$) works better for GPT-OSS-20B, and we use these settings. We also report results for both values of η in Fig. 16. For final answer selection, we follow the paper and assign each trajectory a confidence score equal to the confidence of its lowest-confidence window. For the math benchmarks (AIME 2024, HMMT 2025), these scores are used as weights in a weighted majority vote. For LCBv6, we select the surviving trajectory with the highest confidence.

ThinkPRM (Khalifa et al., 2025). Our search setup for ThinkPRM differs from the authors' original setup in two ways. First, their search uses double newlines to separate steps and a maximum tree depth of 20, whereas ours uses a tree depth of 4 with 2,048 tokens generated per step. Second, we replace beam search with the early-stopping procedure adapted from DeepConf. These design choices were made to keep search-based strategies competitive in cost with verification-based methods, which generate complete solutions and then verify them. Building deeper trees with more frequent intermediate state evaluations, as in the original ThinkPRM setup, would incur substantially higher costs. We set $\rho = 0.75$, retaining the top 75% of partial solutions by score. For math tasks, we find that weighted majority voting works better than Best-of-N selection. For coding tasks, we use Best-of-N selection, as weighted majority voting is not possible.

E ADDITIONAL RESULTS

E.1 SELECTION ACCURACY VERSUS THE PASS@N GAP

In this section, we expand on the results in Fig. 1(b) and Fig. 1(c), which report the final accuracies of V_1 (Singh et al., 2026) and RSA (Venkatraman et al., 2025) at different population sizes. To assess whether accuracy plateaus within the reported budget, we examine accuracy as a function of inference budget for GPT-OSS-20B on LCBv6. For V_1, Fig. 9 shows that accuracy plateaus for all values of N, with little improvement at higher budgets. Similar observations hold for RSA in Fig. 10. We also report results on HMMT 2025 in Fig. 11 where a significant gap remains between the model's selection accuracy and oracle accuracy.

E.2 SCALING TEST-TIME COMPUTE ON VERIFICATION

In this section, we discuss additional results for scaling verification compute.

Comparison with PPT (Kwok et al., 2026). In Fig. 12, we compare PPT with SCORPIO ($K = 4$) on HMMT Feb 2025 and LCBv6 using GPT-OSS-20B and $N = 128$ candidates. On HMMT,

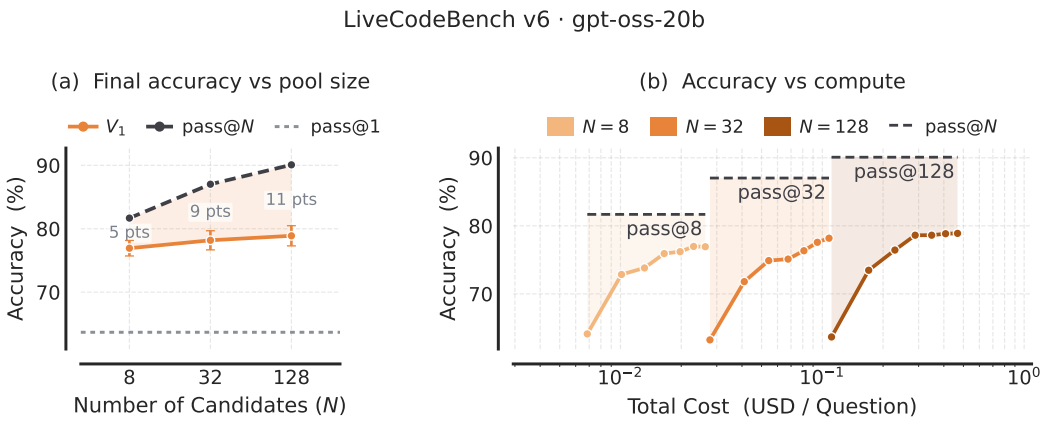

Figure 9: V_1 (Singh et al., 2026) **accuracy plateaus well below** pass@N**, and the gap widens as** N **increases.** Results for GPT-OSS-20B on LCBv6. **(a)** Increasing the candidate pool from $N = 8$ to $N = 128$ yields modest gains in selection accuracy, while the gap to pass@N grows from roughly 5 to 11 percentage points. Points show means over 20 seeds, with error bars indicating 95% confidence intervals. **(b)** Accuracy plateaus as inference compute increases for each pool size, remaining below the corresponding pass@N ceiling (dashed lines). Inference compute is measured as the total cost of the input and output tokens of each LLM call; see App. D.2 for further details.

