JACOBIAN POLICY OPTIMIZATION: GEOMETRY-AWARE LEARNING FROM STRUCTURED REWARDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning for language-model reasoning increasingly uses multiple reward signals to evaluate the consequences and properties of generated responses. Training a shared policy to respond to different priorities requires learning behaviors whose outcomes reflect the requested emphasis among these signals. We introduce **Jacobian Policy Optimization (JPO)**, a preference-conditioned reinforcement learning method that uses objective-specific gradient geometry to construct policy updates. JPO independently normalizes each reward dimension, constructs separate clipped policy losses, and differentiates them to form a Jacobian of policy gradients. Preference-weighted UPGrad modifies conflicting gradient components before aggregation, allowing both the requested preference and the relationships among learning signals to determine the update. We evaluate JPO in two complementary settings. On MaZE, where rewards describe completion, resource collection, and hazard avoidance, JPO achieves substantially higher matched-preference utility and stronger preference specialization than GRPO and GDPO on held-out mazes under the training preference set. On Level-5 MATH, where rewards assess correctness, conciseness, and rigour, JPO achieves the highest individual-sample accuracy and Pass@8 among the evaluated methods, while remaining competitive on Majority@8. It also achieves the highest Majority@8 after Recursive Self-Aggregation. These results support geometry-aware multi-reward policy optimization as an approach to improving both preference-conditioned behavior and reasoning performance under multiple quality criteria.

1 INTRODUCTION

Reinforcement learning for language-model reasoning increasingly draws on multiple sources of feedback. A mathematical solution may be evaluated for correctness, rigour, and conciseness; a generated plan may be evaluated for goal attainment, resource collection, and hazard avoidance. These signals describe the consequences and properties of a model's response. In practice, however, they are commonly combined into a scalar reward or advantage before the policy update is constructed. When a shared model must respond to different priorities, the learning problem extends beyond obtaining a high return under one fixed objective: the model must learn to adjust its behavior as the desired outcomes change.

Multi-objective reinforcement learning (MORL) provides a natural formulation through vector-valued rewards (Roijers et al., 2013; Hayes et al., 2022), while preference-conditioned policies allow a single model to respond to different priorities (Yang et al., 2025; Ambadkar et al., 2026). We adopt this formulation with a policy $\pi_\phi(y \mid x, \omega)$, conditioned on a problem x and a preference vector ω. The preference changes the requested emphasis, while the definitions of the reward coordinates remain fixed. This leaves an important optimization choice: *should the update depend only on a weighted combination of learning signals, or also on the relationships among their individual gradients?* We investigate whether using this gradient geometry improves the policy's ability to realize the requested outcomes.

Existing methods differ in how they process multiple reward signals before constructing the policy update. GRPO (Shao et al., 2024) computes group-relative advantages from a scalar reward, while GDPO (Liu et al., 2026) normalizes each reward dimension independently before combining the advantages. D^{3}PO (Ambadkar et al., 2026) retains objective-specific surrogate losses and ap-

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.

plies preference weights after clipping. Separately normalized rewards and objective-specific policy losses preserve distinctions at different stages of optimization. Retaining their gradients also makes the interactions among objectives available when constructing the update. A fixed weighted sum of the same objective losses would produce the same gradient whether applied before or after differentiation; changing the update therefore requires using information beyond that weighted sum. Gradient-based multi-task methods use gradient geometry to address interference between objectives (Yu et al., 2020; Liu et al., 2021). We build on Jacobian Descent (JD), which collects objective gradients as rows of a Jacobian and uses the UPGrad aggregator to modify conflicting gradient components before aggregation (Quinton & Rey, 2025).

We introduce **Jacobian Policy Optimization (JPO)**, which applies this geometry-aware update to preference-conditioned language-model reinforcement learning. JPO independently normalizes each reward dimension, constructs a separate clipped policy loss for each objective, and differentiates these losses to form a Jacobian of policy gradients. The same sampled responses contribute to every objective loss, allowing each update to use several assessments of the same behavior. Preference-weighted UPGrad, an aggregator from the Jacobian Descent framework, modifies conflicting gradient components and combines the gradients into an update direction. The resulting update depends on both the requested preference and the geometry of the objective-specific gradients.

Beyond the optimization problem, learning from multiple reward signals raises a broader question about what a reasoning policy acquires during training. In planning tasks, these signals can describe distinct consequences of a generated action sequence, allowing learning to distinguish behaviors that achieve the same final goal. This perspective connects to General Value Functions (GVFs), which represent knowledge through predictions of different signals under specified behaviors and continuation rules (White et al., 2012). The same consequence may matter under several objectives, even when its relative importance changes. Here, we investigate how feedback about multiple consequences can support the acquisition of preference-conditioned behavior. JPO learns these behaviors directly, without explicit consequence prediction; whether they provide capabilities that can be reused under new objectives remains an open empirical question.

We first evaluate whether JPO learns behaviors that respond to different requested outcomes. In MaZE, a navigation environment adapted from Vector Policy Optimization (Bahlous-Boldi et al., 2026), rewards describe task completion, gold collection, diamond collection, and lava avoidance. Each training maze is paired with four preferences, requiring a shared model to accommodate different priorities within the same environment. Evaluating each held-out maze under all four preferences tests whether changing the request changes the outcomes within the same environment. On held-out mazes, JPO learns substantially stronger preference-dependent behavior than GRPO and GDPO, achieving higher matched-preference utility and a larger preference margin. These results support improved preference-conditioned behavior on new mazes under the training preference set.

We then examine whether JPO is useful when multiple rewards assess the quality of a single reasoning solution. On Level-5 MATH, we train with rewards for correctness, conciseness, and rigour. JPO achieves the highest individual-sample accuracy and Pass@8 among the evaluated methods under direct sampling, and the highest Majority@8 after Recursive Self-Aggregation. Together, these results show that JPO, which preserves objective-specific learning signals through differentiation and combines the resulting gradients using geometry-aware aggregation, improves both the ability to map requested objective emphasis to realized policy outcomes and the quality of policies used for inference-time sampling and aggregation.

Our main contributions are:

1. **Jacobian Policy Optimization.** We introduce JPO, which combines objective-wise reward normalization and separately clipped policy losses with preference-weighted UPGrad aggregation of their gradients.
2. **Preference-conditioned behavior.** On held-out MaZE environments, JPO substantially improves matched-preference utility and preference specialization over GRPO and GDPO.
3. **Reasoning with multiple quality criteria.** On Level-5 MATH, JPO improves individual-sample accuracy and Pass@8 over GRPO and GDPO when trained with rewards for correctness, conciseness, and rigour.

2 Related Work

2.1 Preference-conditioned learning with vector-valued rewards

Multi-Objective Reinforcement Learning (MORL) represents an agent's feedback using vector-valued rewards and has traditionally studied how to optimize competing objectives, recover policies for different preferences, or approximate a Pareto frontier (Roijers et al., 2013; Hayes et al., 2022). A related line of work learns a single policy conditioned on a preference vector, so that changing the preference at inference time can induce different behaviors (Yang et al., 2025; Ambadkar et al., 2026). Our work adopts this preference-conditioned formulation but emphasizes a different question from classical Pareto-front recovery. We treat reward coordinates as task-relevant consequences or properties of behavior, and train a single language model $\pi_\phi(y \mid x, \omega)$ to realize different outcome profiles as the requested preference ω changes.

