POLICY OPTIMIZATION WITH MULTIPLE VERIFIERS VIA PARETO RANKINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) has become an important approach to improving the reasoning capabilities of large language models. In tasks such as code generation and instruction following, multiple verifiers evaluate whether a response satisfies different requirements. Their outcomes provide detailed feedback about partial success, yet this feedback is commonly aggregated into a scalar reward for policy optimization. Rewarding only complete success discards distinctions among partial solutions, while counts and weighted sums impose comparisons between responses that satisfy different subsets of requirements. We introduce Partial-Order Likelihood Policy Optimization (POLPO), which learns from the comparisons established by verification without first constructing an aggregate reward. Pareto dominance identifies responses that satisfy additional requirements without losing any already satisfied, while leaving responses with conflicting outcomes incomparable. POLPO extends the listwise likelihood underlying Direct Preference Optimization by maximizing the total probability of all complete rankings consistent with these partial orders. This objective provides supervision from verified improvements even before complete solutions are available, without requiring a prescribed ordering of incomparable responses. Experiments on code generation and instruction following demonstrate improved learning efficiency and higher rates of complete requirement satisfaction compared with scalar-reward training. These results support partial-order likelihood as an effective approach to learning from multiple verifiers.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has become an important approach to improving the reasoning capabilities of large language models (LLMs) (Lambert et al., 2025; Guo et al., 2025). By evaluating model-generated responses automatically, RLVR provides training supervision without requiring annotations for every intermediate reasoning step. In many applications, however, verification involves multiple requirements rather than a single correctness check. A generated program must pass tests covering different inputs and behaviors, while an instruction-following response may need to satisfy several content and formatting constraints (Hendrycks et al., 2021a; Pyatkin et al., 2025). The resulting verification outcomes identify both whether a response succeeds completely and which requirements it satisfies along the way. How to use this information during training is therefore a central question for RLVR with multiple verifiers.

A common approach is to aggregate verification outcomes into a scalar reward and optimize the policy using a policy-gradient method, such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024). This aggregation determines which distinctions among responses are available to the learning algorithm. An all-pass reward directly expresses the goal of satisfying every requirement, but assigns the same reward to all partially successful responses. A pass-rate reward provides more graded feedback, but ranks responses by how many checks they pass, regardless of which checks they satisfy. Weighted sums allow different priorities, at the cost of specifying how gains on one requirement compensate for losses on another. Although these rewards can share complete success as their optimum, they induce different comparisons among the partial solutions encountered during training. The challenge is to learn from partial progress without having to prescribe trade-offs between requirements.

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.

Partial orders offer a way to express this distinction. Under Pareto dominance, a response improves on another if it passes every check passed by the other and at least one additional check. Consider three programs y_1, y_2, y_3 with outcomes on three tests given by

$$\mathbf{c}_1 = (1, 1, 0), \qquad \mathbf{c}_2 = (0, 0, 1), \qquad \mathbf{c}_3 = (1, 0, 0), \tag{1}$$

where 1 denotes a passed test. Program y_1 improves on y_3, whereas y_2 passes a test that neither of the others passes. Verification therefore establishes

$$y_1 \succ_{\mathrm{P}} y_3, \qquad y_1 \parallel_{\mathrm{P}} y_2, \qquad y_2 \parallel_{\mathrm{P}} y_3, \tag{2}$$

where $\parallel_{\mathrm{P}}$ denotes incomparability. An all-pass reward misses the improvement from y_3 to y_1, while a pass-rate reward additionally favors y_1 over y_2. Treating both incomparabilities as scalar reward ties would instead force y_1 and y_3 to tie, erasing the verified improvement. A partial order preserves that improvement while leaving the trade-off involving y_2 unresolved.

To turn these partial orders into a training objective, we introduce *Partial-Order Likelihood Policy Optimization (POLPO)*, illustrated in Figure 1. We use the implicit-reward parameterization of Direct Preference Optimization (DPO), originally proposed for aligning language models with human preferences (Rafailov et al., 2023). Policy-to-reference log probability ratios supply scores for a Plackett–Luce model, which assigns probabilities to rankings of the responses. POLPO maximizes the probability that all observed Pareto comparisons hold, obtained by summing the probabilities of the complete rankings consistent with them. For the partial order in Eq. (2), every position for y_2 is allowed as long as y_1 precedes y_3. The objective thus captures the comparisons established by verification while marginalizing over the ordering decisions that verification leaves unspecified. Our contribution is to bring the established partial-order PL likelihood (Liu et al., 2019) into online policy learning from multiple verifiers, using Pareto comparisons as supervision and DPO-style scores to connect the likelihood to the response policy.

This formulation makes the partial order itself the supervision for policy optimization. Its likelihood models the specified comparisons jointly, and differentiating its logarithm yields a weighted sum of response log-probability gradients. Consequently, a group containing a Pareto improvement can provide a learning signal even when no response satisfies every requirement. Although POLPO uses scalar scores internally to parameterize ranking probabilities, it does not require a scalar reward that orders every pair of responses or represents incomparability through equal values. The distinction is between assigning scores within a probabilistic model and prescribing a complete ordering as the training target.

We evaluate whether this use of verification feedback improves learning in controlled program synthesis and practical code-generation tasks. The controlled task isolates the difficulty of learning from partial solutions that satisfy different subsets of requirements: POLPO reaches complete success with fewer sampled programs than training with either all-pass or pass-rate rewards. On MBPP and APPS, POLPO improves both complete program correctness and average test satisfaction under matched rollout budgets, compared with scalar-reward and comparison-based baselines. Training on AceCode further demonstrates transfer to external coding benchmarks. We also examine instruction following, where verifiers check heterogeneous requirements, to study how the approach generalizes beyond execution-based feedback. Together, these experiments investigate partial-order supervision as a way to use multiple verifiers directly in language-model training.

2 Reinforcement Learning with Multiple Verifiers

In tasks with multiple requirements, a response can satisfy some requirements while violating others. We formalize this verification feedback and describe the scalar rewards commonly used to train policies from it.

2.1 Task Setup

Let $x \sim \rho$ denote a prompt and $\pi_\theta(y \mid x)$ a language-model policy. Each prompt has $K = K(x) \geq 1$ binary verifiers, with $c_k(x, y) = 1$ when response y satisfies requirement k and 0 otherwise. Their outputs form a verification vector $\mathbf{c}(x, y) = (c_1(x, y), \ldots, c_K(x, y)) \in \{0, 1\}^K$. In code generation, each verifier can check a program on a test input; in instruction following, it can check a content or formatting constraint (Hendrycks et al., 2021a; Pyatkin et al., 2025).

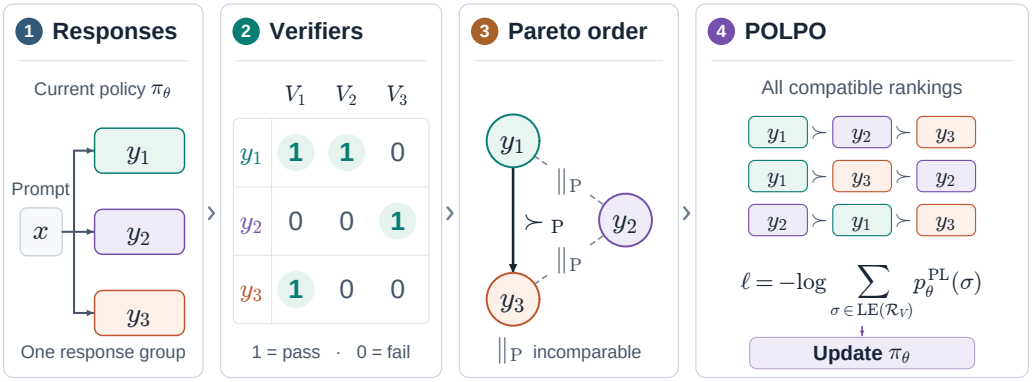


Figure 1: Overview of POLPO. Responses sampled from the current rollout policy are checked against multiple requirements. In the Pareto-order panel, the solid arrow indicates verified dominance, while dashed connections indicate incomparability. For the verification outcomes in Eq. (1), only $y_1 \succ y_3$ is established, so three complete rankings are compatible. POLPO minimizes the negative log of their total probability under the PL model parameterized by the policy scores in Eq. (6); the prompt and response group are omitted from the displayed objective for brevity.

The task goal is to increase the probability of satisfying all requirements. The individual verifier outcomes also describe partial success: they identify which requirements are already satisfied when a response is not yet fully correct. For a group $Y = (y_1, \ldots, y_N)$ of responses to the same prompt, we write $\mathbf{c}_i = \mathbf{c}(x, y_i)$. The learning question is how to use these vectors to guide the policy toward complete solutions.

2.2 Scalar-Reward Aggregation

A common approach maps the verification vector to a scalar reward $r(x, y) = g(\mathbf{c}(x, y))$ and optimizes its expectation:

$$\mathcal{J}_{\mathrm{reward}}(\theta) = \mathbb{E}_{x \sim \rho,\, y \sim \pi_\theta(\cdot|x)}[r(x, y)]. \tag{3}$$

For example, an all-pass reward is one only when every check passes, whereas a pass-rate reward averages the individual outcomes:

$$r_{\mathrm{all}}(x, y) = \prod_{k=1}^{K} c_k(x, y), \qquad r_{\mathrm{rate}}(x, y) = \frac{1}{K} \sum_{k=1}^{K} c_k(x, y). \tag{4}$$

An all-pass reward directly expresses complete success, but gives every partially successful response the same reward. A pass-rate reward distinguishes partial solutions by the number of checks passed, while assigning equal rewards to responses that pass different subsets of the same size. A weighted sum, $r_{\mathbf{w}}(x, y) = \sum_{k=1}^{K} w_k c_k(x, y)$ with $w_k > 0$, permits different priorities but specifies how gains on one check compensate for losses on another. These choices can favor the same fully successful responses while assigning different rewards to the partial solutions encountered during training.

The aggregation determines the verification information supplied to the policy optimizer. GRPO is one representative policy-gradient method that uses this scalar feedback (Shao et al., 2024). It samples several responses per prompt and normalizes their rewards within the group to obtain advantages for a policy update. If every response receives the same reward, as with all-pass rewards when no response succeeds completely, the reward-derived advantages vanish. See Appendix A.2 for the advantage definition and policy surrogate.

These limitations motivate using the improvements established by verification directly as comparison targets, while leaving tradeoffs between conflicting outcomes unspecified.

3 PARTIAL-ORDER LIKELIHOOD POLICY OPTIMIZATION

3.1 PARETO COMPARISONS FROM MULTIPLE VERIFIERS

To use verification feedback without prescribing trade-offs between requirements, we compare responses through Pareto dominance. A response dominates another if it passes every check passed by the other and at least one additional check. Writing $\mathbf{c}_i = (c_{i1}, \ldots, c_{iK})$, we define

$$y_i \succ_{\mathrm{P}} y_j \quad \Longleftrightarrow \quad \big[\forall k,\ c_{ik} \geq c_{jk}\big] \ \wedge\ \big[\exists k,\ c_{ik} > c_{jk}\big]. \tag{5}$$

When each response passes a check that the other fails, neither dominates the other. This leaves the tradeoff between their satisfied requirements unspecified. Responses with identical verification vectors are also left unordered. For the verification outcomes in Eq. (1), verification establishes only $y_1 \succ_{\mathrm{P}} y_3$; both pairs involving y_2 remain incomparable, as shown in Eq. (2).

The resulting relation on response indices, $\mathcal{R}_V = \{(i, j) : y_i \succ_{\mathrm{P}} y_j\}$, is a strict partial order: it is irreflexive and transitive. With a single binary verifier, every correct response precedes every incorrect response, with no comparisons within either group. More generally, we write $\mathcal{R}_V = \mathcal{A}(\mathbf{c}_1, \ldots, \mathbf{c}_N)$ for a task-specific comparison rule, requiring that its output be a strict partial order; Pareto dominance is the rule used for multi-verifier feedback here. An unordered pair supplies no comparison target and does not require equal model scores. The remaining question is how to turn this partial-order supervision into a policy-training objective.

3.2 LEARNING FROM THE PARTIAL-ORDER LIKELIHOOD

To turn these Pareto comparisons into a policy-training objective, we build on the formulation of Direct Preference Optimization (DPO), which was originally proposed for aligning language models with human preferences (Rafailov et al., 2023). DPO uses an implicit reward derived from the policy to model comparison probabilities. We use this parameterization to model the probability that all comparisons in $\mathcal{R}_V$ hold jointly. Given a fixed reference policy π_{ref}, we define the response score as

$$s_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)}. \tag{6}$$

Here $\beta > 0$ sets the score scale, and both log probabilities sum over response tokens (see Appendix A.1). The implicit-reward interpretation follows from KL-regularized reward maximization: holding θ fixed, s_θ is a reward for which π_θ is the optimal policy,

$$\pi_\theta(\cdot \mid x) = \arg\max_q \Big\{ \mathbb{E}_{y \sim q}[s_\theta(x, y)] - \beta D_{\mathrm{KL}}(q \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x)) \Big\}, \tag{7}$$

where q ranges over distributions on complete responses. Adding a prompt-dependent constant $C(x)$ to the reward preserves this optimizer and cancels from ranking probabilities. See Appendix A.3 for a derivation of this correspondence. POLPO learns the implicit reward through the comparison likelihood below: the verification vectors determine which comparisons are observed, while the policy scores determine their probabilities.

We use the Plackett–Luce (PL) model (Luce, 1959; Plackett, 1975) to define a distribution over rankings from these scores. Write $s_i = s_\theta(x, y_i)$ and let $\sigma = (\sigma_1, \ldots, \sigma_N)$ list responses from best to worst. The probability of this ranking is

$$p_\theta^{\mathrm{PL}}(\sigma \mid x, Y) = \prod_{k=1}^{N-1} \frac{\exp(s_{\sigma_k})}{\sum_{j=k}^{N} \exp(s_{\sigma_j})}. \tag{8}$$

At each position, a response is selected with probability proportional to its exponentiated score among the remaining candidates; the final choice has probability one. Higher implicit rewards therefore make a response more likely to appear earlier in the ranking.

To see how this ranking model can learn from partial comparisons, return to the example in Eq. (2). Verification establishes only $y_1 \succ y_3$, leaving the position of y_2 unspecified. There are exactly three complete rankings consistent with this feedback:

$$y_1 \succ y_2 \succ y_3, \qquad y_1 \succ y_3 \succ y_2, \qquad y_2 \succ y_1 \succ y_3.$$

Each ranking satisfies the verified comparison, so learning should increase their total probability without selecting a particular position for y_2. Since the three rankings are mutually exclusive, their probabilities add up to the probability that y_1 precedes y_3:

$$\begin{aligned} p_\theta(\{(1,3)\} \mid x, Y) &= p_\theta^{\mathrm{PL}}((1,2,3) \mid x, Y) + p_\theta^{\mathrm{PL}}((1,3,2) \mid x, Y) + p_\theta^{\mathrm{PL}}((2,1,3) \mid x, Y) \\ &= \frac{e^{s_1}}{e^{s_1} + e^{s_3}}. \end{aligned} \tag{9}$$

The score of y_2 cancels because it participates in no verified comparison. The learning signal therefore depends only on the comparison between y_1 and y_3. Summing over the allowed positions of y_2 preserves the unspecified part of the order, without requiring equal scores for incomparable responses.