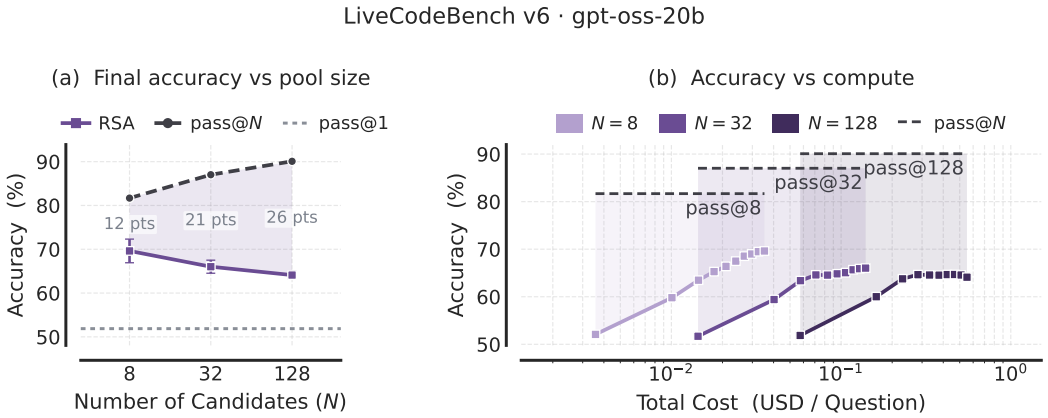

Figure 10: **RSA's (Venkatraman et al., 2025) final accuracy decreases as the candidate pool grows.** Results for GPT-OSS-20B on LCBv6. **(a)** Increasing N from 8 to 128 lowers final accuracy, widening the gap to pass@N from roughly 12 to 26 percentage points. Points show means over 8 seeds, with error bars indicating 95% confidence intervals. **(b)** Each curve tracks accuracy over successive aggregation rounds for a fixed N. Larger pools plateau at lower accuracy despite higher compute costs. Inference compute is measured as the total cost of the input and output tokens of each LLM call; see App. D.2 for further details.

SCORPIO improves peak selection accuracy by **11** percentage points over PPT. On LCBv6, SCORPIO achieves similar accuracy to PPT at **2.3**× lower total cost.

Comparing V_1's pairing strategy with random pairing strategy. To isolate the effect of pairing strategy from group size, Fig. 13 compares V_1 with SCORPIO at $K = 2$ using GPT-OSS-20B. Both methods evaluate two candidates per call, but V_1 pairs candidates based on score similarity, whereas SCORPIO pairs them randomly. Across all three datasets, random pairing reduces variance faster as compute increases.

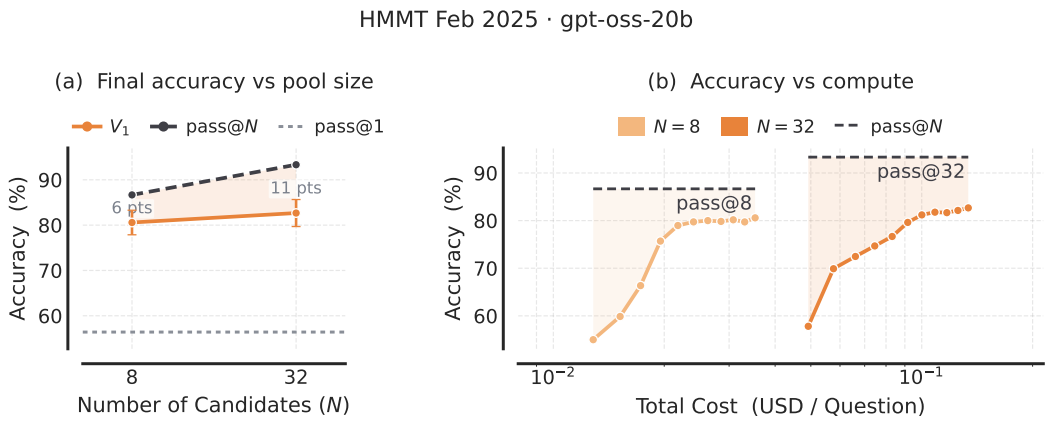

Figure 11: V_1 (Singh et al., 2026) **also exhibits a widening selection gap on HMMT Feb 2025.** Results for GPT-OSS-20B. **(a)** Increasing N from 8 to 32 yields modest gains in selection accuracy, while the gap to pass@N grows from roughly 6 to 11 percentage points. **(b)** Selection accuracy plateaus as inference compute increases, remaining below the corresponding pass@N ceiling (dashed lines). Inference compute is measured as the total cost of the input and output tokens of each LLM call; see App. D.2 for further details.