2.2 Multi-reward LLM post-training

Language-model post-training increasingly uses several reward signals to describe different aspects of a response. Standard approaches typically combine these signals before or during policy optimization. GRPO computes a scalar group-relative advantage from rollout rewards (Shao et al., 2024). GDPO shows that aggregating heterogeneous rewards before group normalization can erase distinctions between reward configurations, and instead normalizes each reward dimension independently before combining them (Liu et al., 2026). D^{3}PO preserves the decomposition further by constructing objective-specific PPO surrogate losses and applying preference weights only after clipping (Ambadkar et al., 2026). JPO continues this progression: objective-specific signals remain separate through differentiation, and are combined only after each objective has produced its own parameter gradient.

Other recent work uses vector-valued or multiple rewards for different purposes. VPO uses randomized scalarizations to maintain reward-diverse candidate solutions for downstream test-time search (Bahlous-Boldi et al., 2026), while ROSA models uncertainty over reward functions and derives policies that maintain diverse behavior under that uncertainty (GX-Chen et al., 2026). These approaches seek diversity within the policy's response distribution under a fixed or uncertain downstream objective. In contrast, JPO conditions the policy explicitly on a supplied preference and asks whether changing that preference produces correspondingly different policy outcomes. Nevertheless, these works provide complementary evidence that preserving structure across multiple reward signals can be useful during post-training.

2.3 Gradient-level multi-objective optimization

Combining multiple parameter gradients has been extensively studied in multi-task and multi-objective optimization. Methods such as PCGrad, CAGrad, Nash-MTL, and Aligned-MTL modify the magnitude or geometry of task gradients to reduce destructive interference or incorporate task preferences (Yu et al., 2020; Liu et al., 2021; Navon et al., 2022; Senushkin et al., 2023).

Jacobian Descent (JD) provides a general formulation for such methods by representing the gradients of a vector-valued objective as the rows of a Jacobian (Quinton & Rey, 2025). Its UPGrad aggregator resolves conflicting gradient components through projection before forming a single parameter update, and can incorporate explicit preference weights. JPO applies this gradient-level view to preference-conditioned language-model reinforcement learning. Instead of scalarizing rewards, advantages, or policy losses before differentiation, it constructs objective-specific policy gradients and allows both the requested preference and their gradient geometry to determine the final update.

3 Methodology

We consider preference-conditioned reinforcement learning with a vector-valued reward. Given an input x, the model is conditioned on a preference vector ω specifying the desired trade-off between task objectives. Previous methods (Shao et al., 2024; Liu et al., 2026; Ambadkar et al., 2026) scalarize these objectives at various stages in the training pipeline before the differentiation step of policy optimization. We preserve their individual learning signals through reward normalization, advantage

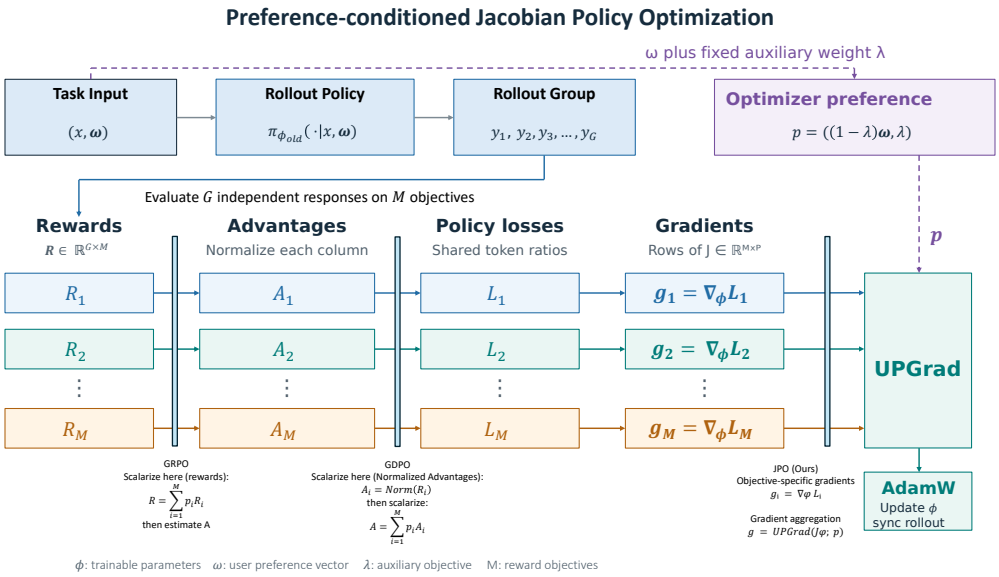


Figure 1: **Overview of preference-conditioned Jacobian policy optimization.** Given a task x and preference vector ω, the rollout policy generates G responses evaluated on M reward objectives. Each reward dimension is normalized independently to form objective-specific PPO losses $L_1, \ldots, L_M$, whose gradients form the Jacobian $J_\phi = [\nabla_\phi L_1^\top; \ldots; \nabla_\phi L_M^\top]$. Preference-weighted UPGrad aggregates these gradients using $p = ((1-\lambda)\omega, \lambda)$ before the AdamW update. GRPO and GDPO instead scalarize the multi-objective signal at earlier stages.

estimation, policy-loss construction, and differentiation. Only after obtaining the objective-specific parameter gradients do we combine them using preference-weighted UPGrad. *Figure* 1 gives an overview of the resulting training procedure.

3.1 Preference-Conditioned Multi-Objective Policy Optimization

We distinguish between K *task objectives*, whose trade-off is controlled by the user through a preference vector

$$\omega \in \Delta^{K-1},$$

and auxiliary objectives that encode system-level considerations specified by the developer. In this work, we use a single auxiliary objective ($L = 1$), so the total number of objectives is $M = K + 1$. The policy

$$\pi_\phi(y \mid x, \omega)$$

is conditioned on the input x and the user preference ω, but not on the auxiliary-objective weight.

For a generated response y, the environment returns

$$r(x, y) = \left(r_1^{\text{task}}(x, y), \ldots, r_K^{\text{task}}(x, y), r^{\text{aux}}(x, y)\right) \in \mathbb{R}^M.$$

Although the task and auxiliary objectives play different roles at deployment, they are treated identically as separate objectives during optimization. Let $\lambda \in [0, 1]$ denote the developer-specified weight assigned to the auxiliary objective. We define the optimizer preference as

$$p = \left((1-\lambda)\omega, \lambda\right) \in \Delta^{M-1}. \tag{1}$$

Thus, ω is the user-specified preference on which the policy is conditioned, whereas p combines the user preference with the developer-specified auxiliary weight and is used only when aggregating the objective-specific gradients.