For a general partial order R_V, we adopt the established Plackett–Luce likelihood obtained by summing over its linear extensions (Liu et al., 2019). These are the complete rankings that satisfy every comparison in R_V, and we denote their set by $\mathrm{LE}(R_V)$. A ranking belongs to this set exactly when y_i precedes y_j for every $(i, j) \in R_V$. POLPO uses this likelihood to train the response policy: verification determines R_V, the policy-to-reference scores in Eq. (6) parameterize the ranking probabilities, and training minimizes the negative log-likelihood:

$$\ell_{\mathrm{POLPO}}(\theta; x, Y, \mathcal{R}_V) = -\log p_\theta(\mathcal{R}_V \mid x, Y) = -\log \sum_{\sigma \in \mathrm{LE}(\mathcal{R}_V)} p_\theta^{\mathrm{PL}}(\sigma \mid x, Y). \tag{10}$$

This sum marginalizes over the ordering decisions left unspecified by verification. The scalar scores parameterize the ranking probabilities; the partial order supplies the comparison targets. A complete ranking leaves a single PL term, while an empty comparison relation gives likelihood one and zero loss. The objective recovers DPO for a single comparison (see Appendix A.4), but generally differs from a sum of pairwise DPO losses when multiple comparisons are observed (see Appendix B.3).

We optimize Eq. (10) with gradient-based updates to the policy parameters, differentiating through the scores in Eq. (6). The likelihood is evaluated exactly using a dynamic program over subsets of responses, avoiding explicit enumeration of complete rankings. See Appendices B and C for the gradient derivations and likelihood computation, and see Algorithm 1 for the training procedure.

4 Experiments

We study whether comparisons from multiple verifiers help a policy learn to satisfy every requirement. A controlled program-synthesis task isolates learning from partial solutions, coding tests this idea with a language model, and multi-instruction following extends it to heterogeneous requirements. The main comparison is between POLPO and GRPO with an all-pass reward (GRPO–All) or the fraction of passed checks (GRPO–Rate). These scalar rewards are defined in Eq. (4). Complete success is the primary outcome; average check satisfaction is a complementary measure of partial progress. See Appendix D for the task-specific training and evaluation settings.

4.1 Controlled Program Synthesis

We first isolate learning from multiple verifier outcomes using the controlled FizzBuzz synthesis task in Figure 2. For divisors (a, b), the target returns `Fizz`, `Buzz`, or `FizzBuzz` according to divisibility and otherwise returns the input itself; standard FizzBuzz uses $(a, b) = (3, 5)$. A categorical policy constructs a fixed three-branch program by choosing three divisor–output pairs (d_j, v_j), with $d_j \in \{2, \ldots, M\}$ and $v_j \in \{\texttt{Fizz}, \texttt{Buzz}, \texttt{FizzBuzz}\}$. Each input $n \in \{1, \ldots, 60\}$ defines a binary verifier, giving every sampled program a 60-dimensional verification vector. Because the policy is small, we can compute its exact probability of generating a program that passes all verifiers.

Model output: six categorical choices

$$(d_1, v_1),\ (d_2, v_2),\ (d_3, v_3), \qquad d_j \in \{2, \ldots, M\}, \quad v_j \in \{\texttt{Fizz}, \texttt{Buzz}, \texttt{FizzBuzz}\}.$$

Synthesized program

```
program(n):
  if n % d1 == 0: return v1
  if n % d2 == 0: return v2
  if n % d3 == 0: return v3
  return str(n)
```

Verifiers

```
for n = 1, ..., 60:
  expected = target_{a,b}(n)
  if program(n) == expected:
    c_n ← 1   # pass
  else: c_n ← 0   # fail
```

$\mathbf{c}(\texttt{program}) = (c_1, \ldots, c_{60}) \in \{0, 1\}^{60}$.

Figure 2: Controlled program-synthesis task. The trainable policy fills only the highlighted branch slots; the program structure, fallback, target outputs, and 60 input-specific verifiers are fixed. For $(a, b) = (3, 5)$, the two correct assignments are $(15, \texttt{FizzBuzz})$ followed by $(3, \texttt{Fizz})$ and $(5, \texttt{Buzz})$ in either order.

GRPO–All receives no signal from incomplete programs, whereas GRPO–Rate orders them only by the number of passed inputs. POLPO instead preserves componentwise dominance between their verification vectors. We test whether this distinction reduces the number of sampled programs needed to reach a 0.9 probability of complete success, varying both the divisor search space M and the target divisors (a, b). See Appendix D.1 for the complete protocol.

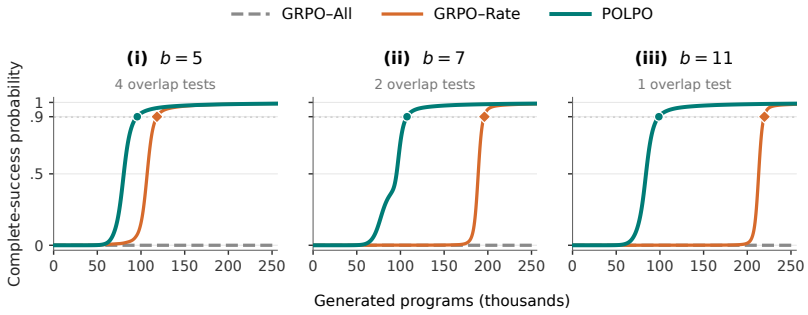


Figure 3: Complete-success probability for target pairs $(3, 5)$, $(3, 7)$, and $(3, 11)$ at $M = 60$ in the seed-42 runs. All settings retain the same 60 input verifiers and initial policy. The horizontal dotted line marks 0.9, and markers indicate the first recorded crossings.

Results. Across all three target pairs in Figure 3, POLPO reaches 0.9 complete-success probability with fewer sampled programs than either scalar-reward baseline. Relative to GRPO–Rate, the reduction in required programs grows from 19.0% for $(3, 5)$ to 45.2% for $(3, 7)$ and 55.0% for $(3, 11)$; GRPO–All does not reach the threshold within 512,000 programs. At 153,600 programs, GRPO–Rate already has 98.04% mean verifier satisfaction but only a 0.10% probability of passing the overlap input $n = 33$, compared with 99.28% for POLPO. Thus, a high scalar pass rate can conceal a persistently unsatisfied part of the specification; see Appendix E.1 for threshold counts and diagnostic curves.

4.2 Coding Tasks

Code generation provides a direct setting for learning from multiple verifiers, with each test checking a particular program behavior. We study MBPP, APPS Introductory, and transfer from AceCode to external benchmarks. All methods use the same tests and eight fresh responses per prompt, with one optimizer update per rollout batch. We evaluate final checkpoints from one training seed using greedy decoding. For MBPP, *All-pass* measures the percentage of problems whose response passes

Table 1: MBPP test results (%) after five training epochs. Both models use thinking during training and evaluation, with one greedy response per problem. All-pass measures success on every test (pass@1); Avg. pass averages the fraction of tests passed equally across problems.

	Qwen3-0.6B		Qwen3-4B	
Method	All-pass	Avg. pass	All-pass	Avg. pass
Initial model	30.00	34.40	62.40	66.40
GRPO–All	40.40	45.60	66.80	71.53
GRPO–Rate	42.00	48.53	65.80	70.40
REINFORCE++–All	39.00	46.67	67.20	72.07
REINFORCE++–Rate	37.20	43.20	66.40	70.67
GDPO	37.80	43.53	65.40	70.40
GraphDPO	42.80	48.27	67.60	72.80
DPO–Pareto	42.00	47.13	66.80	72.27
POLPO	**45.40**	**52.13**	**71.20**	**77.73**

Table 2: APPS Introductory test results (%) after two training epochs.

Method	All-pass	Avg. pass
Initial model	20.50	35.77
GRPO–All	23.60	39.05
GRPO–Rate	23.30	38.95
GDPO	23.40	39.13
GraphDPO	21.80	37.38
POLPO	**25.90**	**42.67**

Table 3: External pass@1 (%) after one epoch of AceCode training. + denotes augmented tests.

	HumanEval (164)		MBPP (378)	
Method	Original	+	Original	+
Initial model	70.12	65.24	68.52	59.52
GRPO–All	**78.05**	70.73	73.02	62.96
GRPO–Rate	76.83	70.73	73.02	62.43
POLPO	**78.05**	**72.56**	**74.87**	**64.29**

every test (pass@1), while *Avg. pass* averages the fraction of tests passed within each problem, then equally across problems. See Appendix D.2 for the full protocols.

Baselines. GRPO–All and GRPO–Rate provide scalar-reward baselines. On MBPP and APPS, we additionally include REINFORCE++–All and REINFORCE++–Rate (Hu et al., 2025), which use the same rewards with batch-wide normalization to test scalar-reward learning beyond GRPO's group-relative update. GDPO (Liu et al., 2026c) instead normalizes each test's outcomes separately before aggregation, providing a baseline that explicitly uses individual verifier signals. DPO–Pareto applies the DPO objective (Rafailov et al., 2023) to all strict Pareto dominance pairs, while GraphDPO (Liu et al., 2026b) learns local rankings over each response and those it dominates. These two baselines use the same Pareto relation as POLPO, allowing us to compare pairwise and local graph objectives with the likelihood of the entire partial order.

MBPP and APPS Introductory. MBPP contains short Python programming problems with three tests per problem (Austin et al., 2021). We train Qwen3-0.6B and Qwen3-4B with thinking enabled, using the official training and test splits. POLPO achieves the highest All-pass and Avg. pass at both model sizes (Table 1), improving complete program correctness as well as average test satisfaction.

APPS provides competition-style programming problems with a variable number of tests (Hendrycks et al., 2021a). We train and evaluate Qwen2.5-Coder-1.5B-Instruct on its Introductory tier. POLPO leads both metrics across the compared methods (Table 2), extending the improvement to a setting where the number and coverage of verifiers vary across problems.

AceCode and external evaluation. To assess transfer beyond the training problems, we train the same Qwen2.5-Coder model on AceCode's hard subset (Zeng et al., 2025) and evaluate on HumanEval and the MBPP subset distributed with EvalPlus (Liu et al., 2023). We report pass@1 on both the original and augmented test suites; this MBPP subset differs from the official test split used above. POLPO matches GRPO–All on original HumanEval and outperforms both GRPO variants on the other three evaluations (Table 3).

Table 4: Complete constraint satisfaction (%) on IFM5 and MulDimIF, with separate training on each dataset. K counts individual constraints; L1–L4 are MulDimIF difficulty levels. Bold marks column maxima at the reported precision

	IFM5						MulDimIF				
Method	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	Overall	L1	L2	L3	L4	Overall
Initial model	56.1	29.9	18.5	7.4	3.7	27.1	85.3	63.7	48.0	32.3	57.3
GRPO–Rate	90.8	66.8	49.9	32.5	20.0	58.4	94.7	89.7	85.7	83.7	88.4
GDPO	88.0	65.6	48.3	29.4	23.7	56.7	94.7	87.7	83.7	82.0	87.0
GRPO–All	90.1	65.6	43.8	26.6	14.8	54.9	95.3	90.0	82.7	81.7	87.4
POLPO	**91.0**	**68.2**	**55.0**	**35.5**	**27.4**	**61.2**	**96.7**	**91.3**	**88.0**	**85.3**	**90.3**

4.3 Multi-Instruction Following

We evaluate learning from multiple verifiable constraints on IFM5, the IF-RLVR data released by Pyatkin et al. (2025), MulDimIF (Ye et al., 2026), and MathIF (Fu et al., 2026). IFM5 and MulDimIF use separate models trained on their respective training splits and evaluated on the corresponding test splits. MathIF additionally evaluates mathematical correctness alongside instruction adherence, using models trained on a separate constraint-augmented MATH dataset. For IFM5 and MulDimIF, the initial model is Qwen3-4B-Instruct-2507. On both datasets, we compare the initial model, GRPO–All, GRPO–Rate, GDPO (Liu et al., 2026c), and POLPO. POLPO constructs the dominance relation in Eq. (5) from individual verification outcomes, with score scale $\beta = 10^{-3}$. Both GRPO–All and GRPO–Rate use a KL coefficient of 10^{-3}, which is distinct from POLPO's score scale. The primary metric is the percentage of evaluated prompts whose responses satisfy every constraint. Appendix D.3 provides the data splits, evaluation protocols, and training configurations.

IFM5 and MulDimIF. IFM5 groups prompts by their number of constraints, $K \in \{1, \ldots, 5\}$, whereas MulDimIF defines four difficulty levels based on both the number and variety of constraints (Ye et al., 2026). Table 4 shows that POLPO outperforms all three training baselines on IFM5, both overall and for every K. On MulDimIF, it also outperforms all three training baselines overall and at every difficulty level.

MathIF. We further evaluate whether responses can satisfy instruction constraints while solving mathematical problems on MathIF (Fu et al., 2026). We train Qwen3-0.6B on a separate constraint-augmented MATH training set and evaluate on 420 MathIF examples, using greedy decoding with thinking disabled for all compared models. HAcc measures complete constraint satisfaction, SAcc the mean fraction of constraints satisfied, and Acc mathematical correctness. As shown in Table 5, POLPO achieves the highest Joint score, the primary metric on MathIF, which requires both mathematical correctness and satisfaction of every instruction constraint. It also achieves the strongest instruction adherence, while GRPO–All achieves the highest mathematical accuracy. Appendices D.3.5 and E.2 provide more detailed results.

Table 5: MathIF results (%).

Method	HAcc	SAcc	Acc	Joint
Initial model	31.0	55.6	13.8	5.7
GRPO–All	52.6	70.4	**29.5**	19.8
GRPO–Rate	71.2	83.1	27.1	19.3
POLPO	**76.7**	**86.5**	26.9	**21.4**

5 Related Work

Multi-verifier RLVR and reward design. Multiple verification outcomes provide training supervision in RLVR tasks, such as instruction following and code generation (Pyatkin et al., 2025; Zeng et al., 2025). A central design choice is how to turn partial success into a policy-training signal. Pass-rate rewards provide denser feedback than all-pass rewards, but do not consistently improve complete-solution accuracy in code generation (Li et al., 2026b). GDPO normalizes reward components separately before aggregation (Liu et al., 2026c), while VeRPO constructs execution-based rewards that account for partial test success, test difficulty, and test density (Wang et al., 2026). In multi-instruction following, Scarcity-Aware Rewards counter exploration biased toward easier instructions by assigning rewards according to how rarely each instruction is satisfied (Zhang et al.,

2026b). HiR takes a complementary approach: it reuses partially successful responses by rewriting their prompts to retain the instructions they satisfy (Zhang et al., 2026a). These studies highlight the importance of how partial success is represented during learning. POLPO uses verification outcomes to identify supported comparisons between responses and learns their joint likelihood.