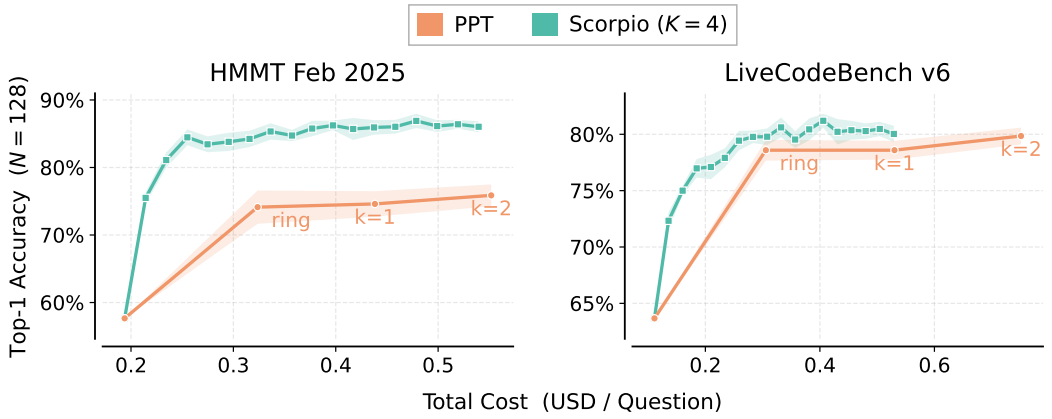

Figure 12: **SCORPIO achieves higher accuracy on math and similar accuracy at lower cost on code compared with PPT (Kwok et al., 2026).** Selection accuracy versus total inference cost for GPT-OSS-20B with $N = 128$ candidates. On HMMT Feb 2025, SCORPIO ($K = 4$) improves peak accuracy by **11** percentage points over PPT. On LCBv6, it achieves similar accuracy at **2.3**× lower total cost. PPT's strategy proceeds in two phases: an initial ring phase ensures that every candidate is evaluated, followed by a tournament phase. Points labeled "ring" show accuracy after the first phase, while $k = 1$ and $k = 2$ indicate whether one or two evaluations are used per pair. Curves show mean accuracy, with shaded regions indicating 95% confidence intervals. Inference compute is measured as the total dollar cost of all input and output tokens across LLM calls; see App. D.2 for details.

E.3 RANKING QUALITY FOR QWEN3.5-4B (INSTRUCT MODE)

In this section, we present results for the bias-variance decomposition for Qwen3.5-4B.

Bias-variance decomposition. Fig. 14 shows the bias-variance decomposition of ranking error for Qwen3.5-4B. Joint evaluation reduces bias compared with pointwise scoring. While V_1 attains lower bias than SCORPIO on HMMT and LCBv6 at higher budgets, SCORPIO reduces variance faster, achieving similar or lower ranking error.