Our central design choice is to avoid applying p directly to the rewards, advantages, or policy losses. Instead, we construct an objective-specific loss vector

$$\mathcal{L}(\phi) = \begin{bmatrix} L_1(\phi) & \cdots & L_M(\phi) \end{bmatrix}^\top,$$

and treat policy optimization itself as a multi-objective optimization problem. This preserves the objective decomposition until differentiation, in contrast to approaches that scalarize at the reward (Shao et al., 2024), advantage (Liu et al., 2026), or surrogate-loss (Ambadkar et al., 2026) stage.

3.2 Jacobian Descent and Preference-Weighted UPGrad

Let M be the total number of objectives, and P be the number of trainable parameters in ϕ. Given the objective-specific policy losses $\mathcal{L}(\phi) = [L_1(\phi), \dots, L_M(\phi)]^\top$, we differentiate each objective separately to form the loss Jacobian

$$J_\phi = \begin{bmatrix} \nabla_\phi L_1^\top \\ \vdots \\ \nabla_\phi L_M^\top \end{bmatrix} \in \mathbb{R}^{M \times P}. \tag{2}$$

Instead of scalarizing these gradients directly, we apply the UPGrad aggregator of Jacobian Descent (Quinton & Rey, 2025). UPGrad uses the geometry of the objective gradients to remove conflicting components and produces projected gradients $\widetilde{g}_1, \dots, \widetilde{g}_M$. We incorporate the optimizer preference p only at this final stage:

$$d_{\mathrm{JD}} = \sum_{i=1}^{M} p_i \widetilde{g}_i. \tag{3}$$

The resulting direction is passed to AdamW for the parameter update. Thus, preferences determine the relative contribution of the objectives, while UPGrad additionally accounts for interactions between their gradients. When the objective gradients do not conflict, this reduces to the ordinary preference-weighted gradient. Further details on Jacobian Descent and UPGrad are provided in Appendix A.

3.3 Training Pipeline

For each training input, we sample or select a preference vector ω and construct a preference-conditioned prompt. A frozen snapshot of the current policy, denoted $\pi_{\phi_{\mathrm{old}}}$, generates a group of G independent responses

$$y_1, \dots, y_G \sim \pi_{\phi_{\mathrm{old}}}(\cdot \mid x, \omega).$$

Every response is evaluated independently on all M reward dimensions, producing the reward matrix

$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,M} \\ \vdots & & \vdots \\ r_{G,1} & \cdots & r_{G,M} \end{bmatrix} \in \mathbb{R}^{G \times M},$$

where $r_{g,i}$ is the reward received by rollout g on objective i.

Objective-wise advantage estimation. Following the motivation of GDPO (Liu et al., 2026), we normalize each reward dimension separately across the rollout group. For objective i,

$$A_{g,i} = \frac{r_{g,i} - \mu_i}{\sigma_i + \epsilon}, \qquad \mu_i = \frac{1}{G} \sum_{g=1}^{G} r_{g,i},$$

where σ_i is the sample standard deviation of objective i within the group and ϵ is a small numerical constant (to avoid division by zero in the case of identical rewards). The resulting matrix $A \in \mathbb{R}^{G \times M}$ is retained. Unlike GDPO, we do not subsequently sum or preference-weight the normalized advantage dimensions.

Objective-specific policy losses. The current policy is evaluated once on the sampled responses. For rollout g and response token t, we compute the policy ratio

$$\rho_{g,t}(\phi) = \frac{\pi_\phi(y_{g,t} \mid x, \omega, y_{g,<t})}{\pi_{\phi_{\mathrm{old}}}(y_{g,t} \mid x, \omega, y_{g,<t})}.$$

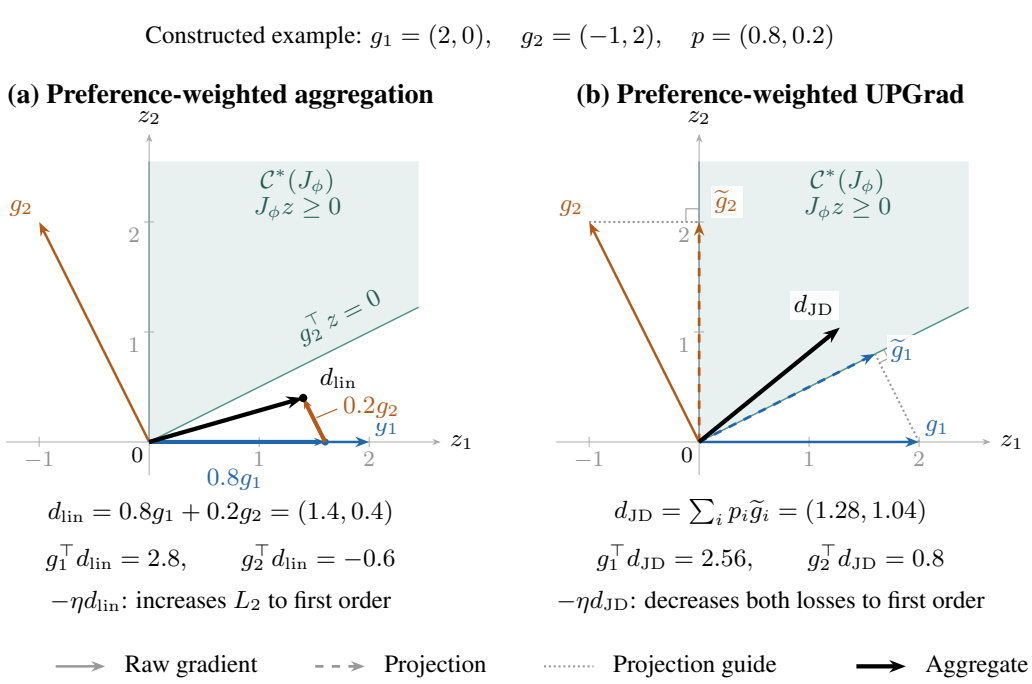


Figure 2: **Geometry of preference-weighted UPGrad.** A constructed two-dimensional illustration, not measured training data, uses identical gradients, weights, coordinates, and scales in both panels. Shading denotes the common cone $\mathcal{C}^*(J_\phi) = \{z : J_\phi z \geq 0\}$. (a) The weighted aggregate lies outside this cone and conflicts with g_2. (b) Euclidean projections $\widetilde{g}_1 = (1.6, 0.8)$ and $\widetilde{g}_2 = (0, 2)$ yield an aggregate inside the cone. Arrows show gradients d, while the first-order loss changes refer to the descent step $-\eta d$, with $\eta > 0$.

The same token-level ratios are reused for every objective. For each objective i, we then construct an independent clipped policy loss

$$L_i(\phi) = -\frac{1}{G}\sum_{g=1}^{G}\frac{1}{T_g}\sum_{t=1}^{T_g}\min\left(\rho_{g,t}A_{g,i}, \text{clip}(\rho_{g,t}, 1-\varepsilon, 1+\varepsilon)A_{g,i}\right).$$

This gives the loss vector $\mathcal{L} = (L_1, \ldots, L_M)^\top$ while requiring only one model forward pass. The objective advantages and old-policy log probabilities are treated as constants during differentiation.