Vector-valued feedback also supports learning across competing objectives. Vector Policy Optimization trains diverse sets of responses that specialize in different reward trade-offs, improving their usefulness for test-time search (Bahlous-Boldi et al., 2026). ParetoPO targets the trade-off between task accuracy and tool-use efficiency through hypervolume-guided reward modulation and Pareto-ranking-based advantages (Li et al., 2026a). These methods share our use of multiple outcomes, while addressing different learning goals. In our setting, the goal is to produce a response satisfying all verified requirements. Pareto dominance identifies improvements among partially successful responses without specifying how gains on one requirement should compensate for losses on another.

Preference optimization with structured comparisons. Preference optimization provides a way to learn from comparisons among responses. DPO parameterizes comparison probabilities through policy-to-reference log-probability ratios (Rafailov et al., 2023). IRPO and DPO-VP apply DPO-based objectives to correct–incorrect response pairs, establishing a connection between verification and preference learning (Pang et al., 2024; Tu et al., 2025). PRO and LiPO extend preference optimization to ranked response lists (Song et al., 2024; Liu et al., 2025). Listwise Policy Optimization also learns from response groups in RLVR, using scalar rewards to construct target distributions for policy projection (Qu et al., 2026). GraphDPO represents response comparisons as a directed acyclic graph and aggregates local choice losses, each contrasting a response with the responses it dominates (Liu et al., 2026b). It therefore shares POLPO's use of structured comparisons without requiring a complete ordering. The distinction lies in the objective: GraphDPO aggregates choice probabilities over overlapping neighborhoods, whereas POLPO optimizes the probability that all observed comparisons hold jointly in a single Plackett–Luce ranking. For general partial orders, these local losses do not constitute a factorization of the joint ranking-event likelihood.

Plackett–Luce models have also been used to learn from supervision that leaves some ranking decisions unspecified. Ma et al. (2021) study the marginalized likelihood of ordered partitions, where the order between groups is observed but the order within each group is unknown. In recommendation, Mult-DPO considers comparisons between sets of positive and negative items and derives a tractable multinomial surrogate that upper-bounds the corresponding marginalized PL loss (Zhu et al., 2026). POLPO builds on this probabilistic treatment of incomplete rankings. It uses verifier-induced Pareto comparisons to define the likelihood of a general partial order and optimizes that likelihood through policy-derived scores. This connects established ranking models to policy learning from multiple verifiers, with unspecified ordering decisions marginalized out of the training objective. See Appendix F for additional related work.

6 Conclusion

We studied how multiple verification outcomes can guide policy learning toward satisfying all requirements. POLPO represents verified improvements as Pareto comparisons and maximizes the probability that these comparisons hold jointly, marginalizing over unspecified ordering decisions. This formulation enables learning from partial success without prescribing trade-offs between requirements. In controlled program synthesis, POLPO reaches a high probability of complete success with fewer sampled programs than training with all-pass or pass-rate rewards. Coding and multi-instruction-following experiments further show improvements in complete requirement satisfaction, with coding gains extending to external benchmarks. These findings support partial-order supervision as an effective approach to learning from multiple verifiers.

Limitation. POLPO's learning signal depends on the comparisons available within each response group. When no response strictly Pareto-dominates another, the group provides no comparison signal, even if some requirements remain unsatisfied. This can limit learning when sampled responses satisfy different, conflicting subsets of requirements. An important direction for future work is to develop sampling strategies that expose verified improvements and provide more informative comparison supervision.

AI USE STATEMENT

In this work, we used generative AI tools (Claude Code and Codex) to clean and reformat datasets, support qualitative and thematic data analysis, assist in the writing of proofs, formulate mathematical claims, provide critical ingredients for proving mathematical claims, implement methods, and assist with translation. We have not used generative AI tools to generate synthetic datasets, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or interpret results, and the development of theoretical models or conceptual frameworks is not applicable to this work. Additionally, we used generative AI tools to support literature reviews, create scientific figures, assist with drafting parts of the research paper, summarize and analyse existing literature, brainstorm ideas, source and search for information, and edit the research paper to improve readability. We have reviewed all AI-assisted work. The author personally reviewed the AI-assisted mathematical claims and proofs, method implementations, data processing and qualitative analyses, literature summaries, translations, figures, and manuscript text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies a training objective for language models using existing benchmark datasets and programmatically constructed tasks. Automatic verifiers assess only specified requirements and may contain errors; satisfying these checks does not guarantee that generated responses are safe, fair, or appropriate for deployment. Improvements in code generation and instruction following may also facilitate harmful applications. Models trained with this objective therefore require safety evaluation and safeguards appropriate to their intended use.

REPRODUCIBILITY STATEMENT

Section 2 defines the verification outcomes and scalar rewards. Section 3 defines the comparison rules, policy scores, and training objective. See Appendix B for the derivations and proofs for the likelihood and its gradients. See Appendix C for the training algorithm, exact likelihood computation, and the conditions under which response groups are included in training. See Appendix D for the datasets, preprocessing, training configurations, and evaluation procedures. Our code is available in an anonymized repository at `https://anonymous.4open.science/r/POLPO-2665/`.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. doi: 10.48550/arXiv.2108.07732. URL https://arxiv.org/abs/2108.07732.

Ryan Bahlous-Boldi, Isha Puri, Idan Shenfeld, Akarsh Kumar, Mehul Damani, Sebastian Risi, Omar Khattab, Zhang-Wei Hong, and Pulkit Agrawal. Vector policy optimization: Training for diversity improves test-time search. *arXiv preprint arXiv:2605.22817*, 2026. doi: 10.48550/arXiv.2605.22817. URL https://arxiv.org/abs/2605.22817.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3–4):324–345, 1952. doi: 10.1093/biomet/39.3-4.324. URL https://doi.org/10.1093/biomet/39.3-4.324.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pp. 129–136. Association for Computing Machinery, 2007. doi: 10.1145/1273496.1273513. URL https://doi.org/10.1145/1273496.1273513.

Yuxin Chen, Junfei Tan, An Zhang, Zhengyi Yang, Leheng Sheng, Enzhi Zhang, Xiang Wang, and Tat-Seng Chua. On softmax direct preference optimization for recommendation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 27463–27489. Curran Associates, Inc., 2024. doi: 10.52202/079017-0863. URL https://proceedings.neurips.cc/paper

_files/paper/2024/hash/30732ddb12d9faf7180f5d0e8b5b5da7-Abstract-Conference.html.

Tingchen Fu, Yafu Li, Jiawei Gu, Xiaoye Qu, and Yu Cheng. Scaling reasoning, losing control: Evaluating instruction following in large reasoning models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40445–40463. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1878. URL https://aclanthology.org/2026.acl-long.1878/.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

Taneesh Gupta, Rahul Madhavan, Xuchao Zhang, Nagarajan Natarajan, Chetan Bansal, and Saravan Rajmohan. Multi-preference optimization: Generalizing DPO via set-level contrasts. *arXiv preprint arXiv:2412.04628*, 2024. doi: 10.48550/arXiv.2412.04628. URL https://arxiv.org/abs/2412.04628.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021a. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. REINFORCE++: Stabilizing critic-free policy optimization with global advantage normalization. *arXiv preprint arXiv:2501.03262*, 2025. doi: 10.48550/arXiv.2501.03262. URL https://arxiv.org/abs/2501.03262.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Junyi Li, Xiaowei Qian, Yingyi Zhang, Wenlin Zhang, Guojing Li, Sheng Zhang, Xiao Han, Yichao Wang, and Xiangyu Zhao. Towards Pareto-optimal tool-integrated agents with Pareto ranking policy optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=9sg34nj13o.

Xin-Ye Li, Ren-Biao Liu, Yun-Ji Zhang, Hui Sun, Zheng Xie, and Ming Li. Exploring pass-rate reward in reinforcement learning for code generation. *arXiv preprint arXiv:2605.02944*, 2026b. doi: 10.48550/arXiv.2605.02944. URL https://arxiv.org/abs/2605.02944.

Ao Liu, Zhibing Zhao, Chao Liao, Pinyan Lu, and Lirong Xia. Learning Plackett-Luce mixtures from partial preferences. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 4328–4335, 2019. doi: 10.1609/aaai.v33i01.33014328.

Haotian Liu, Yihao Liu, Jingwei Ni, Siyuan Huang, Xinpeng Liu, Pengyu Cheng, Jiajun Song, Ruijin Ding, Junfeng Li, Zhechao Yu, Mengyu Zhou, Hongteng Xu, Xiaoxi Jiang, and Guanjun Jiang. GD^{2}PO: Mitigating multi-reward conflicts via group-dynamic reward-decoupled policy optimization. *arXiv preprint arXiv:2606.16771*, 2026a. doi: 10.48550/arXiv.2606.16771. URL https://arxiv.org/abs/2606.16771.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://papers.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Ning Liu, Chuanneng Sun, Kristina Klinkner, and Shervin Malmasi. Beyond pairs: Your language model is secretly optimizing a preference graph. *arXiv preprint arXiv:2605.08037*, 2026b. doi: 10.48550/arXiv.2605.08037. URL https://arxiv.org/abs/2605.08037.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. GDPO: Group reward-decoupled normalization policy optimization for multi-reward RL optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026c. URL https://openreview.net/forum?id=Y1rHxA58If.

Tianqi Liu, Zhen Qin, Junru Wu, Jiaming Shen, Misha Khalman, Rishabh Joshi, Yao Zhao, Mohammad Saleh, Simon Baumgartner, Jialu Liu, Peter J. Liu, and Xuanhui Wang. LiPO: Listwise preference optimization through learning-to-rank. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2404–2420. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.121. URL https://aclanthology.org/2025.naacl-long.121/.

R. Duncan Luce. *Individual Choice Behavior: A Theoretical Analysis*. John Wiley & Sons, New York, 1959. URL https://books.google.com/books?id=a80DAQAAIAAJ.

Jiaqi Ma, Xinyang Yi, Weijing Tang, Zhe Zhao, Lichan Hong, Ed Chi, and Qiaozhu Mei. Learning-to-rank with partitioned preference: Fast estimation for the Plackett–Luce model. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 928–936. PMLR, 2021. URL https://proceedings.mlr.press/v130/ma21a.html.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. doi: 10.52202/068431-2011. URL https://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 116617–116637. Curran Associates, Inc., 2024. doi: 10.52202/079017-3702. URL https://papers.nips.cc/paper_files/paper/2024/hash/d37c9ad425fe5b65304d500c6edcba00-Abstract-Conference.html.

Kirill Pavlenko, Alexander Golubev, Simon Karasik, and Boris Yangel. Blockwise advantage estimation for multi-objective RL with verifiable rewards. *arXiv preprint arXiv:2602.10231*, 2026. doi: 10.48550/arXiv.2602.10231. URL https://arxiv.org/abs/2602.10231.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. VerIF: Verification engineering for reinforcement learning in instruction following. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 30324–30339. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1542. URL https://aclanthology.org/2025.emnlp-main.1542/.

R. L. Plackett. The analysis of permutations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 24(2):193–202, 1975. doi: 10.2307/2346567. URL https://academic.oup.com/jrsssc/article/24/2/193/6953554.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1645. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/46499a0622ecf568b72d17b61e45dbd5-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Yun Qu, Qi Wang, Yixiu Mao, Heming Zou, Yuhang Jiang, Yingyue Li, Wutong Xu, Lizhou Cai, Weijie Liu, Clive Bai, Kai Yang, Yangkun Chen, Saiyong Yang, and Xiangyang Ji. Listwise policy optimization: Group-based RLVR as target-projection on the LLM response simplex. *arXiv preprint arXiv:2605.06139*, 2026. doi: 10.48550/arXiv.2605.06139. URL https://arxiv.org/abs/2605.06139.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741. Curran Associates, Inc., 2023. doi: 10.52202/075280-2338. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.

Mohammad Mahdi Salmani-Zarchi, Zahra Rahimi, Heshaam Faili, and Mohammad Javad Dousti. MDP-GRPO: Stabilized group relative policy optimization for multi-constraint instruction following. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 42784–42797. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1982. URL https://aclanthology.org/2026.acl-long.1982/.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. doi: 10.48550/arXiv.1707.06347. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. doi: 10.48550/arXiv.2402.03300. URL https://arxiv.org/abs/2402.03300.

Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. Preference ranking optimization for human alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 18990–18998, 2024. doi: 10.1609/aaai.v38i17.29865. URL https://ojs.aaai.org/index.php/AAAI/article/view/29865.

Songjun Tu, Jiahao Lin, Xiangyu Tian, Qichao Zhang, Linjing Li, Yuqian Fu, Nan Xu, Wei He, Xiangyuan Lan, Dongmei Jiang, and Dongbin Zhao. Enhancing LLM reasoning with iterative DPO: A comprehensive empirical investigation. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=OgWh4J7bkT.

Longwen Wang, Yirui Liu, Xuan'er Wu, Xiaohui Hu, Yuankai Fan, Kaidong Yu, Qizhen Weng, Wei Xi, and Xuelong Li. Beyond binary: Turning partial success into dense verifiable rewards for reinforcement learning in code generation. *arXiv preprint arXiv:2601.03525*, 2026. doi: 10.48550/arXiv.2601.03525. URL https://arxiv.org/abs/2601.03525.

Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 1192–1199. Association for Computing Machinery, 2008. doi: 10.1145/1390156.1390306. URL https://doi.org/10.1145/1390156.1390306.

Junjie Ye, Caishuang Huang, Zhuohan Chen, Wenjie Fu, Chenyuan Yang, Leyi Yang, Yilong Wu, Peng Wang, Meng Zhou, Xiaolong Yang, Tao Gui, Qi Zhang, Zhongchao Shi, Jianping Fan, and Xuanjing Huang. MulDimIF: A multi-dimensional constraint framework for evaluating and improving instruction following in large language models. In *Findings of the Association for*

Computational Linguistics: ACL 2026, pp. 2078–2104. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.99. URL https://aclanthology.org/2026.findings-acl.99/.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. AceCoder: Acing coder RL via automated test-case synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12023–12040. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.587. URL https://aclanthology.org/2025.acl-long.587/.

Kongcheng Zhang, Qi Yao, Shunyu Liu, Wenjian Zhang, Min Cen, Yang Zhou, Wenkai Fang, Yiru Zhao, Baisheng Lai, and Mingli Song. Replay failures as successes: Sample-efficient reinforcement learning for instruction following. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=YiTfnJursw.

Mian Zhang, Yueqin Yin, Kaiyu He, Peilin Wu, Xinlu Zhang, Mingyuan Zhou, and Zhiyu Zoey Chen. Mitigating exploration bias in RL for multi-instruction following. *arXiv preprint arXiv:2608.23830*, 2026b. doi: 10.48550/arXiv.2608.23830. URL https://arxiv.org/abs/2608.23830. Accepted to EMNLP 2026.