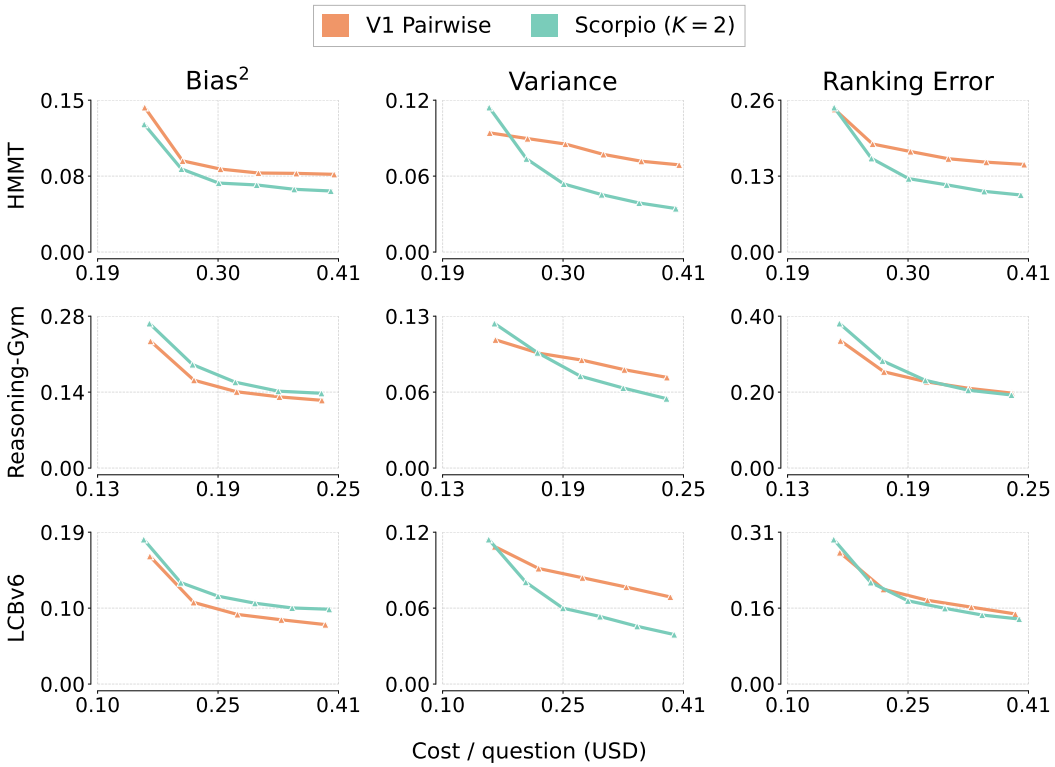

Figure 13: **Random pairing matches or outperforms tournament-based pairing.** Results for GPT-OSS-20B on HMMT, LCBv6 and Reasoning-Gym-Games. Bias-variance decomposition of ranking error for V_1 (Singh et al., 2026) and SCORPIO ($K = 2$) using GPT-OSS-20B. Across all three datasets, variance decreases quicker with SCORPIO as compute increases. V_1 attains lower bias on Reasoning Gym and LCBv6, while SCORPIO attains lower bias on HMMT. At higher budgets, SCORPIO achieves similar or lower ranking error, demonstrating that a simpler pairing strategy can be equally or more effective than tournament based strategies. Inference compute is measured as the total dollar cost of all input and output tokens across LLM calls; see App. D.2 for details.

Larger groups help with variance reduction but can hurt bias. Fig. 15 shows how group size affects bias and variance for Qwen3.5-4B. Larger groups allow more evaluations within the same cost budget, helping reduce variance. However, increasing group size does not always reduce bias, highlighting the tradeoff between obtaining more score estimates and comparing more candidates in a single context.

E.4 SCALING TEST TIME COMPUTE USING TREE-SEARCH

In this section, we present additional results on scaling the compute spent on test-time search.

DeepConf's (Fu et al., 2025) pruning threshold depends on the model. Fig. 16 compares aggressive ($\eta = 10$) and conservative ($\eta = 90$) pruning on HMMT Feb 2025 and LCBv6. Aggressive pruning generally works better for Qwen3.5-4B, while conservative pruning works better for GPT-OSS-20B. We therefore use $\eta = 10$ for Qwen3.5-4B and $\eta = 90$ for GPT-OSS-20B in our results.

SCORPIO improves test-time search on Qwen3.5-4B. Fig. 17 compares SCORPIO, ThinkPRM, and DeepConf on AIME 2024 and LCBv6. On AIME 2024, DeepConf and SCORPIO attain similar peak accuracies, with SCORPIO exceeding ThinkPRM by 6 percentage points. On LCBv6, SCORPIO improves peak accuracy over the strongest baseline by 12 percentage points.

Repeated evaluation of partial solutions improves search accuracy. In Fig. 18, we fix the number of solutions we start with at $N = 128$ and vary the number of times the random grouping and scoring process in Algorithm 3 is repeated, $M \in \{1, 2, 3\}$. Across both models and all three datasets,