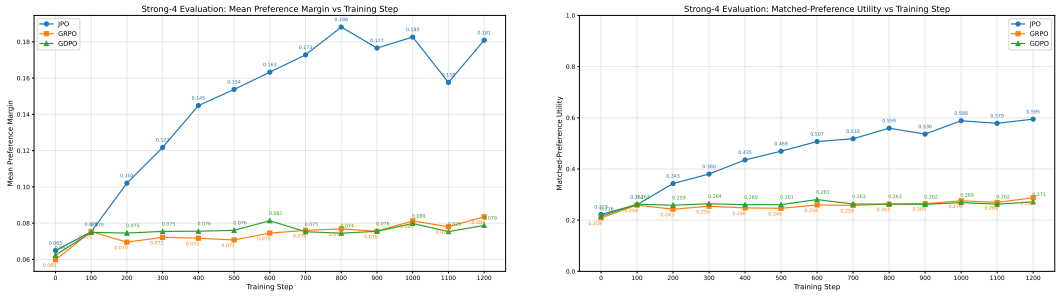
Gradient aggregation and model update. We differentiate the complete loss vector to obtain J_ϕ in Eq. 2. The user preference ω is converted to the optimizer preference p when required, for example using Eq. 1 when a fixed auxiliary objective is present. Preference-weighted UPGrad then maps the M objective gradients to the single direction d_{JD}. We place this direction in the model's gradient buffers and perform the parameter update using AdamW:

$$\phi \leftarrow \text{AdamW}(\phi, d_{\text{JD}}).$$

The updated parameters are subsequently synchronized with the rollout policy and the procedure repeats.

4 Experiments

We evaluate JPO in two complementary multi-reward settings that probe different aspects of the method. MaZE provides a controlled test of *preference-conditioned outcome control*: each reward coordinate describes an interpretable consequence of an agent's trajectory, and we ask whether changing the requested preference causes a single trained model to realize a correspondingly differ-



(a) Mean preference margin over training.

(b) Matched-preference utility over training.

Figure 3: **Preference-conditioned outcome control on MaZE during training.** (a) Mean preference margin and (b) matched-preference utility for JPO, GRPO, and GDPO. Together, these metrics measure how strongly the final trajectory reward profiles reflect the conditioning preference. Higher values indicate stronger preference responsiveness.

Table 1: Preference-conditioned evaluation on 50 held-out MaZE instances. Each entry averages eight stochastic rollouts per maze. The first four columns report matched-preference normalized utility. U_{match} is their mean and Δ_{pref} is the mean preference margin. Higher values indicate stronger alignment between the conditioning preference and the realized trajectory outcomes.

Method	Completion	Gold	Diamond	Lava	U_{match}	Δ_{pref}
GRPO	0.298	0.221	0.285	0.346	0.288	0.084
GDPO	0.264	0.247	0.285	0.289	0.271	0.079
JPO (ours)	**0.716**	**0.487**	**0.482**	**0.694**	**0.595**	**0.181**

ent outcome profile. MATH provides a complementary setting in which the reward dimensions describe related properties of a single reasoning solution. Here, we ask whether preserving objective-specific learning signals remains useful even when the rewards are not constructed to induce a sharp conflict between objectives.

4.1 Preference-Conditioned Outcome Control on MaZE

We first evaluate JPO on MaZE, a controlled navigation environment adapted from VPO (Bahlous-Boldi et al., 2026). Each 9×9 maze admits multiple feasible routes and provides four reward coordinates describing interpretable consequences of the generated trajectory: task completion, gold collection, diamond collection, and lava avoidance. We train Qwen3.5-4B (Qwen Team, 2026) with four strongly concentrated preferences, placing 85% of the preference weight on one objective and 5% on each of the others. The training set is such that each unique maze is paired with all four preferences. JPO is compared with GRPO and GDPO under the same model, data, and LoRA configuration. Details are provided in Appendix B.

For evaluation, we use 50 held-out mazes and generate eight responses for each maze–preference pair. We report matched-preference utility, which measures performance when a response is evaluated under the same preference that conditioned its generation, and preference margin, which measures how strongly the resulting reward profile is oriented toward that preference relative to the other three. Higher values therefore indicate stronger preference-conditioned outcome control. Detailed definitions are provided in Appendix B.7.

JPO exhibits substantially stronger preference-dependent behavior than GRPO and GDPO. As shown in Table 1, JPO achieves higher matched-preference utility across all four requested objective emphases and a substantially larger preference margin. Figure 3 shows that this separation develops over training, while the two scalarized baselines remain comparatively insensitive to the conditioning preference. Thus, the main effect is not simply higher aggregate reward, as changing the preference supplied to the same JPO-trained model produces final outcome profiles that more closely reflect the requested objective emphasis. This indicates that preserving objective-specific

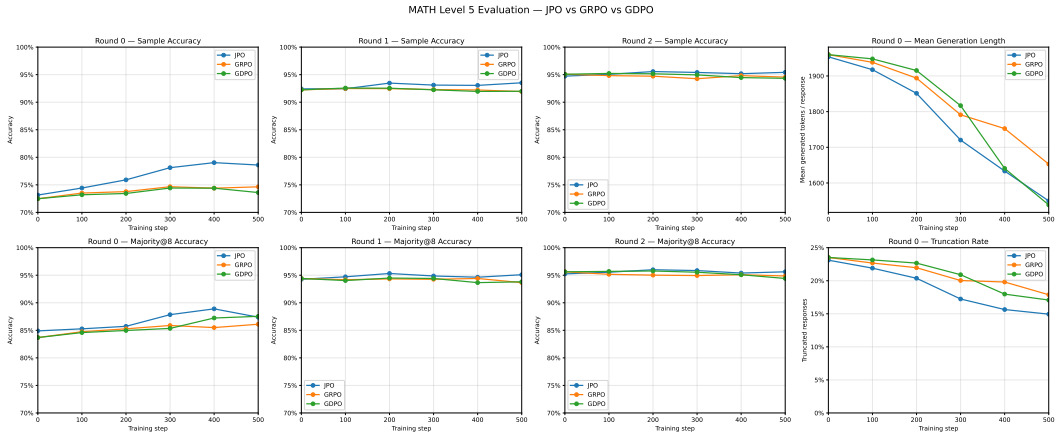


Figure 4: MATH evaluation performance throughout training in the related-reward setting. We report direct-sampling performance together with Majority@8 after Recursive Self-Aggregation. Each point corresponds to an evaluated training checkpoint for GRPO, GDPO, and JPO. The final-checkpoint results are reported separately in Table 2.

learning signals through differentiation improves the model's ability to learn a family of preference-conditioned policies.

4.2 Mathematical Reasoning with Related Reward Signals

We next evaluate JPO on competition-level mathematical reasoning. Unlike MaZE, where reward coordinates describe distinct trajectory outcomes, MATH uses related aspects of solution quality: correctness, conciseness, and rigour. A strong solution can perform well on all three simultaneously, so this setting tests whether preserving objective-specific learning signals remains useful even when the rewards are overlapping rather than deliberately conflicting.