Guang Zhao, Byung-Jun Yoon, Gilchan Park, Shantenu Jha, Shinjae Yoo, and Xiaoning Qian. Pareto prompt optimization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HGCk5aaSvE.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. doi: 10.48550/arXiv.2311.07911. URL https://arxiv.org/abs/2311.07911.

Yaochen Zhu, Harald Steck, James McInerney, Aditya Sinha, Yinhan He, Nathan Kallus, and Jundong Li. Mult-DPO: Multinomial direct preference optimization for recommender systems. *arXiv preprint arXiv:2606.10078*, 2026. doi: 10.48550/arXiv.2606.10078. URL https://arxiv.org/abs/2606.10078.

A Policy Optimization Preliminaries

This appendix specifies the sequence-log-probability convention and GRPO surrogate, derives the correspondence between implicit rewards and KL-regularized optimal policies, and shows how the partial-order likelihood recovers DPO for a single comparison.

A.1 Sequence Log Probabilities

For a response of length T_i, we use the sequence log probability

$$\log \pi_\theta(y_i \mid x) = \sum_{t=1}^{T_i} \log \pi_\theta(y_{i,t} \mid x, y_{i,<t}), \tag{11}$$

where $y_{i,<t}$ denotes preceding response tokens. All policy scores in this paper use this sum over tokens.

A.2 Group Relative Policy Optimization

GRPO samples a response group from a rollout policy π_{old}, a snapshot held fixed during the ensuing updates (Shao et al., 2024). Given scalar rewards $\mathbf{r} = (r_1, \ldots, r_N)$, it computes

$$\bar{r} = \frac{1}{N} \sum_{j=1}^{N} r_j, \qquad \widehat{A}_i^{\text{GRPO}} = \frac{r_i - \bar{r}}{\text{std}(\mathbf{r}) + \delta}, \tag{12}$$

where $\text{std}(\mathbf{r})$ is the group standard deviation and $\delta > 0$ is a stabilizer. Each token in y_i receives the same advantage: responses above the group mean have positive advantages, and those below it have

negative advantages. GRPO uses these weights in a clipped policy surrogate, with an additional KL penalty in its original formulation. The surrogate and its gradient are given below.

The policy update uses the token probability ratio

$$q_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid x, y_{i,<t})}{\pi_{\text{old}}(y_{i,t} \mid x, y_{i,<t})}. \tag{13}$$

For one response group, the clipped policy surrogate is

$$\mathcal{J}^{\text{clip}}_{\text{GRPO}}(\theta; x, Y) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{t=1}^{T_i}\min\Big\{ q_{i,t}(\theta)\,\widehat{A}_i^{\text{GRPO}}, \\ \text{clip}\big(q_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\big)\,\widehat{A}_i^{\text{GRPO}}\Big\}, \tag{14}$$

where $\varepsilon > 0$ controls clipping, as in proximal policy optimization (Schulman et al., 2017). GRPO maximizes this surrogate averaged over prompts and rollout groups, with an additional KL penalty to a reference policy in the original formulation. The comparison with POLPO concerns the clipped policy term alone (see Appendix B.2).

Proposition 1. *At $\theta = \theta_{\text{old}}$, the gradient of the clipped surrogate Eq. (14) is*

$$\nabla_\theta \mathcal{J}^{\text{clip}}_{\text{GRPO}}(\theta; x, Y)\Big|_{\theta=\theta_{\text{old}}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\widehat{A}_i^{\text{GRPO}}}{T_i}\,\nabla_\theta \log \pi_\theta(y_i \mid x)|_{\theta=\theta_{\text{old}}}. \tag{15}$$

Proof. The advantages $\widehat{A}_i^{\text{GRPO}}$ depend only on the rewards and are constants. At $\theta = \theta_{\text{old}}$ every ratio equals $q_{i,t}(\theta_{\text{old}}) = 1$, which lies strictly inside the clipping interval $[1-\varepsilon, 1+\varepsilon]$. In a neighborhood of this point the clipping function is the identity, so both arguments of the minimum in Eq. (14) coincide with $q_{i,t}(\theta)\widehat{A}_i^{\text{GRPO}}$, and the surrogate is differentiable with

$$\nabla_\theta\big[q_{i,t}(\theta)\widehat{A}_i^{\text{GRPO}}\big] = \widehat{A}_i^{\text{GRPO}}\,q_{i,t}(\theta)\,\nabla_\theta \log \pi_\theta(y_{i,t} \mid x, y_{i,<t}).$$

Evaluating at θ_{old} sets $q_{i,t} = 1$. Summing over tokens and using the sequence log probability Eq. (11) gives $\frac{1}{N}\sum_i\big(\widehat{A}_i^{\text{GRPO}}/T_i\big)\nabla_\theta \log \pi_\theta(y_i \mid x)$ at θ_{old}, which is Eq. (15). □

A.3 Implicit Rewards and KL-Regularized Optimal Policies

For a fixed prompt x and reward function $r(x, y)$, consider the KL-regularized objective

$$\mathcal{J}_r(q \mid x) = \mathbb{E}_{y\sim q}[r(x, y)] - \beta D_{\text{KL}}(q \,\|\, \pi_{\text{ref}}(\cdot \mid x)), \tag{16}$$

with $\beta > 0$. The distributions are over complete responses, rather than just the sampled group Y. Assume that the reference and policies under consideration have common positive support, the partition function below is finite, and the displayed expectations and divergences are finite. Define

$$\pi_r^*(y \mid x) = \frac{\pi_{\text{ref}}(y \mid x)\exp(r(x, y)/\beta)}{Z_r(x)}, \qquad Z_r(x) = \sum_y \pi_{\text{ref}}(y \mid x)\exp(r(x, y)/\beta). \tag{17}$$

Substituting this expression into Eq. (16) gives

$$\mathcal{J}_r(q \mid x) = \beta \log Z_r(x) - \beta D_{\text{KL}}(q \,\|\, \pi_r^*(\cdot \mid x)). \tag{18}$$

The reward is fixed while optimizing over q, so $Z_r(x)$ is constant. Nonnegativity of KL divergence shows that the unique maximizing response distribution is $q = \pi_r^*(\cdot \mid x)$. Rearranging Eq. (17) yields the inverse reward parameterization used in DPO (Rafailov et al., 2023):

$$r(x, y) = \beta \log \frac{\pi_r^*(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z_r(x). \tag{19}$$

The final term depends only on the prompt and cancels from comparison probabilities.

Conversely, fix θ and set $r_\theta^{\text{imp}}(x, y) = s_\theta(x, y) + C(x)$, using the score in Eq. (6) and any finite prompt-dependent $C(x)$. Then $Z_{r_\theta^{\text{imp}}}(x) = \exp(C(x)/\beta)$ and $\pi^*_{r_\theta^{\text{imp}}} = \pi_\theta$, since $\sum_y \pi_\theta(y \mid x) = 1$. Equivalently,

$$\mathcal{J}_{r_\theta^{\text{imp}}}(q \mid x) = C(x) - \beta D_{\text{KL}}(q \,\|\, \pi_\theta(\cdot \mid x)), \tag{20}$$

which proves Eq. (7) by taking $C(x) = 0$. This establishes the policy–reward correspondence for each fixed θ. During POLPO training, θ changes through comparison-likelihood optimization, so the associated implicit reward changes with it; the identity does not assert maximization of a fixed scalar verifier reward. Rescaling the reward and KL coefficient by the same positive factor preserves the optimal policy. Taking $C(x) = 0$ and dividing both by β gives an equivalent unit-KL interpretation with reward $\log(\pi_\theta/\pi_{\mathrm{ref}})$; the PL likelihood continues to use the score $s_\theta = \beta \log(\pi_\theta/\pi_{\mathrm{ref}})$.

A.4 Connection to Direct Preference Optimization

For a group containing two responses $Y = (y_w, y_l)$ and the comparison $y_w \succ y_l$, the partial-order event contains only the ranking that places y_w first. Writing $s_w = s_\theta(x, y_w)$ and $s_l = s_\theta(x, y_l)$ using Eq. (6), its PL probability is the Bradley–Terry comparison probability (Bradley & Terry, 1952):

$$p_\theta^{\mathrm{pair}}(y_w \succ y_l \mid x) = \frac{e^{s_w}}{e^{s_w} + e^{s_l}} = \frac{1}{1 + \exp[-(s_w - s_l)]}. \tag{21}$$

Taking the negative logarithm gives the DPO loss (Rafailov et al., 2023),

$$\ell_{\mathrm{DPO}}(\theta; x, y_w, y_l) = -\log p_\theta^{\mathrm{pair}}(y_w \succ y_l \mid x) = \log\big(1 + \exp[-(s_w - s_l)]\big), \tag{22}$$

where $s_w - s_l = \beta\big[\log(\pi_\theta(y_w \mid x)/\pi_{\mathrm{ref}}(y_w \mid x)) - \log(\pi_\theta(y_l \mid x)/\pi_{\mathrm{ref}}(y_l \mid x))\big]$. A prompt-dependent additive constant in the implicit reward cancels from this difference and, more generally, from every PL choice probability. Thus the score in Eq. (6) can be used directly in the likelihood.

The same pairwise likelihood results when a larger response group has only one specified comparison, by Lemma 2. If every position is specified, the partial-order likelihood contains a single PL term and recovers the listwise extension of DPO (Rafailov et al., 2023, Appendix A.3). Verification-derived comparison targets, such as correct–incorrect response pairs used by DPO-based reasoning methods (Pang et al., 2024; Tu et al., 2025), fit within this likelihood formulation.

B Derivations and Proofs

This appendix collects the derivations behind Section 3. Throughout, $Y = (y_1, \ldots, y_N)$ is a fixed response group, $s_i = s_\theta(x, y_i)$ are the policy scores in Eq. (6), $w_i = e^{s_i}$, and $\mathcal{R}_V$ is a strict partial order on $\{1, \ldots, N\}$ with linear extensions $\mathrm{LE}(\mathcal{R}_V)$. For a complete ranking σ, $R_k(\sigma) = \{\sigma_k, \ldots, \sigma_N\}$ denotes the candidates remaining at position k, and

$$p_i^{(k)}(\sigma) = \frac{w_i}{\sum_{j \in R_k(\sigma)} w_j} \qquad \text{for } i \in R_k(\sigma) \tag{23}$$

is the probability of choosing y_i at position k given the remaining candidates. Set $p_i^{(k)}(\sigma) = 0$ for $i \notin R_k(\sigma)$.

B.1 Gradient of the Plackett–Luce Log-Likelihood

Proposition 2 (Complete-ranking coefficients)**.** *For every complete ranking σ and every $i \in \{1, \ldots, N\}$,*

$$a_i(\sigma) = \frac{\partial}{\partial s_i} \log p_\theta^{\mathrm{PL}}(\sigma \mid x, Y) = \sum_{k=1}^{N-1} \Big[\mathbf{1}\{i = \sigma_k\} - \mathbf{1}\{i \in R_k(\sigma)\}\, p_i^{(k)}(\sigma)\Big], \tag{24}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function. Moreover, $\sum_{i=1}^{N} a_i(\sigma) = 0$.

Proof. By Eq. (8), $\log p_\theta^{\mathrm{PL}}(\sigma \mid x, Y) = \sum_{k=1}^{N-1} \big[s_{\sigma_k} - \log \sum_{j \in R_k(\sigma)} e^{s_j}\big]$. The derivative of the k-th term with respect to s_i is $\mathbf{1}\{i = \sigma_k\}$ from the first part and $-\mathbf{1}\{i \in R_k(\sigma)\}\, e^{s_i} / \sum_{j \in R_k(\sigma)} e^{s_j}$ from the log-partition function, which gives the stated expression. Summing over i, position k contributes $1 - \sum_{i \in R_k(\sigma)} p_i^{(k)}(\sigma) = 0$, because the choice probabilities at each position sum to one over the remaining candidates. □

Since $i = \sigma_k$ implies $i \in R_m(\sigma)$ exactly for $m \le k$, the coefficient of the response at position k can also be written as $a_{\sigma_k}(\sigma) = 1 - \sum_{m=1}^{k} p_{\sigma_k}^{(m)}(\sigma)$ for $k \le N-1$ and $a_{\sigma_N}(\sigma) = -\sum_{m=1}^{N-1} p_{\sigma_N}^{(m)}(\sigma)$ for the last response. For $k < N$, the coefficient is positive exactly when $\sum_{m=1}^{k} p_{\sigma_k}^{(m)}(\sigma) < 1$, zero when this sum equals one, and negative when it exceeds one. For positive weights and $N \ge 2$, the final response has a strictly negative coefficient. The sum here combines choice probabilities from different positions; it is not a probability that the response has already been selected.

B.2 Gradient of the Partial-Order Likelihood

Proposition 3 (Partial-order coefficients)**.** *Let $p_\theta(\mathcal{R}_V \mid x, Y) = \sum_{\sigma \in \mathrm{LE}(\mathcal{R}_V)} p_\theta^{\mathrm{PL}}(\sigma \mid x, Y)$ as in Eq. (10). Then*

$$A_i^{\mathrm{rank}} = \frac{\partial}{\partial s_i} \log p_\theta(\mathcal{R}_V \mid x, Y) = \sum_{\sigma \in \mathrm{LE}(\mathcal{R}_V)} \omega_\theta(\sigma)\, a_i(\sigma), \qquad \omega_\theta(\sigma) = \frac{p_\theta^{\mathrm{PL}}(\sigma \mid x, Y)}{p_\theta(\mathcal{R}_V \mid x, Y)}, \tag{25}$$

where the weights $\omega_\theta(\sigma)$ are the conditional probabilities of the compatible rankings given that the comparisons hold and sum to one. Consequently $\sum_{i=1}^{N} A_i^{\mathrm{rank}} = 0$, and

$$\nabla_\theta \log p_\theta(\mathcal{R}_V \mid x, Y) = \beta \sum_{i=1}^{N} A_i^{\mathrm{rank}}\, \nabla_\theta \log \pi_\theta(y_i \mid x). \tag{26}$$

Proof. The set $\mathrm{LE}(\mathcal{R}_V)$ does not depend on θ, so $\partial_{s_i} p_\theta(\mathcal{R}_V \mid x, Y) = \sum_{\sigma \in \mathrm{LE}(\mathcal{R}_V)} \partial_{s_i} p_\theta^{\mathrm{PL}}(\sigma \mid x, Y) = \sum_{\sigma \in \mathrm{LE}(\mathcal{R}_V)} p_\theta^{\mathrm{PL}}(\sigma \mid x, Y)\, a_i(\sigma)$, using $\partial_{s_i} p_\theta^{\mathrm{PL}} = p_\theta^{\mathrm{PL}}\, \partial_{s_i} \log p_\theta^{\mathrm{PL}}$. Dividing by $p_\theta(\mathcal{R}_V \mid x, Y)$ gives the mixture form, and the weights sum to one by definition. The zero-sum property follows from Proposition 2: $\sum_i A_i^{\mathrm{rank}} = \sum_\sigma \omega_\theta(\sigma) \sum_i a_i(\sigma) = 0$. For the policy gradient, the reference policy is fixed, so $\nabla_\theta s_i = \beta\, \nabla_\theta \log \pi_\theta(y_i \mid x)$ by Eq. (6), and the chain rule gives $\nabla_\theta \log p_\theta(\mathcal{R}_V \mid x, Y) = \sum_i \big(\partial_{s_i} \log p_\theta(\mathcal{R}_V \mid x, Y)\big) \nabla_\theta s_i$. □

The likelihood thus supplies a coefficient for each response gradient, depending on both the verified comparisons and the current policy scores. These coefficients are not standardized advantages and need not have unit variance; their scale comes from the likelihood. Their zero-sum property also follows from the invariance of ranking probabilities to a common shift of all scores.