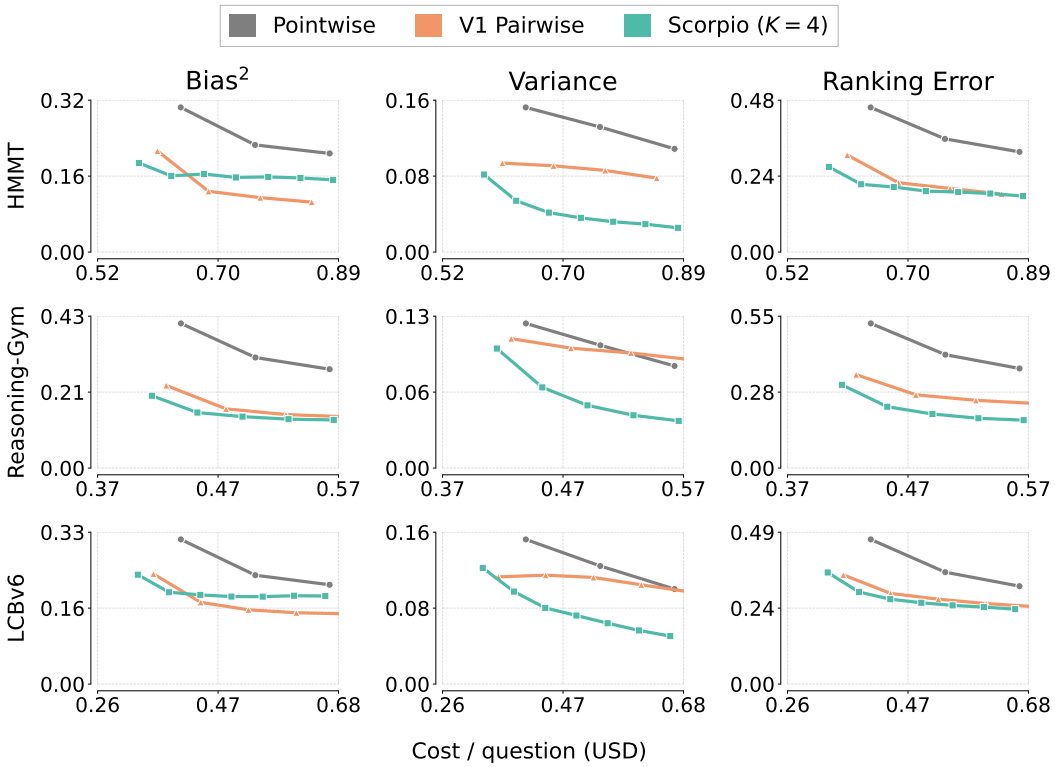

Figure 14: **Decomposition of the ranking error as a bias term and a variance term.** Results for Qwen3.5-4B on HMMT, Reasoning-Gym and LCBv6. While the variance term generally vanishes with increased compute, the bias term converges to a plateau (see: App. C.3). SCORPIO's groupwise scoring leads to a higher variance reduction rate than pointwise or V1's (Singh et al., 2026) pairwise scoring. Compute is measured as the total dollar cost of all input and output tokens across LLM calls; see App. D.2 for further details.

increasing M from 1 to 3 improves accuracy by 2 to 6 percentage points, showing that spending additional compute on state evaluation improves the accuracy of test-time search.

E.5 SCALING TEST-TIME COMPUTE USING AGGREGATION

In this section, we examine how candidate pool size affects the amplification of tool-call responses in RSA and how SCORPIO limits this behavior.

Larger pools could potentially amplify tool-call responses in RSA, while SCORPIO can limit their amplification. Fig. 19 shows that tool-call responses remain uncommon with $N = 8$ but become increasingly common across RSA rounds with $N = 32$ and $N = 128$. In RSA + SCORPIO, scoring candidates after each round and retaining the top half for subsequent aggregation reduces the proportion of these responses to nearly zero across all three pool sizes.

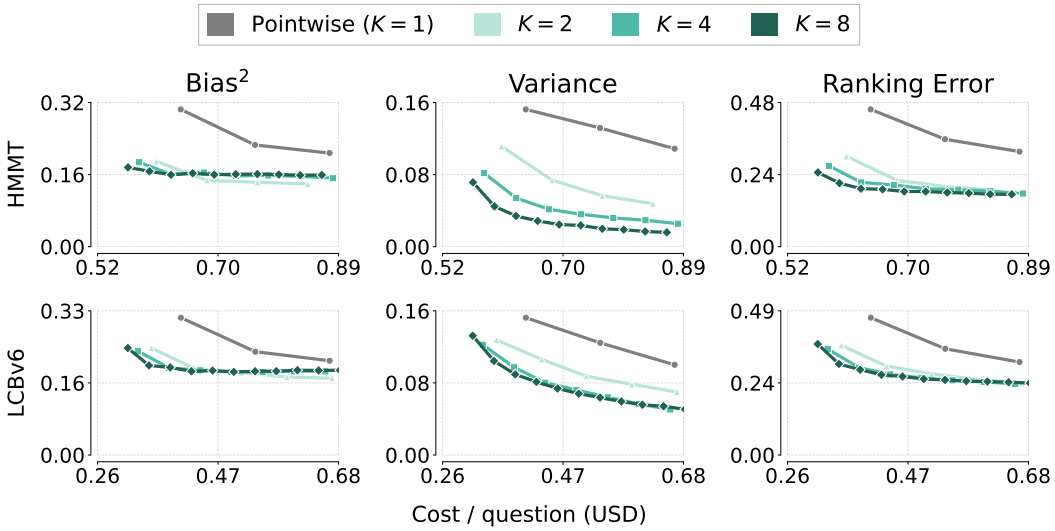

Figure 15: **Larger groups reduce variance faster but can increase bias.** Results for Qwen3.5-4B on HMMT and LCBv6. Increasing group size allows more evaluations within the same cost budget, reducing variance faster. However, these gains can come at the expense of bias: on LCBv6, $K = 8$ yields higher bias than smaller groups. Compute is measured as the total dollar cost of all input and output tokens across LLM calls; see App. D.2 for further details.