We train Qwen3.5-4B on Level-5 problems from the MATH benchmark (Hendrycks et al., 2021) using four fixed preference vectors,

$$(0.80, 0.10, 0.10), \qquad (0.60, 0.20, 0.20), \qquad (0.55, 0.10, 0.35), \qquad (0.55, 0.35, 0.10),$$

where the coordinates correspond to correctness, conciseness, and rigour. These provide correctness-dominated, balanced, rigour-emphasized, and conciseness-emphasized conditions. Formatting is included as an additional fixed auxiliary reward. Details are in Appendix C.

Reasoning-quality rewards. Conciseness and rigour are evaluated by a frozen GPT-OSS-120B judge (OpenAI, 2025) operating in LOW reasoning mode, which assigns independent scores in $\{0, 1, 2, 3, 4\}$. Conciseness measures whether the reasoning is focused and free from unnecessary repetition without penalizing necessary detail, while rigour measures whether non-trivial steps and logical implications are adequately justified. Both are evaluated independently of final-answer correctness and formatting. The complete judge prompt and scoring rubric are in Appendix C.

Evaluation. We evaluate on the 1,324 held-out Level-5 problems, excluding Levels 1–4 because the base model already performs strongly on them. We sample $K = 8$ solutions and report Sample Acc., Pass@8, and Maj.@8. Using the same candidates, we additionally run two rounds of Recursive Self-Aggregation (RSA) (Venkatraman et al., 2026), with $R = 0$ corresponding to direct sampling.

Table 2 shows that JPO achieves the strongest direct sampling performance overall, with the highest sample accuracy and Pass@8, while remaining comparable to GDPO on Majority@8. JPO also performs best after both rounds of RSA, indicating that the learned policy provides a stronger basis for recursive inference-time aggregation. These gains are not driven by longer generations: JPO produces responses of similar length to GDPO while exhibiting the lowest truncation rate. Figure 4 shows the corresponding trends throughout training.

Table 2: MATH evaluation on 1,324 held-out Level-5 problems ($K = 8$). RSA-1/2 report Maj.@8 after one/two RSA rounds. Trunc. is the fraction of direct samples reaching 4096 tokens; Avg. Len. is mean response length. Base Model averages three independent JPO/GRPO/GDPO evaluations. Accuracy and truncation are percentages; best results are bold.

Method	Sample Acc.	Pass@8	Maj.@8	RSA-1 Maj.@8	RSA-2 Maj.@8	Trunc.	Avg. Len.
Base Model	72.70	88.85	84.09	94.36	95.49	23.38	1957.7
GRPO	74.64	91.31	86.10	93.66	94.86	17.88	1652.9
GDPO	73.60	91.31	**87.54**	93.81	94.41	17.08	**1538.6**
JPO (ours)	**78.60**	**92.30**	87.39	**95.09**	**95.62**	**14.95**	1549.9

Unlike MaZE, these results are not primarily about learning different preference-dependent outcome profiles. Correctness, conciseness, and rigour are related signals describing the quality of the same solution. The MATH results therefore show that delaying scalarization remains useful even in this regime: preserving the reward dimensions separately through differentiation improves both direct sampling and downstream aggregation. Together with MaZE, this suggests that JPO is useful both for preference-conditioned outcome control and for learning from structured, multi-dimensional reward signals that describe related properties of a task.

5 Discussion and Conclusion

JPO approaches multi-reward policy optimization as learning a family of preference-conditioned policies instead of recovering a Pareto frontier. On MaZE, the reward dimensions describe interpretable consequences of a trajectory, and JPO more effectively learns the mapping from requested preference to realized outcome than GRPO and GDPO. On MATH, correctness, conciseness, and rigour are related properties rather than deliberately conflicting objectives, suggesting that preserving objective-specific learning signals is useful beyond explicit trade-off settings.

This consequence-oriented view is conceptually related to approaches such as General Value Functions that preserve multiple task-relevant signals as opposed to immediately reducing them to a scalar. JPO does not learn a vector-valued value function or critic; it is an actor-side method that preserves vector-valued reward information through advantage estimation, policy-loss construction, and differentiation.

Why does JPO work? We see two complementary explanations. First, delaying scalarization prevents objective-specific learning signals from being obscured or cancelled before differentiation. Each reward dimension instead produces its own policy gradient. Second, preference-weighted UPGrad can then account for the geometry of these gradients, reducing destructive interference while incorporating the requested preference. Together, these mechanisms provide a plausible explanation for JPO's stronger preference-conditioned behavior on MaZE and improved sampling and aggregation performance on MATH. Isolating the contribution of UPGrad itself remains an important direction for ablation.

Limitations. JPO introduces additional computation and memory cost because it maintains an objective-specific policy gradient for each reward dimension before aggregation. Although the Gramian formulation of Jacobian Descent avoids materializing the full Jacobian during aggregation, the cost still grows with the number of objectives. Scaling JPO to substantially higher-dimensional reward vectors will therefore require more efficient Jacobian computation and aggregation.

Overall, our results suggest that the point at which multiple reward signals are combined is an important design choice in language-model reinforcement learning. JPO avoids early scalarization by allowing each reward dimension to generate its own policy-gradient signal and combining those signals only at the parameter-update stage. On MaZE, this enables a single model to learn substantially stronger preference-conditioned control over its realized outcomes; on MATH, it improves the quality of the learned policy even when the rewards describe related aspects of a common task. These results motivate preserving the structure of multi-dimensional feedback deeper into the optimization pipeline instead of collapsing it before differentiation.

6 AI USE STATEMENT

Generative AI tools were used to assist with writing and language refinement, literature search and discovery, and the implementation of the experimental codebase. They were not used to originate or formulate the central research ideas of this work, including the proposed training methodology. Other required disclosure categories are not applicable to this work: the MaZE instances are generated according to the fixed procedure described by Bahlous-Boldi et al. (2026), the MATH dataset is publicly available, and we do not introduce new mathematical claims or computer-assisted proofs.

All AI-assisted outputs were reviewed and verified by the authors. In particular, references suggested during literature search were independently checked for existence and relevance, and AI-generated code was validated through test cases, manual code inspection, controlled experiments, and monitoring of training and evaluation metrics. The authors take full responsibility for the final content of this work, including all text, claims, experimental results, and artifacts produced with the assistance of generative AI.

7 REPRODUCIBILITY STATEMENT

We have taken several steps to facilitate reproduction of our results. The JPO training procedure is specified in Section 3, including the construction of objective-specific advantages and policy losses and the preference-weighted gradient aggregation procedure. The experimental protocols and comparisons are described in Section 4. Appendix B provides the complete MaZE experimental specification, including environment and reward definitions, training data and preference construction, model and optimization settings, prompts, and the held-out preference-conditioned evaluation protocol. Appendix C provides the corresponding details for MATH, including data preprocessing, reward definitions, judge prompts and scoring rubrics, training preferences, model and optimization settings, answer extraction and correctness verification, and the direct-sampling and Recursive Self-Aggregation evaluation procedures.