Connection to GRPO updates. Both the POLPO gradient in Eq. (26) and the GRPO surrogate gradient at $\theta = \theta_{\mathrm{old}}$ in Eq. (15) weight response gradients using relative information from the group. GRPO derives advantages from scalar rewards; POLPO derives its coefficients from the joint comparison likelihood. This is a connection between update structures, not an equivalence of objectives. GRPO averages token contributions and uses clipping, whereas POLPO uses sequence log probabilities and directly optimizes the likelihood, with no additional explicit KL term in Eq. (10). See Appendix A.2 for the full GRPO surrogate and the conditions for Eq. (15).

B.3 Marginalization over Unspecified Positions

Two facts about the Plackett–Luce model explain the marginalization in Eq. (9) and the additional example below. The first is that the probabilities of all orderings of a set of candidates sum to one. The second is that the marginal probability that one candidate precedes another has the Bradley–Terry form.

Lemma 1 (Completions sum to one)**.** *For any nonempty $R \subseteq \{1, \dots, N\}$,*

$$\sum_{\tau} \prod_{k=1}^{|R|-1} \frac{w_{\tau_k}}{\sum_{m=k}^{|R|} w_{\tau_m}} = 1,$$

where the sum ranges over all orderings $\tau = (\tau_1, \dots, \tau_{|R|})$ of R.

Proof. By induction on $|R|$. For $|R| = 1$ the product is empty and equals one. For $|R| > 1$, group the orderings by their first element $i \in R$: the sum equals $\sum_{i \in R} \big(w_i / \sum_{j \in R} w_j\big)\, S(R \setminus \{i\})$, where $S(R \setminus \{i\})$ denotes the same sum over orderings of $R \setminus \{i\}$ and equals one by the induction hypothesis. The remaining factor is $\sum_{i \in R} w_i / \sum_{j \in R} w_j = 1$. □

Lemma 2 (Pairwise marginal)**.** *For distinct i and j, the total Plackett–Luce probability of the complete rankings in which y_i precedes y_j is $w_i/(w_i + w_j)$.*

Proof. We use the standard sampling representation of the Plackett–Luce model. Draw independent exponential random variables $E_1, \ldots, E_N$ with rates $w_1, \ldots, w_N$ and rank the candidates by increasing E_m. The minimum is attained by candidate m with probability $w_m / \sum_{m'} w_{m'}$. Conditional on the minimum time and its attaining candidate, the memoryless property makes the residual waiting times of the remaining candidates independent exponentials with their original rates. Iterating over positions gives exactly the sequential choice probabilities in Eq. (8), so the induced distribution over rankings is the Plackett–Luce model. Under this representation, y_i precedes y_j if and only if $E_i < E_j$, which has probability $w_i/(w_i + w_j)$. □

Corollary 1 (Special cases of the partial-order likelihood)**.** *(i) If verification establishes no comparisons, then $\mathrm{LE}(\mathcal{R}_V)$ contains every ranking, $p_\theta(\mathcal{R}_V \mid x, Y) = 1$, and the loss and its gradient vanish. (ii) If y_1 is the only correct response among N responses and $\mathcal{R}_V = \{(1, j) : j \neq 1\}$, then $p_\theta(\mathcal{R}_V \mid x, Y) = w_1 / \sum_{j=1}^{N} w_j$. (iii) If $\mathcal{R}_V = \{(1, 3)\}$, then $p_\theta(\mathcal{R}_V \mid x, Y) = w_1/(w_1 + w_3)$, as in Eq. (9), so the loss equals the DPO loss between y_1 and y_3.*

Proof. (i) is Lemma 1 with $R = \{1, \ldots, N\}$. For (ii), every compatible ranking selects y_1 first, which has probability $w_1 / \sum_j w_j$, and the orderings of the remaining responses that follow sum to one by Lemma 1. (iii) is Lemma 2 with $i = 1$ and $j = 3$. □

Learning comparisons jointly. With multiple comparisons, the likelihood accounts for their dependence through shared responses. For example, suppose y_1 is correct and y_2, y_3 are incorrect, giving $\mathcal{R}_V = \{(1, 2), (1, 3)\}$. The two compatible rankings are $(1, 2, 3)$ and $(1, 3, 2)$: both place y_1 first, while leaving the incorrect responses unordered. Their total probability is

$$p_\theta(\mathcal{R}_V \mid x, Y) = \frac{e^{s_1}}{e^{s_1} + e^{s_2} + e^{s_3}}. \tag{27}$$

Once y_1 is selected first, the conditional probabilities of the two remaining orderings sum to one. At equal scores, the probability that y_1 precedes both other responses is $1/3$. Each pairwise comparison has probability $1/2$, so multiplying the two comparison probabilities as if they were independent would instead give $1/4$. The negative log-likelihood of the joint event therefore differs from the sum of the pairwise comparison losses.

Example policy updates. In Eq. (27), write $p_i = e^{s_i} / \sum_{j=1}^{3} e^{s_j}$. The coefficients are $(A_1^{\mathrm{rank}}, A_2^{\mathrm{rank}}, A_3^{\mathrm{rank}}) = (1 - p_1, -p_2, -p_3)$. At equal scores, they are $(2/3, -1/3, -1/3)$, favoring the correct response relative to both incorrect alternatives. As the model assigns greater probability to placing the correct response first, its positive coefficient decreases. Both incorrect responses receive a signal because each is compared with the correct response; no comparison between them is needed. By contrast, y_2 in Eq. (9) has zero score derivative because it participates in no comparison. A zero score derivative does not prevent its response probability from changing through shared policy parameters.

B.4 Computing the Partial-Order Likelihood

In the binary-correctness example in Eq. (27), we did not need to evaluate both complete rankings separately. Once the correct response was selected first, every order of the remaining responses was acceptable, and their probabilities summed to one. The same principle gives a recurrence for general partial-order likelihoods: many compatible rankings share the same remaining candidates, so their possible continuations can be handled together.

Let $R \subseteq \{1, \dots, N\}$ be the remaining candidate indices, write $w_i = e^{s_i}$, and let $H(R)$ be the probability of completing the ranking without violating a comparison. A candidate is eligible for the next position when none of its required predecessors remain. Denote these candidates by

$$\mathrm{Max}_{\mathcal{R}_V}(R) = \{i \in R : \text{there is no } j \in R \text{ with } (j, i) \in \mathcal{R}_V\}. \tag{28}$$

The likelihood can be computed recursively by adding the probabilities of the eligible next choices and their valid continuations:

$$H(\varnothing) = 1, \qquad H(R) = \sum_{i \in \mathrm{Max}_{\mathcal{R}_V}(R)} \frac{w_i}{\sum_{j \in R} w_j} H(R \setminus \{i\}). \tag{29}$$

The desired probability is $p_\theta(\mathcal{R}_V \mid x, Y) = H(\{1, \dots, N\})$. In the binary example, only y_1 is eligible initially, and $H(\{2, 3\}) = 1$ after it is selected. The recurrence therefore gives $H(\{1, 2, 3\}) = w_1/(w_1 + w_2 + w_3)$, as before. This also explains why the denominator must include all remaining candidates. Normalizing over the eligible set $\{1\}$ alone would always give probability one and remove the learning signal. The PL model can choose an incorrect first response; the likelihood measures the probability that it avoids doing so.

Dynamic programming stores each remaining-set value once, avoiding explicit enumeration of complete rankings. There are at most 2^N subset states and at most N outgoing transitions per state, so the general computation can remain exponential. The number of reachable states depends on the comparison structure. Evaluating the recurrence in log space with log-sum-exp operations also allows differentiation through the policy scores without listing the compatible rankings or their conditional probabilities.

For $R \subseteq \{1, \dots, N\}$, let $\mathrm{LE}_R(\mathcal{R}_V)$ be the set of orderings of R that satisfy every comparison $(i, j) \in \mathcal{R}_V$ with $i, j \in R$, and define

$$H^\star(R) = \sum_{\tau \in \mathrm{LE}_R(\mathcal{R}_V)} \prod_{k=1}^{|R|-1} \frac{w_{\tau_k}}{\sum_{m=k}^{|R|} w_{\tau_m}}, \qquad H^\star(\varnothing) = 1. \tag{30}$$

By Eq. (10), $H^\star(\{1, \dots, N\}) = p_\theta(\mathcal{R}_V \mid x, Y)$.

Proposition 4 (Recurrence). *$H^\star$ satisfies the recurrence Eq. (29). Hence the value $H(\{1, \dots, N\})$ computed by Eq. (29) equals $p_\theta(\mathcal{R}_V \mid x, Y)$.*

Proof. Fix a nonempty R. An ordering τ of R satisfies the comparisons within R if and only if its first element has no predecessor in R, that is, $\tau_1 \in \mathrm{Max}_{\mathcal{R}_V}(R)$, and the remaining ordering $(\tau_2, \dots, \tau_{|R|})$ satisfies the comparisons within $R \setminus \{\tau_1\}$. Grouping the orderings in Eq. (30) by their first element therefore gives

$$H^\star(R) = \sum_{i \in \mathrm{Max}_{\mathcal{R}_V}(R)} \frac{w_i}{\sum_{j \in R} w_j} H^\star(R \setminus \{i\}),$$

which is Eq. (29). Both H and $H^\star$ equal one on the empty set and satisfy the same recurrence, so they agree on every subset by induction on $|R|$. □

The subsets reachable from $\{1, \dots, N\}$ by repeatedly removing eligible candidates are exactly the subsets R that are closed under the relation in the following sense: if $i \in R$ and $(i, j) \in \mathcal{R}_V$, then $j \in R$. These subsets are the order ideals under the convention that a preferred candidate is greater, and the eligible candidates of an ideal are its maximal elements. A top-down implementation that memoizes reachable states and expands every nonempty state therefore visits one state per order ideal, including the empty set. This count equals 2^N only when $\mathcal{R}_V$ is empty, equals $N + 1$ when $\mathcal{R}_V$ is a complete ranking, and equals $1 + \sum_{g=1}^{M} (2^{|G_g|} - 1)$ for an ordered partition $G_1 \succ \cdots \succ G_M$. Early termination when no comparisons remain can reduce the number of evaluated states. These are counts of reachable recurrence states. The batched evaluation instead evaluates all subsets within each series component, grouped by cardinality (see Appendix C.3).

B.5 Ordered Partitions

Let $\mathcal{R}_V = G_1 \succ \cdots \succ G_M$ be an ordered partition of $\{1, \ldots, N\}$: every candidate in an earlier block precedes every candidate in a later block, and no ordering is specified within a block. Binary correctness and graded scalar feedback induce this structure. General Pareto relations need not do so. Write $W_{>g} = \sum_{h>g} \sum_{j \in G_h} w_j$ for the total weight of the blocks below G_g, with $W_{>M} = 0$. For $S \subseteq G_g$, define

$$F_g(S) = \sum_{\tau} \prod_{k=1}^{|S|} \frac{w_{\tau_k}}{\sum_{m=k}^{|S|} w_{\tau_m} + W_{>g}}, \qquad F_g(\varnothing) = 1, \tag{31}$$

where the sum ranges over all orderings τ of S. Thus $F_g(G_g)$ is the probability that the sequential choice process selects every candidate in G_g before any candidate in a lower block, given that all higher blocks have already been selected.

Proposition 5 (Block factorization)**.** *The likelihood factorizes as*

$$p_\theta(\mathcal{R}_V \mid x, Y) = \prod_{g=1}^{M-1} F_g(G_g). \tag{32}$$

Each factor satisfies

$$F_g(S) = \sum_{i \in S} \frac{w_i}{\sum_{j \in S} w_j + W_{>g}} F_g(S \setminus \{i\}). \tag{33}$$

Once the block weights $W_{>g}$ are computed, evaluating all factors by this recurrence takes $O\big(\sum_{g=1}^{M-1} |G_g|\, 2^{|G_g|}\big)$ arithmetic operations.

Proof. The linear extensions of an ordered partition are the concatenations $\sigma = (\tau^{(1)}, \ldots, \tau^{(M)})$ of arbitrary orderings $\tau^{(g)}$ of the blocks G_g. When position k of σ lies in block g, the remaining candidates are the unselected members of G_g together with all lower blocks, so the denominator of the choice probability in Eq. (8) is the weight of the unselected members of G_g plus $W_{>g}$. The product over positions therefore factorizes over blocks, and summing over the orderings of each block independently gives $\prod_{g=1}^{M} F_g(G_g)$. For the final block, $W_{>M} = 0$ and $F_M(G_M) = 1$ by Lemma 1. The recurrence follows by grouping the orderings of S by their first element, as in Proposition 4. The cost follows because block g has $2^{|G_g|}$ subset states with at most $|G_g|$ transitions each. □

Extension to series components. The same factorization applies when the ordered blocks retain internal comparisons. Form the undirected incomparability graph of $\mathcal{R}_V$, with an edge between distinct candidates i and j exactly when neither (i, j) nor (j, i) belongs to $\mathcal{R}_V$. Its connected components can be ordered as $C_1 \succ \cdots \succ C_M$, meaning that every candidate in C_g precedes every candidate in C_h whenever $g < h$. Unlike tie blocks, a component need not be an antichain: candidates linked by a path of incomparable pairs may still be comparable to one another. The original comparisons within each component must therefore be retained. For this decomposition, write $W_{>g} = \sum_{h>g} \sum_{j \in C_h} w_j$ and let $\mathcal{R}_V|_{C_g}$ be the restriction of $\mathcal{R}_V$ to C_g.

Proposition 6 (Series-component factorization)**.** *For $S \subseteq C_g$, start with S and all lower components as the remaining candidates. Let $Q_g(S)$ be the probability of selecting all candidates in S, in an order satisfying every comparison in $\mathcal{R}_V$ whose endpoints both lie in S, before selecting any candidate in a lower component. Then*

$$Q_g(\varnothing) = 1, \qquad Q_g(S) = \sum_{i \in \mathrm{Max}_{\mathcal{R}_V|_{C_g}}(S)} \frac{w_i}{\sum_{j \in S} w_j + W_{>g}} Q_g(S \setminus \{i\}), \tag{34}$$

and

$$p_\theta(\mathcal{R}_V \mid x, Y) = \prod_{g=1}^{M} Q_g(C_g). \tag{35}$$

For an ordered partition, the components are the tie blocks and $Q_g = F_g$. The final factor is one if the final component is an antichain. If it contains an internal comparison, its factor must be evaluated even though $W_{>M} = 0$.