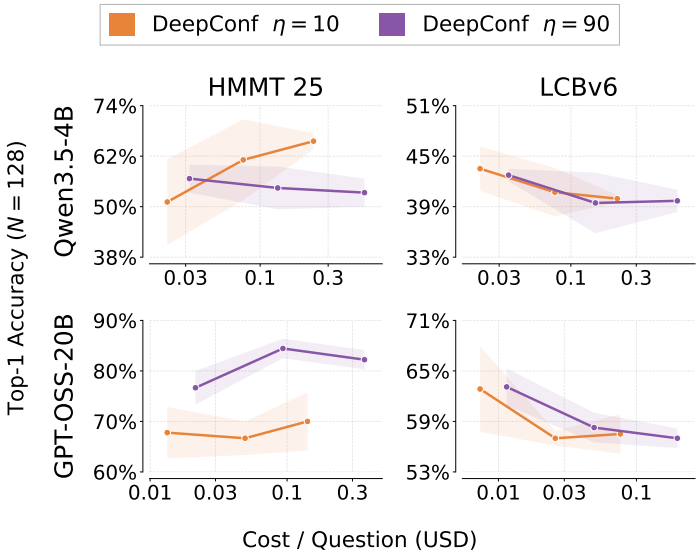

Figure 16: **DeepConf's (Fu et al., 2025) pruning threshold depends on the model.** Top-1 Accuracy versus total inference cost for DeepConf with aggressive ($\eta = 10$) and conservative ($\eta = 90$) pruning. Aggressive pruning generally works better for Qwen3.5-4B, while conservative pruning works better for GPT-OSS-20B. Compute is measured as the total dollar cost of all input and output tokens across LLM calls; see App. D.2 for further details.

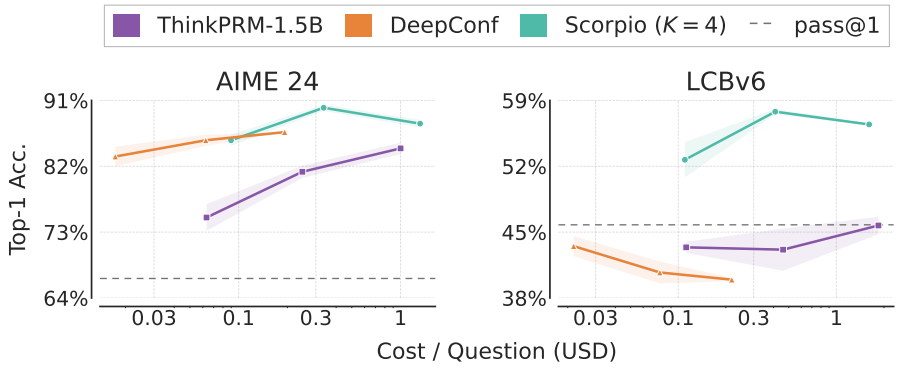

Figure 17: **Test-time search using SCORPIO matches or outperforms prior methods.** Results for Qwen3.5-4B on AIME 2024 and LCBv6. Top-1 accuracy of test-time search versus total inference cost on AIME 2024 and LCBv6 for Qwen3.5-4B. We vary the number of solutions we start with: $N \in \{8, 32, 128\}$ for SCORPIO and ThinkPRM, and $N \in \{16, 64, 256\}$ for DeepConf to keep total costs comparable. Inference compute is measured as the total dollar cost of all input and output tokens across LLM calls; see App. D.2 for details.