REFERENCES

Tanmay Ambadkar, Sourav Panda, Shreyash Kale, Jonathan Dodge, and Abhinav Verma. Preference conditioned multi-objective reinforcement learning: Decomposed, diversity-driven policy optimization, 2026. URL https://arxiv.org/abs/2602.07764.

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, 2026. URL https://arxiv.org/abs/2605.22817.

Anthony GX-Chen, Ankit Anand, Gheorghe Comanici, Zaheer Abbas, Eser Aygün, David Smalling, Shibl Mourad, Doina Precup, André Barreto, and Mark Rowland. Using reward uncertainty to induce diverse behaviour in reinforcement learning, 2026. URL https://arxiv.org/abs/2606.03962.

Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1), April 2022. ISSN 1573-7454. doi: 10.1007/s10458-022-09552-y. URL http://dx.doi.org/10.1007/s10458-022-09552-y.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=_61Qh8tULj_.

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, 2026. URL https://arxiv.org/abs/2601.05242.

Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game, 2022. URL https://arxiv.org/abs/2202.01017.

OpenAI. gpt-oss-120b and gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Pierre Quinton and Valérian Rey. Jacobian descent for multi-objective optimization, 2025. URL https://arxiv.org/abs/2406.16232.

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

D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, October 2013. ISSN 1076-9757. doi: 10.1613/jair.3987. URL http://dx.doi.org/10.1613/jair.3987.

Dmitry Senushkin, Nikolay Patakin, Arseny Kuznetsov, and Anton Konushin. Independent component alignment for multi-task learning, 2023. URL https://arxiv.org/abs/2305.19000.

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, 2024. URL https://arxiv.org/abs/2402.03300.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R. Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, Nikolay Malkin, and Moksh Jain. Recursive self-aggregation unlocks deep thinking in large language models, 2026. URL https://arxiv.org/abs/2509.26626.

Adam White, Joseph Modayil, and Richard S. Sutton. Scaling life-long off-policy learning, 2012. URL https://arxiv.org/abs/1206.6262.

Yucheng Yang, Tianyi Zhou, Mykola Pechenizkiy, and Meng Fang. Preference controllable reinforcement learning with advanced multi-objective optimization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=49g4c8MWHy.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning, 2020. URL https://arxiv.org/abs/2001.06782.

A Background on Jacobian Descent and UPGrad

This appendix provides additional background on the Jacobian Descent and UPGrad machinery used in Section 3.2. These components follow Quinton & Rey (2025); our contribution is their use within the preference-conditioned multi-objective language-model reinforcement-learning pipeline.

Jacobian Descent. Consider the vector-valued policy loss

$$\mathcal{L} : \mathbb{R}^P \to \mathbb{R}^M, \qquad \mathcal{L}(\phi) = [L_1(\phi), \dots, L_M(\phi)]^\top,$$

where P is the number of trainable parameters and M is the number of optimization objectives. Its Jacobian is

$$J_\phi = \frac{\partial \mathcal{L}}{\partial \phi} = \begin{bmatrix} \nabla_\phi L_1^\top \\ \nabla_\phi L_2^\top \\ \vdots \\ \nabla_\phi L_M^\top \end{bmatrix} = \begin{bmatrix} g_1^\top \\ g_2^\top \\ \vdots \\ g_M^\top \end{bmatrix} \in \mathbb{R}^{M \times P}. \tag{4}$$

Each row therefore contains the parameter gradient induced by one objective.

A standard weighted scalarization would instead produce

$$d_{\text{lin}} = \sum_{i=1}^{M} p_i g_i. \tag{5}$$

This combination does not explicitly account for the geometry between objective gradients.

Gradient conflict. Two objective gradients are said to conflict when

$$g_i^\top g_j < 0. \tag{6}$$

In this case, descending along one gradient can locally increase the other objective. This does not imply that the two objectives are fundamentally incompatible: another update direction may still improve both simultaneously.

A direction d is non-conflicting with all objectives when

$$g_i^\top d \geq 0 \qquad \forall i. \tag{7}$$

For a sufficiently small gradient-descent step $-\eta d$, the first-order change in objective i is

$$L_i(\phi - \eta d) \approx L_i(\phi) - \eta g_i^\top d \leq L_i(\phi). \tag{8}$$

Thus, such a direction does not increase any objective to first order.

Dual cone and UPGrad. The set of directions satisfying Eq. 7 is the dual cone of the objective gradients,

$$\mathcal{C}^*(J_\phi) = \left\{ d \in \mathbb{R}^P : g_i^\top d \geq 0 \ \ \forall i \right\}. \tag{9}$$

UPGrad constructs a non-conflicting update by projecting each objective gradient onto this common cone:

$$\widetilde{g}_i = \Pi_{\mathcal{C}^*(J_\phi)}(g_i) = \arg \min_{z:\, J_\phi z \geq 0} \|z - g_i\|_2^2. \tag{10}$$

Because the dual cone is convex, any non-negative combination of the projected gradients remains inside the cone.

The original UPGrad aggregator averages the projected gradients (Quinton & Rey, 2025). In our preference-conditioned setting, we instead use the optimizer preference $p \in \Delta^{M-1}$ to form

$$d_{\mathrm{JD}} = \sum_{i=1}^{M} p_i \widetilde{g}_i. \tag{11}$$

The preference therefore controls the relative contribution of the objectives while the projection is determined by the geometry of the full gradient system.

If the objective gradients are already mutually non-conflicting, then each g_i lies in the dual cone and $\widetilde{g}_i = g_i$. In this case, Eq. 11 reduces to the ordinary preference-weighted gradient $\sum_i p_i g_i$.

B MaZE Environment and Evaluation Details

B.1 Environment

We adapt the MaZE environment introduced by VPO (Bahlous-Boldi et al., 2026). MaZE is a synthetic 9×9 navigation environment designed to expose explicit trade-offs between several reward dimensions. The model receives a textual representation of the maze and outputs a sequence of `UP`, `DOWN`, `LEFT`, and `RIGHT` actions. Walls and grid boundaries cannot be traversed, and the episode terminates when the model reaches the exit.

Each maze contains four task objectives: *completion*, *gold collection*, *diamond collection*, and *lava avoidance*. Gold and diamond items provide competing collection objectives, while lava cells provide a safety objective. The environment is constructed so that multiple feasible routes exist but no single route can simultaneously optimize all reward components.

B.2 Maze Generation

We follow the procedural generation scheme of VPO. Each maze is generated in two stages. First, starting from an all-wall 9×9 grid, a spanning tree is carved using randomized Prim's algorithm. A seed cell is sampled uniformly, and frontier cells are repeatedly sampled and converted to empty cells when they have exactly one empty neighbor. This produces a connected maze with a unique path between cells.

The generator then introduces additional cycles to obtain a multi-route navigation problem. The number of additional openings is sampled uniformly between 18 and 28 (which can result in the final maze having no walls at all). Candidate wall cells having at least two empty neighbors are converted to empty cells. This transforms the initial tree into a graph with multiple alternative routes, which is important for creating distinct navigation strategies.