Algorithm 1 Partial-Order Likelihood Policy Optimization (POLPO)

Input: Initial policy π_θ, fixed reference π_{ref}, prompt distribution ρ, verification function $\mathbf{c}$, comparison rule $\mathcal{A}$ (Pareto dominance by default), and score scale $\beta > 0$; group size $N \geq 2$, rollout batch size B_{roll}, training epochs per rollout E, rollout iterations T, and an optimizer.

for $t = 1, \ldots, T$ **do**
  $\pi_{\mathrm{old}} \leftarrow \pi_\theta$; hold this rollout policy fixed
  Sample prompts $x_1, \ldots, x_{B_{\mathrm{roll}}} \sim \rho$
  for $b = 1, \ldots, B_{\mathrm{roll}}$ **do**
    Sample $Y_b = (y_{b1}, \ldots, y_{bN})$ from $\pi_{\mathrm{old}}(\cdot \mid x_b)$
    Evaluate $\mathbf{c}_{bi} \leftarrow \mathbf{c}(x_b, y_{bi})$ for $i = 1, \ldots, N$
    $\mathcal{R}_{V,b} \leftarrow \mathcal{A}(\mathbf{c}_{b1}, \ldots, \mathbf{c}_{bN})$
  end for
  Prepare training groups using the inclusion conditions (see Appendix C.3)
  for $e = 1, \ldots, E$ **do**
    for each training batch $\mathcal{D}$ of response groups **do**
      Evaluate $\ell_{\mathrm{POLPO}}(\theta; x, Y, \mathcal{R}_V)$ from Eq. (10) for each $(x, Y, \mathcal{R}_V) \in \mathcal{D}$
      Update θ with the optimizer using the gradients of these group losses
    end for
  end for
end for
return π_θ

Proof. Candidates in distinct incomparability components are comparable. Their comparison direction is uniform across the two components: if a precedes b and a' is incomparable with a in the same component, then b cannot precede a', since transitivity would imply that a precedes a'. Thus a' precedes b. Propagating along incomparability paths in both components gives the same direction for every cross-component pair. Transitivity then orders the components totally.

Every compatible full ranking is consequently a concatenation of compatible internal rankings of $C_1, \ldots, C_M$. Within component g, the remaining candidates consist of its unselected members and all lower components. Grouping internal rankings by their first candidate gives Eq. (34); its eligible set enforces all internal comparisons. Factoring each full-ranking probability over components and summing separately over their compatible internal rankings gives Eq. (35), without assuming independence of the comparison events. In a final antichain, all internal rankings are allowed and their probabilities sum to one by Lemma 1. If the final component contains a comparison, some internal rankings are forbidden and have positive PL probability because all weights are positive, so its factor is strictly less than one. □

C Likelihood Computation and Implementation Details

This appendix describes how we evaluate and optimize the POLPO loss in Eq. (10). Scalar verifier scores induce an ordered partition of tied responses, whereas the Pareto rule in Eq. (5) preserves the comparisons supported by individual verification outcomes. Both cases use the same partial-order likelihood.

C.1 POLPO Training Algorithm

POLPO alternates between collecting responses and updating the policy (see Algorithm 1). At each rollout iteration, a frozen behavior policy π_{old} generates N responses for each of B_{roll} prompts. Verification determines the comparison relation within each response group. The collected groups are used for E training epochs, with the group loss defined in Eq. (10).

The reference policy π_{ref} remains fixed throughout training. Sampled responses, verifier outcomes, comparison relations, and reference log probabilities are treated as constants during policy updates. Current-policy scores are recomputed at each update, and gradients flow through these scores and the likelihood calculation. Sampling, verification, and group selection are not differentiated.

Likelihood evaluation. The group loss in Eq. (10) is evaluated using the series-component factorization in Proposition 6, with each required factor computed by a subset recurrence in log space.

Policy scores are recomputed from the current parameters according to Eq. (6). At every step of the recurrence, the choice denominator includes all remaining candidates, including candidates that cannot be selected next without violating the relation. The computation is differentiable through the policy scores and does not enumerate complete rankings. An empty comparison relation has log-likelihood zero.

C.2 Log-Space Block Recurrence

For ordered partitions $G_1 \succ \cdots \succ G_M$, the evaluator uses the factors in Proposition 5. Equal scalar verifier scores form a tie block; binary correctness yields at most two blocks. To evaluate the factors stably, let $\bar{s}_{>g} = \log W_{>g}$ and $\phi_g(S) = \log F_g(S)$. For a non-final block, $\bar{s}_{>g}$ is the log-sum-exp of the scores of all lower candidates. With $\phi_g(\varnothing) = 0$, the log-space recurrence is

$$\phi_g(S) = \operatorname*{logsumexp}_{i \in S} \Big[s_i - \operatorname{logsumexp}\big((s_j)_{j \in S}, \bar{s}_{>g}\big) + \phi_g(S \setminus \{i\}) \Big]. \tag{36}$$

The inner log-sum-exp includes one entry for every remaining candidate and one entry for the aggregate weight of the lower blocks; repeated score values retain their multiplicities. The final tie block contributes a factor of one and is omitted, giving $\log p_\theta(\mathcal{R}_V \mid x, Y) = \sum_{g=1}^{M-1} \phi_g(G_g)$.

The implementation enumerates all subsets of a processed block as bit masks, groups them by cardinality, and caches score-independent index and membership tables for each block size and device. All subsets of a given cardinality are evaluated together by tensor gathers and log-sum-exp operations. Processing cardinalities in increasing order ensures that the values for $S \setminus \{i\}$ are available when computing the value for S.

Automatic differentiation through this recurrence and the sequence-summed scores yields the coefficients in Proposition 3. Differentiating the log-likelihood gives A_i^{rank}; differentiating its negative gives $-A_i^{\mathrm{rank}}$. Neither linear extensions nor their conditional probabilities are explicitly constructed. Reference log probabilities are computed for the sampled responses and treated as constants.

C.3 General Partial Orders

For binary multi-verifier feedback, Pareto dominance in Eq. (5) is equivalent to strict inclusion between sets of satisfied constraints. These sets are represented as integer bit masks for efficient comparison. Every response used to construct the relation must have recorded verifier outcomes; missing outcomes are not interpreted as failed checks.

The evaluator constructs the incomparability graph of the retained candidates and finds its connected components $C_1, \ldots, C_M$. As shown in Proposition 6, these components can be ordered so that every candidate in an earlier component dominates every candidate in a later one. Internal comparisons within a component are retained. This decomposition does not impose additional comparisons between incomparable responses.

Each component factor Q_g is evaluated using Eq. (34). In log space, the update has the form of Eq. (36), with the outer sum restricted to $\mathrm{Max}_{\mathcal{R}_V}(S)$ and the denominator unchanged. If there are no lower components, the lower-weight term is omitted from the denominator. A final component is omitted only when it contains no internal comparison; a final component with internal comparisons still requires evaluation. For example, if the only comparison is $y_1 \succ y_3$ and y_2 is incomparable with both, all three candidates belong to one component. Its factor is evaluated and equals $w_1/(w_1 + w_3)$.

Vectorized evaluation and computational cost. The vectorized computation evaluates all 2^m subsets of each processed component of size m, rather than traversing only the reachable order ideals (see Appendix B.4). For each subset, candidates with a predecessor still present are excluded from the eligible set. Values for unreachable subsets are computed but do not contribute to the final component likelihood. The denominator retains all candidates in the subset and all lower components.

For a component without internal comparisons, the subset transitions require $O(m2^m)$ arithmetic operations after the lower-component weight is available. For a component with internal comparisons, determining the eligible candidates uses a dense matrix multiplication between subset membership indicators and the dominance matrix; this requires $O(m^2 2^m)$ arithmetic operations. The

cached index and membership tables contain $O(m2^m)$ entries. These implementation costs differ from counting only the reachable states of a top-down recurrence.

Evaluation limits and candidate filtering. Groups with missing or empty responses are excluded before any optional filtering. An optional filter removes responses that reach the generation-length limit; the likelihood is then evaluated on the remaining responses and their induced partial order. Groups with fewer than two responses or no strict comparison contribute zero loss. Each series component requiring dynamic programming is limited to 16 candidates, and groups exceeding this limit are omitted. A final antichain requires no computation and is exempt. Consequently, the component-size limit cannot exclude a group when $N \leq 16$.

D Experimental Details

This appendix describes the datasets, training configurations, and evaluation protocols used in the experiments.

D.1 Controlled Program Synthesis

Each program is specified by three independent categorical choices (d_j, v_j), one for each branch $j \in \{1, 2, 3\}$. Each choice has $3(M-1)$ possible values, with $d_j \in \{2, \ldots, M\}$ and $v_j \in \{\texttt{Fizz}, \texttt{Buzz}, \texttt{FizzBuzz}\}$. The program executes the first satisfied branch:

```
def f(n):
    if n % d1 == 0:
        return v1
    elif n % d2 == 0:
        return v2
    elif n % d3 == 0:
        return v3
    return str(n)
```

Branch distributions are initialized uniformly, and duplicate choices are allowed. The log probability of a program is the sum of the log probabilities of its three branch choices. For target divisors (a, b), the intended output is `FizzBuzz` for a multiple of both divisors, `Fizz` or `Buzz` for a multiple of only one, and the input string otherwise. For the divisors and search spaces considered here, the two completely correct programs begin with $(ab, \texttt{FizzBuzz})$ and place $(a, \texttt{Fizz})$ and $(b, \texttt{Buzz})$ in either order afterward. Their initial total probability is $2/[3(M-1)]^3$.

Each integer input from 1 to 60 defines one verifier. The first sweep uses $(a, b) = (3, 5)$ and $M \in \{15, 30, 60\}$; the second uses $M = 60$ and $(a, b) \in \{(3, 5), (3, 7), (3, 11)\}$. For these three divisor pairs, respectively, 4, 2, and 1 of the 60 inputs are divisible by both divisors. The other input-category counts also change in the second sweep. The experiment therefore varies the specification and does not attribute differences solely to the frequency of overlap inputs.

The recorded runs use seed 42, 1,000 updates, and 64 independent groups of eight programs per update, or 512 new programs per update. Each group contains responses to the same synthesis problem; groups are independent draws from the current policy. Optimization uses Adam with learning rate 0.03, moment coefficients $(0.9, 0.999)$, and $\epsilon = 10^{-8}$. The GRPO analogue uses rewards from Eq. (4), centered and divided by the population standard deviation within each group, with no KL penalty. POLPO uses $\beta = 0.1$, the fixed initial uniform policy as reference, and the group loss in Eq. (10). The verifier relation is built from the complete 60-dimensional outcome vector.

Complete-success probability is evaluated exactly by summing the probabilities of the two correct programs after every update. The first update at which it reaches 0.9 defines the sample-efficiency statistic, multiplied by 512 to obtain the number of generated programs. No interpolation between recorded updates is needed. Input-specific success probabilities are also computed from the categorical distributions, so evaluation has no response-sampling noise; the training trajectory still depends on the seed. These experiments use one training seed and therefore do not estimate variation across training runs.

D.2 Coding Tasks

D.2.1 Datasets and Prompts

MBPP. We use the `full` configuration of `google-research-datasets/mbpp`, with the official 374-problem training split and 500-problem test split (Austin et al., 2021). Each problem has three test assertions. The prompt requests a self-contained Python implementation in a Markdown code block and includes the problem description and the first test as an example. All three assertions, including the displayed example, are used to verify the response. The released reference implementations are not used for supervised warmup or added to response groups. The 90-problem validation split is separate from the reported 500-problem test evaluation; results use the prescribed final checkpoint rather than a checkpoint selected by validation or test performance.

APPS Introductory. We use the Introductory tier of `codeparrot/apps` (Hendrycks et al., 2021a). Of its 2,639 training problems, 241 have no executable test cases and are excluded, leaving 2,398 training problems. All 1,000 Introductory test problems are retained. There is no further filtering by test count, and every available test is used. Prompts contain the problem statement and supplied starter code when present, together with instructions appropriate for standard-input/output or function-call evaluation. Test cases used by the checker are not appended to the prompt; examples already present in the problem statement are retained. The training and test problem statements have no exact overlap after whitespace normalization; this check does not establish the absence of semantic near-duplicates.

AceCode. We use the hard RL-training subset released with AceCoder (Zeng et al., 2025), distributed as `CodeDPO/rlhf_dataset_20250126_openrlhf_format_hard` at revision `f40aa9de97d882c6d411a8ca5e674ab6b1da2c12`. All 21,262 released training rows are used. The hard-subset selection is inherited from the release, rather than recomputed using our initial model. Each problem supplies a prompt and a list of test-code entries. Each entry defines one verifier and can contain setup code as well as assertions; entries execute in their supplied order within the generated program's namespace. Thus, a verifier corresponds to a supplied test entry, not necessarily a single assertion. The checker records all entries' outcomes, including those after a failure. We use neither teacher responses nor a supervised warmup stage. Because this release has no official test split, evaluation uses the external benchmarks described below.

D.2.2 Models and Optimization

MBPP initializes either `Qwen/Qwen3-0.6B` or `Qwen/Qwen3-4B`, using each model's native chat template with thinking enabled during both training and evaluation. APPS and AceCode initialize `Qwen/Qwen2.5-Coder-1.5B-Instruct`, also using its native chat template. The same initial weights and seed (42) are used across methods within each dataset and model size. All experiments perform full-parameter fine-tuning. Table 6 summarizes the training hyperparameters.

Each training round samples eight responses per prompt at temperature 1, with top-p and top-k truncation disabled. All sampled responses contribute to a single optimizer update; the optimizer mini-batch equals the full rollout batch. The prompt order is shuffled, and the final incomplete prompt batch in each epoch is retained. Training uses only fresh responses, without replay or demonstrations. Within each MBPP model size, all runs train from the common initial model for all five epochs.

The optimizer is AdamW with a constant learning rate, no warmup, moment coefficients $(0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay 0.01, and gradient clipping at norm 1. Table 7 lists the learning-rate settings for each dataset, initial model, and method.

The validation-based learning-rate selection protocol specifies a random problem-level split of each eligible training set into 80% training and 20% validation, using seed 42 and rounding the validation count up. This gives training/validation counts of 299/75 for MBPP, 1,918/480 for APPS Introductory, and 17,009/4,253 for AceCode. All tests and responses associated with a problem remain in the same partition, and the partition is shared across methods, model sizes, and learning-rate candidates. The official test sets and external evaluation benchmarks remain separate.

Table 6: Training hyperparameters for coding tasks. Rollout and update mini-batch sizes count prompts in a full batch; the final incomplete prompt batch is retained. Learning-rate settings are listed in Table 7. Token limits and sampling settings apply to training rollouts; evaluation context limits are specified in the text.