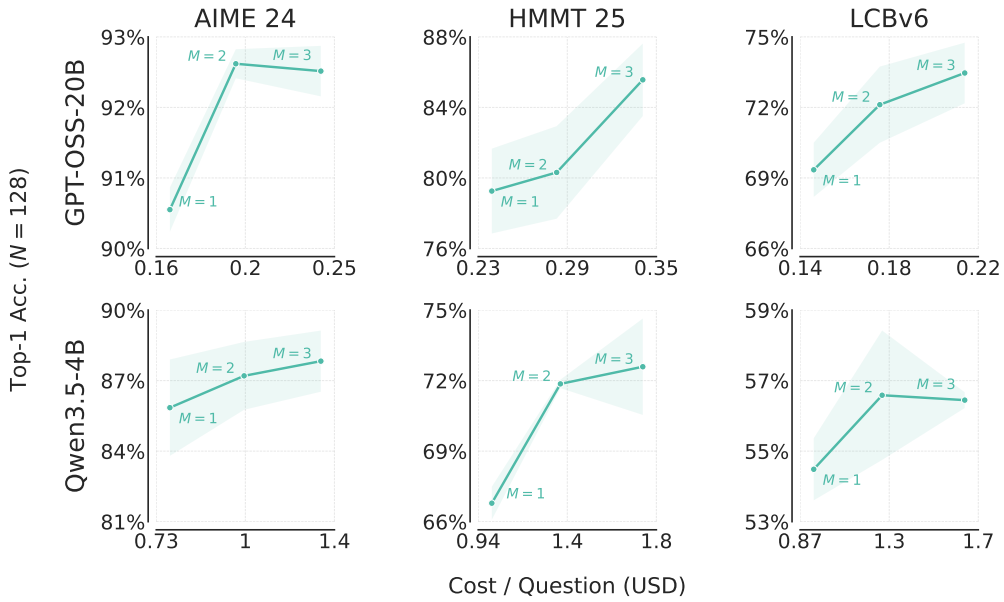

Figure 18: **Repeated evaluation of partial solutions improves search accuracy.** Top-1 accuracy of test-time search using SCORPIO versus total inference cost on AIME 2024, HMMT Feb 2025, and LCBv6. We fix the number of solutions we start with at $N = 128$ and vary the number of times the random grouping and scoring process is repeated, $M \in \{1, 2, 3\}$. Repeated evaluations improve accuracy over $M = 1$ across all three datasets. Inference compute is measured as the total dollar cost of all input and output tokens across LLM calls; see App. D.2 for details.

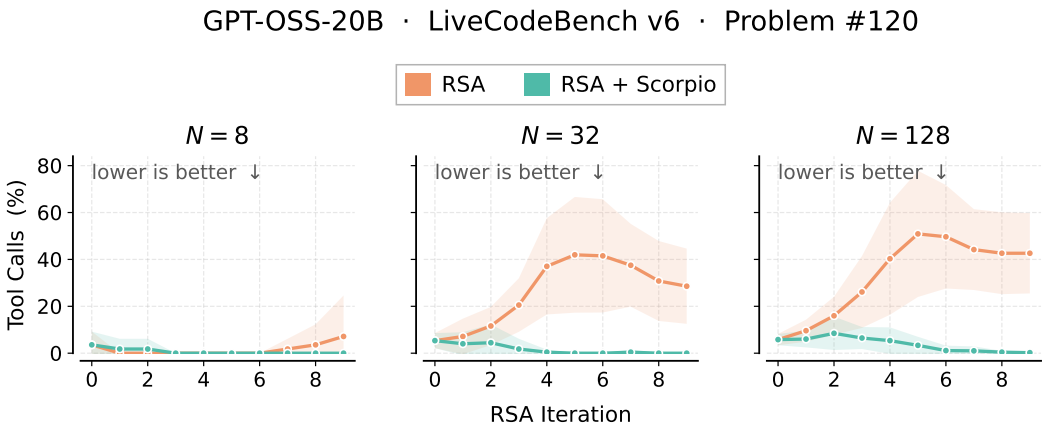

Figure 19: **Larger pools amplify tool-call responses in RSA, while SCORPIO limits their amplification.** Proportion of responses containing tool calls across aggregation rounds for a single LCBv6 problem using GPT-OSS-20B. This is a single-turn task that requires a solution in a Python code block. Some responses attempt tool calls to test their logic, which we identify using the pattern `to=[A-Za-z_]`. The harness does not support tool calls, and 95% of these responses receive zero reward. Tool-call responses remain uncommon with $N = 8$ but become increasingly common with $N = 32$ and $N = 128$ as aggregation rounds progress. In RSA + SCORPIO, candidates are scored using SCORPIO after each round, and only the top half of candidates are used to build aggregation sets for the next round. This selection reduces the proportion of tool-call responses to nearly zero across all three pool sizes.

F SCORING CANDIDATES USING LLMS

F.1 PROMPT USED FOR SCORPIO SCORING OF COMPLETE CANDIDATES

Our scoring prompt is largely based on the pairwise scoring prompt of Singh et al. (2026), with the change that we provide K solutions instead of two and ask the model to output a score for each solution.