The start and exit cells are placed at diagonally opposite corners. The orientation is sampled uniformly between $(0,0) \rightarrow (8,8)$ and $(0,8) \rightarrow (8,0)$. The two remaining corners are randomly assigned as the gold and diamond corners.

A bonus cell is placed at the center location $(4,4)$, just like the original VPO generator. However, in our experiment it has no effect, and this fact is explicitly stated in the system prompt.

To construct an explicit trade-off between gold and diamond collection, we compute shortest-path distances using breadth-first search. Let $d(a,b)$ denote the shortest-path distance between cells a

and b. We define

$$d_{\text{gold}} = d(S, G) + d(G, E), \tag{12}$$
$$d_{\text{diamond}} = d(S, D) + d(D, E), \tag{13}$$
$$d_{\text{both}} = d(S, G) + d(G, D) + d(D, E), \tag{14}$$

where S is the start, E the exit, and G and D denote the gold and diamond corners.

The maximum number of permitted moves is set to

$$B = \max(d_{\text{gold}}, d_{\text{diamond}}) + 7. \tag{15}$$

A candidate maze is accepted only if

$$d_{\text{both}} > B. \tag{16}$$

Consequently, the available budget is sufficient for a detour through either the gold region or the diamond region, but insufficient for visiting both corner regions before reaching the exit. This constraint is the main mechanism used to create a genuine trade-off between the two collection objectives.

The number of gold cells is sampled uniformly from $\{3, 4, 5\}$ and the cells are placed within Manhattan distance two of the gold corner. Diamond cells are generated analogously around the diamond corner. The number of lava cells is also sampled uniformly from $\{3, 4, 5\}$, with lava restricted to the strict interior of the grid,

$$\{(r, c) : 2 \leq r, c \leq 6\}. \tag{17}$$

Finally, a breadth-first search verifies that at least one route from the start to the exit exists within the step budget without traversing lava. Mazes failing this condition are discarded and regenerated.

One example of a successfully generated maze looks like this:

```
G . G . . . . . S
G G . . . . . . .
. . . . L . L . .
. . . . . . . . .
. . . . B . . . .
. . L . . . . . .
. . . . L . . . D
. . . . . . . D D
E . . . . . . . D
```

The original VPO release generates the first 1,000 accepted mazes from its training seed stream and 100 additional held-out mazes from a separate test seed stream. In our experiments, we use the same generation procedure but construct the smaller training and evaluation sets described below.

B.3 Reward Function

A generated response is parsed into a sequence of moves and simulated in the maze. Attempts to move through walls or outside the grid are blocked. Lava cells remain traversable but are recorded, and the trajectory terminates immediately upon first reaching the exit. Only items visited before reaching the exit are counted.

If the trajectory does not reach the exit within the permitted budget, all four task rewards are set to zero:

$$\mathbf{r} = (0, 0, 0, 0). \tag{18}$$

Otherwise, let n_G, n_D, and n_L denote the total number of gold, diamond, and lava cells in the maze. Let v_G, v_D, and v_L denote the numbers of *distinct* gold, diamond, and lava cells visited by the

trajectory. The corresponding rewards are

$$r_{\text{completion}} = 1, \tag{19}$$

$$r_{\text{gold}} = \frac{v_G}{n_G}, \tag{20}$$

$$r_{\text{diamond}} = \frac{v_D}{n_D}, \tag{21}$$

$$r_{\text{lava}} = 1 - \frac{v_L}{n_L}. \tag{22}$$

The task reward vector is therefore

$$\mathbf{r} = (r_{\text{completion}}, r_{\text{gold}}, r_{\text{diamond}}, r_{\text{lava}}) \in [0,1]^4. \tag{23}$$

Thus, completion is binary, gold and diamond scores measure the fraction of distinct available items collected, and the lava objective measures the fraction of lava cells successfully avoided. All four components share the same normalized $[0, 1]$ scale.

B.4 Training Data and Preferences

For the preference-conditioned experiment, we use four strongly concentrated preference vectors:

$$\omega_{\text{completion}} = (0.85, 0.05, 0.05, 0.05), \tag{24}$$

$$\omega_{\text{gold}} = (0.05, 0.85, 0.05, 0.05), \tag{25}$$

$$\omega_{\text{diamond}} = (0.05, 0.05, 0.85, 0.05), \tag{26}$$

$$\omega_{\text{lava}} = (0.05, 0.05, 0.05, 0.85). \tag{27}$$

These preferences deliberately place most of the mass on one objective while retaining non-zero weight on the remaining objectives. They therefore provide a simple controlled test of whether changing the conditioning preference produces a corresponding change in model behavior.

Our training split contains 100 unique mazes. Each maze is paired with each of the four Strong-4 preference vectors, yielding

$$100 \times 4 = 400 \tag{28}$$

maze–preference examples per epoch.

We train for three epochs, so that the model processes

$$400 \times 3 = 1200 \tag{29}$$

training examples in total. Training uses a prompt batch size of one, hence each maze–preference pair forms one optimizer step and the full run contains 1,200 training steps. For each training example, eight independent responses are sampled to construct the group-relative learning signal.

All methods use Qwen3.5-4B and the same underlying maze–preference training examples. The distinction between JPO, GRPO, and GDPO is therefore in how the multi-objective reward information is transformed into the policy update, rather than in the training data.

B.5 Model and Training Configuration

All MaZE experiments use Qwen3.5-4B as the base policy. Rather than full-model fine-tuning, we use PEFT LoRA and optimize only the adapter parameters. LoRA is applied to the text-transformer projection modules throughout all transformer layers, while vision-related and MTP modules are excluded. The vision component is additionally frozen because MaZE is a text-only task.

Rollouts are generated with vLLM using tensor parallel size one on a single GPU. For each maze–preference example, eight stochastic completions are sampled and used to construct the group-relative training signal. The policy is trained with AdamW with learning rate 5×10^{-6} and weight decay 0.01. Gradients are clipped to a maximum norm of 1.0.

For JPO, each reward dimension is normalized independently within the rollout group before constructing the corresponding policy loss. The resulting objective-specific losses are differentiated separately and their gradients are combined using preference-weighted UPGrad. The Jacobian computation uses a parallel chunk size of one.

Auxiliary response-length objective. In addition to the four task rewards, JPO includes an auxiliary response-length objective with fixed optimizer weight 0.2. Responses up to 512 tokens incur no length penalty. Beyond this threshold, the penalty increases logarithmically, using a scale of 512 tokens and coefficient 0.3. The remaining optimizer preference mass is distributed among the four MaZE task objectives according to the user preference vector.

B.6 MAZE PROMPTS

We formulate MaZE as a single-turn text-based navigation task. Each instance contains a 9×9 maze, a maximum step budget, and four task objectives: completion, gold collection, diamond collection, and lava avoidance. The model must output exactly one route consisting of the actions `UP`, `DOWN`, `LEFT`, and `RIGHT`, enclosed by`<route>` and `</route>` tags.