Setting	MBPP	APPS Introductory	AceCode
Initial model	Qwen3-0.6B / Qwen3-4B	Qwen2.5-Coder-1.5B-Instruct	
Optimizer	AdamW	AdamW	AdamW
Learning rate	See Table 7		
Adam coefficients (β_1, β_2)	$(0.9, 0.999)$	$(0.9, 0.999)$	$(0.9, 0.999)$
Weight decay	0.01	0.01	0.01
Learning-rate schedule	Constant	Constant	Constant
Warmup	None	None	None
Gradient clipping norm	1.0	1.0	1.0
Prompts per rollout batch	32	64	256
Responses per prompt	8	8	8
Update mini-batch size	32	64	256
Update epochs per rollout batch	1	1	1
Training epochs	5	2	1
Maximum prompt length (tokens)	1,024	16,384	16,384
Maximum response length (tokens)	2,048	2,048	2,048
Sampling temperature	1.0	1.0	1.0
Top-p	1.0	1.0	1.0
Top-k	Disabled	Disabled	Disabled

Table 7: Learning-rate settings for the coding experiments. A dash indicates that the method is not evaluated in that setting.

	MBPP		APPS Introductory	AceCode
Method	Qwen3-0.6B	Qwen3-4B	Qwen2.5-Coder-1.5B-Instruct	
GRPO–All	2×10^{-6}	1×10^{-6}	2×10^{-6}	2×10^{-6}
GRPO–Rate	2×10^{-6}	1×10^{-6}	2×10^{-6}	2×10^{-6}
REINFORCE++–All	2×10^{-6}	2×10^{-6}	2×10^{-6}	–
REINFORCE++–Rate	2×10^{-6}	2×10^{-6}	2×10^{-6}	–
GDPO	10^{-6}	2×10^{-6}	2×10^{-6}	–
GraphDPO	2×10^{-6}	5×10^{-6}	2×10^{-6}	–
DPO–Pareto	2×10^{-6}	1×10^{-6}	2×10^{-6}	–
POLPO	5×10^{-6}	5×10^{-6}	2×10^{-6}	2×10^{-6}

The protocol compares learning rates $\{1 \times 10^{-6}, 2 \times 10^{-6}, 5 \times 10^{-6}, 10^{-5}\}$, training each candidate on the training portion for the dataset-specific number of epochs in Table 6. For each method and initial model, selection uses the highest validation All-pass score at the final checkpoint, with one greedy response per problem. Ties are resolved by higher Avg. pass and then by the smaller learning rate. Training then restarts from the initial model with the selected rate on the full training set, including the validation portion, for the same number of epochs before final-checkpoint evaluation.

GRPO–All and GRPO–Rate use clipping parameter 0.2, standardize rewards within each response group, and average token contributions within each response before averaging over responses. Neither baseline uses a KL or entropy penalty.

POLPO uses $\beta = 10^{-3}$, the fixed initial model as reference, and sequence-summed token log-probability ratios. Its loss averages over all prompt groups and divides each group's negative log-likelihood by its eight candidates. Groups without a dominance comparison contribute zero loss and remain in the denominator. This candidate-count normalization does not normalize scores by response length. The verifier outcomes used to construct the Pareto relation are the same outcomes used for the two GRPO rewards. Micro-batches preserve each complete eight-response group.

D.2.3 Additional Baseline Implementations

The following baselines are evaluated on MBPP and APPS; AceCode compares POLPO with GRPO–All and GRPO–Rate. They use the same prompts, tests, rollout budgets, and training horizons as the corresponding GRPO and POLPO runs.

REINFORCE++. We use the original global-normalization variant of REINFORCE++ (Hu et al., 2025), without a within-prompt mean-reward baseline. The All and Rate variants assign the all-pass indicator and the fraction of tests passed, respectively, as terminal rewards. With discount factor 1, the terminal reward propagates to all response tokens; returns are then standardized across valid response tokens in the rollout batch. The policy loss uses the same clipping parameter 0.2 and sequence-mean token-mean reduction as GRPO, without a KL or entropy penalty.

GDPO. We adapt GDPO (Liu et al., 2026c) to the problem-specific test sets by treating each binary test outcome as a reward component. Each component is standardized across the eight responses to the same prompt using the sample standard deviation plus 10^{-6}. We average the resulting components over that problem's actual tests, broadcast the aggregate to its response tokens, and standardize across valid response tokens in the batch. A test with a constant outcome within a response group contributes zero. Test identities are aligned only within a prompt; absent tests are not padded into the average. Averaging rather than summing components avoids scaling the aggregate simply by the number of tests. The policy-loss configuration is otherwise the same as for REINFORCE++.

DPO–Pareto and GraphDPO. Both baselines use the same strict Pareto relation, fixed initial reference, sequence-summed log-probability ratios, and $\beta = 10^{-3}$ as POLPO. DPO–Pareto averages the negative log-sigmoid DPO loss over all strict dominance pairs within each prompt, including transitive pairs. Ties and incomparable responses introduce no preference pairs. For GraphDPO, we apply the local objective of Liu et al. (2026b) to this Pareto dominance graph. Each response that dominates at least one other response is a positive candidate in a softmax over itself and all responses it dominates, including transitive descendants. We average these negative log-choice probabilities over such responses within the prompt. This applies GraphDPO's objective to our verifier-derived graph; no reference solutions or ground-truth anchoring are added. For both methods, prompt losses are averaged over all prompts, with comparison-free groups contributing zero and remaining in the denominator. Their within-prompt pair or node averages receive no additional division by the number of candidates, whereas POLPO divides each group's negative log-likelihood by eight.

D.2.4 Evaluation

All coding evaluations generate one greedy response per problem and score the final checkpoint at the fixed training horizon. The initial-model row uses the same evaluation protocol as the trained models in its table. The response limit is 2,048 tokens for all three coding settings. For both Qwen3 model sizes, this budget includes thinking tokens. Training on MBPP uses a 3,072-token context, while evaluation allows a 4,096-token context to accommodate longer test prompts without reducing the 2,048-token response budget. The APPS and AceCode evaluations use an 8,192-token context, accommodating their evaluation prompts and the 2,048-token response budget.

For MBPP and APPS, let Q be the number of evaluation problems, let K_j be the number of tests for problem j, and let c_{jk} indicate whether its generated response passes test k. We report

$$\text{All-pass} = \frac{100}{Q} \sum_{j=1}^{Q} \prod_{k=1}^{K_j} c_{jk}, \tag{37}$$

$$\text{Avg. pass} = \frac{100}{Q} \sum_{j=1}^{Q} \frac{1}{K_j} \sum_{k=1}^{K_j} c_{jk}. \tag{38}$$

Here $Q = 500$ for MBPP and $Q = 1{,}000$ for APPS Introductory. The second metric is a macro-average over problems, so problems with many tests do not dominate the score. Compilation errors, runtime errors, and timeouts count as failures for the affected checks.

AceCode-trained models are evaluated with EvalPlus version 0.3.1, using HumanEval+ v0.1.10 (164 problems) and MBPP+ v0.2.0 (378 problems) (Liu et al., 2023). We use EvalPlus prompt formatting,

code sanitization, and execution checking. Each generated response is evaluated against both the original tests and the augmented tests; success requires passing every test in the corresponding suite. Both columns therefore report pass@1, rather than a per-test pass rate. The original-test MBPP score here is computed on the 378 EvalPlus problems and should not be equated with the official 500-problem MBPP test score. Generated programs run in isolated execution environments with time limits.

D.3 Multi-Instruction Following

The IFM5 and MulDimIF experiments train separate models using the training data of the corresponding dataset and evaluate on its test data. MulDimIF results do not evaluate transfer of an IFM5-trained checkpoint. For IFM5 and MulDimIF, the initial-model row denotes Qwen3-4B-Instruct-2507 before the additional training in these experiments.[1]

D.3.1 IFM5 Data and Split

We use `allenai/IF_multi_constraints_upto5`, released with Pyatkin et al. (2025).[2] The dataset card identifies this as IF-RLVR training data with up to five constraints per prompt, drawn from 25 IFEval and 29 IFBench-Train constraint templates. We abbreviate this dataset as IFM5. Our test set is a holdout from this released training dataset, not the original IFBench benchmark test set.

All 95,373 rows are uniformly shuffled, the first 2,000 rows are assigned to the test split, and the remaining 93,373 rows to the training split. This is a row-level random split without stratification by the number of constraints. Table 8 gives the test distribution before and after length filtering. The initial distribution is close to the reported approximate training proportions of 24%, 25%, 24%, 19%, and 7% for $K = 1, \ldots, 5$, respectively; these training percentages are rounded summaries, not exact counts.

Table 8: IFM5 test counts before and after excluding prompts longer than 2,048 tokens. The retained counts are the denominators for the reported complete satisfaction rates and the weights for overall aggregation.

Constraint count K	1	2	3	4	5	Total
Before filtering	478	493	500	394	135	2,000
Excluded	2	5	7	3	0	17
Retained	476	488	493	391	135	1,983

Prompts exceeding 2,048 tokens are excluded as entire examples, not shortened. This excludes 17 test prompts, leaving 1,983 for evaluation.

D.3.2 MulDimIF Data and Levels

Models in the MulDimIF experiment are trained on its training split and evaluated on its test split (Ye et al., 2026). The paper defines four levels using both the number of constraint categories and the number of individual constraints (Table 9). Thus, Level IV does not mean exactly four individual constraints. The framework also distinguishes example, listing, and incorporation presentation patterns.

D.3.3 Verification and Evaluation Metrics

For evaluated prompt x_j, let K_j be its number of constraints and $c_{jk} \in \{0, 1\}$ indicate whether the generated response passes check k. Complete constraint satisfaction is

$$u_j = \prod_{k=1}^{K_j} c_{jk}, \qquad \text{CS} = \frac{1}{n} \sum_{j=1}^{n} u_j, \tag{39}$$

[1] `https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507`

[2] `https://huggingface.co/datasets/allenai/IF_multi_constraints_upto5`

Table 9: MulDimIF difficulty levels, following Ye et al. (2026, Section 3.1.3).

Level	Constraint categories	Individual constraints
I	1	1–2
II	2	2–4
III	3	3–6
IV	4	4–8

Table 10: Training hyperparameters for IFM5 and MulDimIF. Rollout batch size counts prompts. The training horizon is measured in trainer steps; the update mini-batch size is reported as configured. Sampling settings apply to training rollouts.

Setting	IFM5	MulDimIF
Initial model	Qwen3-4B-Instruct-2507	
Optimizer	AdamW	AdamW
Learning rate	10^{-6}	10^{-6}
Adam coefficients (β_1, β_2)	$(0.9, 0.999)$	$(0.9, 0.999)$
Weight decay	0.01	0.01
Learning-rate schedule	Constant	Constant
Warmup	None	None
Gradient clipping norm	1.0	1.0
Prompts per rollout batch	64	64
Responses per prompt	16	16
Update mini-batch size	16	16
Update epochs per rollout batch	1	1
Training horizon (trainer steps)	400	400
Maximum prompt length (tokens)	2,048	4,096
Maximum response length (tokens)	2,048	4,096
Sampling temperature	1.0	1.0
Top-p	1.0	1.0
Top-k	Disabled	Disabled

where n is the number of evaluated prompts after applicable exclusions. Tables report $100\,\mathrm{CS}$ as a percentage.

For IFM5, let n_k count retained prompts with $K_j = k$ and let A_k be their mean complete satisfaction. The overall rate is the prompt-count-weighted average

$$\mathrm{CS}_{\mathrm{IFM5}} = \frac{\sum_{k=1}^{5} n_k A_k}{\sum_{k=1}^{5} n_k}. \tag{40}$$

The weights are the retained counts in Table 8: $(n_1, \ldots, n_5) = (476, 488, 493, 391, 135)$. Overall values are retained from the evaluation summaries rather than recomputed from rounded subgroup rates.

IFM5 results are from one training run per reported configuration and use greedy decoding for test responses.

D.3.4 Training Configurations

Table 10 summarizes the training hyperparameters for IFM5 and MulDimIF. Both configurations use 64 prompts per rollout batch and 16 sampled responses per prompt, giving 1,024 responses per full rollout batch. The configured training horizon is 400 trainer steps for each dataset.

On both IFM5 and MulDimIF, we compare the initial model, GRPO–All, GRPO–Rate, GDPO (Liu et al., 2026c), and POLPO (Table 4). GRPO–All assigns reward one only when every constraint is satisfied, whereas GRPO–Rate uses the fraction of satisfied constraints, as defined in Eq. (4). Both GRPO–All and GRPO–Rate use a KL coefficient of 10^{-3}.

POLPO constructs comparisons from the individual pass/fail vectors using Eq. (5) and optimizes the partial-order likelihood in Eq. (10) with score scale $\beta = 10^{-3}$. The GRPO KL coefficient controls

the KL penalty, whereas POLPO's β scales the policy-to-reference scores; their equal numerical values do not imply equal regularization strength.

D.3.5 MATHIF: DATA, TRAINING, AND EVALUATION

Benchmark and task. MathIF combines mathematical questions with one, two, or three programmatically verifiable instruction constraints (Fu et al., 2026). We evaluate all 420 examples: GSM8K, MATH-500, Minerva, and OlympiadBench each contribute 30 examples at each constraint count, while AIME 2024/25 contributes 20, giving 140 examples at each count. The benchmark uses 15 constraint types covering lexical, length, format, language, and affix requirements. The prompt contains the mathematical question, a request for step-by-step reasoning and a final answer in `\boxed{}`, and the constraint descriptions, in a single user turn without a system prompt.

Training data and overlap filtering. We construct a separate training dataset by adding constraints to problems from the MATH training split (Hendrycks et al., 2021b), using `DigitalLearningGmbH/MATH-lighteval` as the source release. Reference answers are extracted from the final boxed expression in each supplied solution. We filter source problems against both the 420 MathIF examples and MATH-500 using normalized exact matching, or a shared word 10-gram together with a `difflib` similarity ratio of at least 0.9. Problems sharing a template but differing only in numerical values are retained; this filtering therefore does not establish the absence of all semantic overlap. Of the 7,500 source problems, 55 are removed by overlap filtering, two because their question text exceeds 3,000 characters, and two because their solutions contain no boxed answer that can be extracted. These exclusions leave 7,441 source problems, split into 6,941 for training and 500 for validation. Each problem yields one example at each of $K = 1, 2, 3$ constraints, producing 20,823 training examples and 1,500 validation examples. An additional validation set contains the 500 MATH-500 problems without added constraints. No difficulty-based or all-pass filtering is applied.

Constraint generation. We use the constraint wording and sample constraint parameters from the MathIF implementation included in our codebase. Training uses 14 of the benchmark's 15 constraint types, excluding the requirement to answer in a specified language. Each training prompt follows the same format as an evaluation prompt: the mathematical question, a request for step-by-step reasoning, and the sampled constraints.