Prompt for coding tasks (LiveCodeBench)

```
You are an expert code reviewer.  Compare K solutions to a programming
problem and rate their correctness.

**Evaluation Guidelines:**
- Analyze the problem's requirements and constraints.
- Mentally trace each solution with test cases (including edge cases) to
verify correctness.

- If multiple solutions appear correct, prefer the more robust and
fault-tolerant ones.


**Problem**
{problem}

**Solution 1**
{solution 1}
...
**Solution K**
{solution K}

**Output Format:** First, compare and contrast the solutions provided in
context.  Then provide your step-by-step reasoning following the evaluation
guidelines above.  Then, on separate new lines, provide your final ratings
using the EXACT tags below.
<rating_1>INTEGER_1_TO_10</rating_1>
...
<rating_K>INTEGER_1_TO_10</rating_K>

**Rating Rules:**
- Rate correctness on a 1-10 scale (10 = correct & robust,
5 = borderline, 1 = incorrect).
- Higher rating wins.  Equal ratings imply ties.  Please provide your
analysis now.
```

Prompt for math (HMMT 25)

```
You are an expert math contest grader.  Compare K submissions and rate
correctness based solely on the final answer.

**Evaluation Guidelines:**
- Extract each submission's final answer.  Use any provided reasoning only to
help you assess whether the stated final answer is trustworthy.  Do not award
credit for method quality or rigor.

- Carefully analyze the problem statement and the submissions to assess
whether each final answer is correct.  Grade only the final answer.


**Problem**
{problem}

**Solution 1**
{solution 1}
...
**Solution K**
{solution K}

**Output Format:** First, compare and contrast the solutions provided in
context.  Then provide your reasoning (what checks you performed) following
the evaluation guidelines above.  Then, on separate new lines, give ratings
using the EXACT tags below.
<rating_1>INTEGER_1_TO_10</rating_1>
...
<rating_K>INTEGER_1_TO_10</rating_K>

**Rating Rules:**
- Rate correctness on a 1-10 scale (10 = certainly correct, 8 = very likely
correct, 5 = uncertain/borderline, 3 = likely incorrect,

1 = certainly incorrect).
- Higher rating wins.  Equal ratings imply ties.  Please provide your
analysis now.
```

For reasoning-gym games we use the math prompt above verbatim, replacing `expert math contest grader` with `expert grader for a reasoning-puzzle benchmark`.

F.2 Prompt used for Scorpio scoring of partial candidates

For the tree-search experiments in Sec. 5.2, Scorpio scores *in-progress* trajectories rather than completed solutions. The prompt therefore asks for the probability that each candidate will reach a correct final answer within the remaining token budget if generation continues. A single template is used for both the math (HMMT Feb 2025) and coding (LCBv6) benchmarks. The judge is shown each candidate's reasoning so far, the number of tokens consumed, and the remaining token budget.

Prompt for scoring partial candidates (math and code)

```
You are evaluating K in-progress attempts at the SAME problem, labelled
Candidate 1 to Candidate K. For EACH candidate, estimate the probability that
it reaches a correct final answer when continued.  Do not solve the problem
yourself.

## Generation budget
- Each attempt has used {tokens used} tokens so far
- Each attempt can use at most {remaining budget} more tokens to continue

## Problem

{problem}

## Candidates ### Candidate 1

{partial solution 1}
...  ### Candidate K

{partial solution K}

## How to judge Do this in two stages, in a single response.

Stage 1 - compare and contrast.  Looking at all K attempts together, compare
and contrast the approaches they take.  For each candidate, work through
its reasoning and CHECK it - verify the key claims and steps (calculations,
logic, edge cases - whatever the task involves) - and identify what is
correct and what is incorrect or questionable about it.  Where the candidates
agree or diverge is itself evidence about what is right.

Stage 2 - score each candidate.  Using that analysis, assign each candidate
a score = the probability that it reaches a correct final answer when
continued.  For each candidate, weigh:
- Correctness so far - are the written steps right, and are any mistakes
minor (recoverable) or fatal (cannot reach the answer without changing
approach)?

- Approach - is it a valid method that, if continued correctly, can reach
the answer?

- Budget feasibility - with {remaining budget} tokens left, can it
realistically finish?
Incompleteness by itself is not a penalty.

## Score meaning (per candidate) Score = probability that THAT candidate
reaches a correct final answer when continued, on a 1-10 scale:

10 = almost certain 7 = likely 5 = about 50%
3 = unlikely 1 = almost no chance

## Output format First write your Stage 1 compare-and-contrast analysis
across all candidates.  Then go through the candidates IN ORDER - for each,
reason about its state and then give its score:

Candidate 1:  <reasoning about this candidate's state>
Score:  N
Candidate 2:  <reasoning about this candidate's state>
Score:  N
...  continue through Candidate K

Then end with EXACTLY one final line repeating all K scores in order,
comma-separated:

«<SCORES: N1, N2, ..., NK»>

Score every candidate (an integer from 1 to 10).
```