We check which constraints can be combined before sampling them together. If MathIF lists two constraint types as incompatible in either order, we treat the pair as incompatible in both orders. For the constraint that requires repeating the prompt, we adjust this list to permit the relevant benchmark combinations: its incompatible types are quotation, all-uppercase, all-lowercase, multiple-section, and response-language requirements. We also exclude combinations of all-lowercase text with multiple sections, and combinations of multiple sections with a bullet list. When all-uppercase text and multiple sections are required together, we use `SECTION` as the section delimiter.

We also check the question and reference answer when their contents could conflict with a constraint. A no-comma constraint is sampled only if the reference answer contains no comma; if the prompt must be repeated, the question must also contain no comma. Before sampling combinations that require repeating the prompt, we check the question for forbidden words and ensure that repeating it would not already exceed the specified word-frequency or word-count limits. Keywords are distinct within each set of constraints.

Verification and training objectives. For a response to a problem with K instruction constraints, the feedback vector is

$$\mathbf{v} = (a, c_1, \ldots, c_K) \in \{0,1\}^{K+1}, \qquad r_{\mathrm{rate}} = \frac{a + \sum_{k=1}^{K} c_k}{K+1}, \tag{41}$$

where a indicates mathematical correctness and c_k indicates satisfaction of constraint k. GRPO–All uses $r_{\mathrm{all}} = a \prod_{k=1}^{K} c_k$, which is one only when the response is mathematically correct and satisfies every instruction constraint. GRPO–Rate uses r_{rate} as its scalar reward, while POLPO constructs the dominance relation from the full vector, including correctness. We use the same verifier implementations during training and evaluation for both mathematical correctness and instruction-constraint

Table 11: Training hyperparameters for MathIF. Rollout and update mini-batch sizes count prompts. The training horizon is measured in trainer steps, each comprising one rollout batch; sampling settings apply to training rollouts.

Setting	MathIF
Initial model	Qwen3-0.6B
Optimizer	AdamW
Learning rate	10^{-6}
Adam coefficients (β_1, β_2)	$(0.9, 0.999)$
Weight decay	0.01
Learning-rate schedule	Constant
Warmup	None
Gradient clipping norm	1.0
Prompts per rollout batch	64
Responses per prompt	8
Update mini-batch size	16
Update epochs per rollout batch	1
Training epochs	1
Maximum prompt length (tokens)	2,048
Maximum response length (tokens)	2,048
Sampling temperature	1.0
Top-p	1.0
Top-k	Disabled

satisfaction. Mathematical correctness is evaluated with the mathematical-answer grader distributed with `verl` and used by MathIF.

Optimization. Table 11 summarizes the MathIF training hyperparameters in the same format as Table 10. All MathIF training runs initialize `Qwen/Qwen3-0.6B` with thinking disabled. Each rollout batch contains 64 prompts with eight responses per prompt, giving 512 sampled responses. The evaluated checkpoints correspond to the reported one-epoch runs of 325 trainer steps. Both GRPO–All and GRPO–Rate use a KL coefficient of 10^{-3}; POLPO uses score scale $\beta = 10^{-3}$ and no additional KL penalty.

Evaluation and metrics. All main comparisons use one greedy response per MathIF example, thinking disabled, and a maximum generation length of 16,384 tokens. Benchmark source and constraint count are identified from the dataset filenames because some record identifiers do not match those attributes. For the $Q = 420$ evaluated examples, let a_j be mathematical correctness, K_j the number of instruction constraints, and c_{jk} the official checker's outcome for constraint k. We report

$$\mathrm{HAcc} = \frac{100}{Q}\sum_{j=1}^{Q}\prod_{k=1}^{K_j} c_{jk}, \qquad \mathrm{SAcc} = \frac{100}{Q}\sum_{j=1}^{Q}\frac{1}{K_j}\sum_{k=1}^{K_j} c_{jk}, \tag{42}$$

$$\mathrm{Acc} = \frac{100}{Q}\sum_{j=1}^{Q} a_j, \qquad \mathrm{Joint} = \frac{100}{Q}\sum_{j=1}^{Q} a_j \prod_{k=1}^{K_j} c_{jk}. \tag{43}$$

HAcc and SAcc measure instruction adherence without requiring mathematical correctness; Joint measures their conjunction with correctness. All displayed percentages retain the precision of the evaluation summaries.

E Additional Experiments

E.1 Measured Toy Learning Efficiency

Table 12 summarizes the threshold crossings in Figure 3. The values are computed from the saved seed-42 trajectories. The reduction in generated programs is $1 - n_{\mathrm{POLPO}}/n_{\mathrm{GRPO\text{-}Rate}}$, where n is the first recorded budget at which complete-success probability reaches 0.9.

Table 12: Generated programs needed to reach 0.9 complete-success probability in the measured seed-42 runs, with $M = 60$. A dash means that the threshold was not reached within 512,000 programs. Crossings are taken from every-update records without interpolation; the reduction is relative to GRPO–Rate.

Target	GRPO–All	GRPO–Rate	POLPO	Reduction
$(3, 5)$	—	118,272	**95,744**	19.0%
$(3, 7)$	—	196,096	**107,520**	45.2%
$(3, 11)$	—	219,648	**98,816**	55.0%

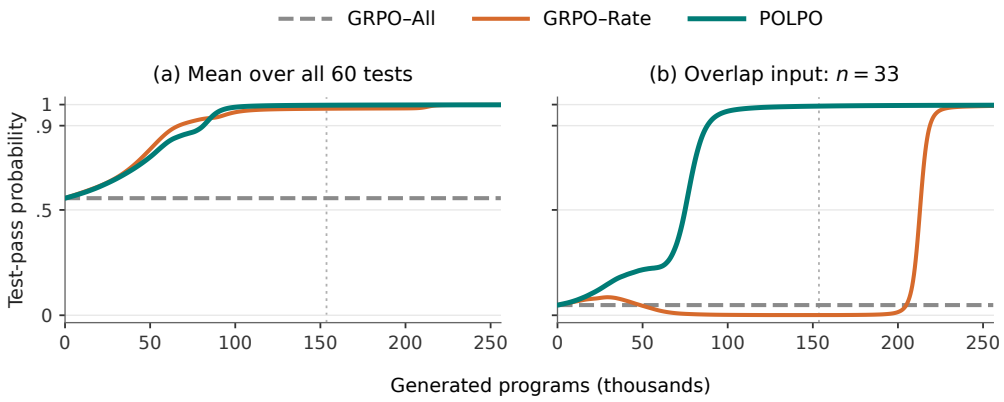


Figure 4: Diagnostic curves for target $(3, 11)$ in the seed-42 runs. The left panel averages pass probability over all 60 verifiers, and the right panel shows the overlap input $n = 33$. The vertical dotted line marks 153,600 generated programs.

E.2 Additional MathIF Results

POLPO achieves the highest overall Joint score on MathIF: 21.4%, compared with 19.3% for GRPO–Rate and 19.8% for GRPO–All (Table 5). The following breakdowns show where these gains arise.

Satisfying multiple constraints while answering correctly. Table 13 shows that POLPO achieves the highest complete constraint satisfaction (HAcc) at every constraint count. For two and three constraints, its Joint scores are 17.1% and 17.9%, compared with 13.6% and 14.3% for GRPO–Rate. With three constraints, both methods have the same mathematical accuracy at the reported precision (27.1%), but POLPO has higher Joint success. Thus, more of POLPO's mathematically correct responses also satisfy every instruction constraint. POLPO also exceeds GRPO–All in Joint success at every constraint count, despite GRPO–All having higher mathematical accuracy. These results show an advantage in meeting the combined requirements of correctness and instruction adherence. The advantage over GRPO–Rate is concentrated in the multiple-constraint settings: with one constraint, POLPO's Joint score is slightly lower (29.3% versus 30.0%).

Results across benchmark sources. Table 14 shows that POLPO exceeds GRPO–Rate in pooled Joint success on all five benchmark sources and achieves or ties the highest score on four of them. On MATH-500, POLPO reaches 37.8%, compared with 35.6% for GRPO–Rate and 30.0% for GRPO–All; on Minerva, it reaches 8.9%, compared with 5.6% and 7.8%, respectively. On OlympiadBench, POLPO matches GRPO–All at 13.3% and exceeds GRPO–Rate (11.1%). On AIME, only POLPO obtains a joint success, although this amounts to just one of the 60 examples. GSM8K is the exception to POLPO's leading pooled score: its 38.9% exceeds GRPO–Rate (37.8%) but remains below GRPO–All (41.1%). The improvements over GRPO–Rate therefore occur across sources, while the comparison with GRPO–All depends on the source.

Table 13: MathIF results (%) by constraint count, with 140 examples per count. Within each metric, columns are the initial model (Init.), GRPO–All (All), GRPO–Rate (Rate), and POLPO (POLPO). All models use non-thinking mode.

	HAcc				Acc				Joint			
K	Init.	All	Rate	POLPO	Init.	All	Rate	POLPO	Init.	All	Rate	POLPO
1	56.4	73.6	80.7	84.3	15.0	31.4	33.6	30.7	10.7	27.9	30.0	29.3
2	22.1	47.9	65.7	72.9	10.0	23.6	20.7	22.9	2.9	15.7	13.6	17.1
3	14.3	36.4	67.1	72.9	16.4	33.6	27.1	27.1	3.6	15.7	14.3	17.9

Table 14: MathIF joint success (%) by source and constraint count. Joint success requires mathematical correctness and satisfaction of every instruction constraint. All models use non-thinking mode. Rows marked All pool the three constraint counts for that source.

Source	K	n	Initial	GRPO–All	GRPO–Rate	POLPO
GSM8K	1	30	26.7	56.7	60.0	63.3
GSM8K	2	30	10.0	33.3	23.3	23.3
GSM8K	3	30	6.7	33.3	30.0	30.0
GSM8K	All	90	14.4	41.1	37.8	38.9
MATH-500	1	30	16.7	43.3	60.0	53.3
MATH-500	2	30	3.3	30.0	26.7	33.3
MATH-500	3	30	3.3	16.7	20.0	26.7
MATH-500	All	90	7.8	30.0	35.6	37.8
Minerva	1	30	6.7	13.3	6.7	6.7
Minerva	2	30	0.0	3.3	6.7	13.3
Minerva	3	30	3.3	6.7	3.3	6.7
Minerva	All	90	3.3	7.8	5.6	8.9
OlympiadBench	1	30	0.0	16.7	13.3	13.3
OlympiadBench	2	30	0.0	6.7	6.7	10.0
OlympiadBench	3	30	3.3	16.7	13.3	16.7
OlympiadBench	All	90	1.1	13.3	11.1	13.3
AIME	1	20	0.0	0.0	0.0	0.0
AIME	2	20	0.0	0.0	0.0	0.0
AIME	3	20	0.0	0.0	0.0	5.0
AIME	All	60	0.0	0.0	0.0	1.7

F Extended Related Work

This appendix complements Section 5 with related optimization and evaluation methods beyond the direct comparison between scalar aggregation and partial-order likelihoods in multi-verifier RLVR.

F.1 Additional Approaches to Multiple Verification Signals

IFEval provides automatically checkable instruction constraints that underpin later work on verifiable instruction following (Zhou et al., 2023). VerIF combines code-based checks for hard constraints with language-model judgments for soft constraints, then aggregates their outcomes for GRPO training (Peng et al., 2025); its feedback therefore includes learned judgments as well as executable checks. MDP-GRPO uses the mean constraint-satisfaction reward and modifies sampling and advantage estimation to address difficult reward distributions in multi-constraint instruction following (Salmani-Zarchi et al., 2026).

Beyond the aggregation methods discussed in the main text, GD^{2}PO addresses conflicts between component advantages through rollout masking and prompt reweighting (Liu et al., 2026a). Blockwise advantage estimation applies objective-specific advantages to the corresponding output segments (Pavlenko et al., 2026). These approaches show that using multiple rewards need not mean assigning one undifferentiated signal to every token. POLPO addresses a different design choice: it

uses comparisons between complete responses, without assuming a correspondence between verification criteria and output segments.

F.2 Policy Optimization and Listwise RLVR

Reinforcement learning from human feedback trains policies using learned preference rewards (Ouyang et al., 2022), while RLVR obtains feedback from automatic verification (Lambert et al., 2025; Guo et al., 2025). GRPO constructs advantages from rewards within a response group (Shao et al., 2024) and uses the clipped policy surrogate of PPO (Schulman et al., 2017). See Appendix A.2 for the definitions needed for the gradient comparison in this paper.

Listwise Policy Optimization interprets group-based RLVR as projection onto target distributions over sampled responses (Qu et al., 2026). Its targets are constructed from scalar rewards, and its update coefficients can depend on current policy scores and sum to zero within the group. These properties are therefore not unique to POLPO. The distinction is the source of the coefficients: POLPO differentiates the probability of a verifier-induced partial-order event, marginalizing over compatible complete rankings rather than projecting onto a scalar-reward target distribution.

F.3 Listwise and Set-Level Preference Optimization

DPO uses policy-to-reference log-probability ratios in a Bradley–Terry comparison model and extends this construction to complete rankings with the Plackett–Luce model (Rafailov et al., 2023). The complete-ranking negative log-likelihood is the ListMLE loss (Xia et al., 2008); ListNet provides another listwise learning-to-rank formulation (Cao et al., 2007). PRO (Song et al., 2024) and LiPO (Liu et al., 2025) bring learning from ranked or scored response lists to language-model alignment. POLPO instead specifies a partial-order event and sums the probabilities of the complete rankings that satisfy it.

Related methods also learn from sets rather than individual pairs. In recommendation, S-DPO contrasts one positive item with several negatives, corresponding to the PL probability that the positive is selected first (Chen et al., 2024). Mult-DPO handles multiple positives, negatives, and preference levels with a multinomial surrogate that upper-bounds the corresponding marginalized PL loss (Zhu et al., 2026). Multi-Preference Optimization generalizes Bradley–Terry comparisons to chosen and rejected sets (Gupta et al., 2024). These set-based objectives address particular comparison structures. The PL likelihood for a general partial order, formed by summing over compatible complete rankings, is established in the ranking literature (Liu et al., 2019); ordered-partition likelihoods have also been studied explicitly (Ma et al., 2021). POLPO adopts this probabilistic construction for verifier-induced comparisons between model responses and optimizes it through DPO-style policy scores in an online sampling-and-update procedure.

F.4 Verifier-Guided Preferences and Prompt Optimization

Verifier outcomes can also generate pairwise preference data for reasoning. Iterative Reasoning Preference Optimization pairs correct and incorrect responses and adds a negative log-likelihood term on the correct response (Pang et al., 2024). DPO-VP constructs correct–incorrect pairs from binary verifiable rewards and trains iteratively (Tu et al., 2025). These methods connect verification to preference optimization, while POLPO uses the comparison structure of a response group and models its specified comparisons as one ranking event.

ParetoPrompt uses dominance relations between prompts for preference-based policy optimization (Zhao et al., 2025). This is related through its use of multiple criteria, but its optimization objects are prompts. POLPO constructs comparisons between responses to the same prompt and updates the response policy through their joint partial-order likelihood.