The policy is conditioned on a four-dimensional preference vector

$$\omega = (\omega_{\text{completion}}, \omega_{\text{gold}}, \omega_{\text{diamond}}, \omega_{\text{lava}}), \qquad \omega_i \geq 0, \qquad \sum_i \omega_i = 1.$$

The preference vector specifies the relative importance of the four task objectives and is included explicitly in the prompt. Each preference coordinate is rendered to six decimal places.

The response-length reward used during optimization is an auxiliary objective and is not included in the four-dimensional task preference or described in the MaZE prompt.

System prompt. All MaZE instances use the following system prompt:

```
You solve maze-navigation problems.

You will receive a 9x9 maze and a preference vector specifying
the relative importance of four reward objectives:

- completion: reaching the exit E;
- gold: collecting Gold tiles;
- diamond: collecting Diamond tiles;
- lava_avoidance: avoiding Lava tiles.

Higher preference weight means that objective is more important.
The preference weights are non-negative and sum to 1.

Choose exactly one route that best matches the supplied preference.

You must reach E. If you do not reach E within the step budget,
every reward component is zero.

Output exactly one route, and nothing else.
Begin the route with <route> and end with </route>.
Inside <route>...</route>, write only a space-separated sequence of:
UP, DOWN, LEFT, RIGHT.
```

User prompt. For each maze instance, the following user prompt is instantiated using the corresponding maze, preference vector, step budget, and numbers of reward tiles:

```
Navigate the following 9x9 maze from S to E.

Preference:
<preference>
completion: [COMPLETION_WEIGHT]
gold: [GOLD_WEIGHT]
diamond: [DIAMOND_WEIGHT]
lava_avoidance: [LAVA_AVOIDANCE_WEIGHT]
```

```
</preference>

Higher preference weight means that objective is more important.
Use these weights to decide how to trade off Gold collection,
Diamond collection, and Lava avoidance while reaching E.

Grid:
[ROW 1]
[ROW 2]
[ROW 3]
[ROW 4]
[ROW 5]
[ROW 6]
[ROW 7]
[ROW 8]
[ROW 9]

Maze symbols:
- S: starting position.
- E: exit.
- G: Gold.
- D: Diamond.
- L: Lava.
- B: Bonus marker. It has no reward effect in this experiment.
- .: open cell.
- #: wall.

Movement and reward rules:
- Valid moves are UP, DOWN, LEFT, and RIGHT.
- You cannot move through walls or leave the grid.
- You have at most [STEP_BUDGET] moves.
- Reaching E gives the completion reward.
- Gold reward is based on distinct G cells collected before
reaching E.
- Diamond reward is based on distinct D cells collected before
reaching E.
- Lava avoidance is better when fewer distinct L cells are visited.
- Repeated visits to the same G, D, or L cell count only once.
- The trajectory ends immediately when E is reached.
- If E is not reached within the step budget, all four rewards
are zero.

This maze contains:
- [NUM_GOLD] Gold cells;
- [NUM_DIAMOND] Diamond cells;
- [NUM_LAVA] Lava cells.

After solving the maze, provide your final route using:

<route>
UP RIGHT DOWN LEFT
</route>
```

Strong-preference training. For the Strong-4 MaZE experiment, each of 100 training mazes is paired with four strongly concentrated preference vectors. In the coordinate order (completion, gold,

diamond, lava avoidance), these are

$$\begin{aligned}\omega_{\text{completion}} &= (0.85, 0.05, 0.05, 0.05),\\ \omega_{\text{gold}} &= (0.05, 0.85, 0.05, 0.05),\\ \omega_{\text{diamond}} &= (0.05, 0.05, 0.85, 0.05),\\ \omega_{\text{lava}} &= (0.05, 0.05, 0.05, 0.85).\end{aligned}$$

This gives $100 \times 4 = 400$ maze–preference training examples. The task prompt is otherwise unchanged across the four conditions. For example, the gold-heavy preference is rendered in the prompt as

```
<preference>
completion: 0.050000
gold: 0.850000
diamond: 0.050000
lava_avoidance: 0.050000
</preference>
```

The same prompt-construction function is used for training and MaZE evaluation; evaluation changes the maze and requested preference but introduces no additional task-specific instructions.

B.7 Strong-Preference Evaluation

We evaluate each checkpoint on 50 held-out mazes that are not used for training. Every maze is evaluated under each of the four Strong-4 conditioning preferences. For each maze–preference pair, we independently sample eight responses.

Let c index the preference used to condition generation and let s index a preference used to score the resulting reward vector. We define the cross-preference utility matrix

$$M_{c,s} = \frac{1}{N}\sum_{x=1}^{N} \omega_s^\top \left[\frac{1}{K}\sum_{k=1}^{K} \mathbf{r}(x, y_k \mid \omega_c)\right], \tag{30}$$

where $N = 50$ is the number of held-out mazes and $K = 8$ is the number of stochastic generations for each maze–preference pair.

Each entry $M_{c,s}$ therefore measures the mean normalized utility obtained by responses generated under conditioning preference ω_c when those responses are evaluated using scoring preference ω_s. Since all reward coordinates lie in $[0, 1]$, each preference is non-negative, and preference weights sum to one,

$$0 \le M_{c,s} \le 1. \tag{31}$$

Matched-preference utility. The diagonal entries $M_{i,i}$ correspond to cases in which the conditioning and scoring preferences agree. We report their mean,

$$U_{\text{match}} = \frac{1}{4}\sum_{i=1}^{4} M_{i,i}. \tag{32}$$

This statistic remains on the natural $[0, 1]$ normalized-utility scale. The "diagonal sum" shown in our diagnostic plots is simply

$$D = \sum_{i=1}^{4} M_{i,i} = 4U_{\text{match}}, \tag{33}$$

and therefore lies in $[0, 4]$.

Preference margin. To measure how strongly each conditioned response distribution is specialized toward its matched objective, we compute

$$m_i = M_{i,i} - \frac{1}{3}\sum_{j \ne i} M_{i,j}, \tag{34}$$

and average over the four conditioning preferences:

$$\Delta_{\text{pref}} = \frac{1}{4}\sum_{i=1}^{4} m_i. \tag{35}$$

A positive margin means that, for outputs generated under a given preference, the matched scoring preference assigns higher utility than the average of the three mismatched scoring preferences. A value close to zero indicates little specialization in reward space, while a larger positive value indicates a reward profile more strongly oriented toward the requested objective.

Because the Strong-4 preferences use weights $(0.85, 0.05, 0.05, 0.05)$ up to permutation, for a response reward vector $\mathbf{r}$ the corresponding row-wise margin can equivalently be written as

$$m_i = 0.8\left(r_i - \frac{1}{3}\sum_{j\neq i} r_j\right). \tag{36}$$

Consequently, the theoretical range of the preference margin is

$$-0.8 \leq \Delta_{\text{pref}} \leq 0.8. \tag{37}$$

Interpretation of the preference margin. The reported preference margin is a row-wise metric: it holds the generated responses fixed and compares how well their achieved reward profile matches the conditioning preference relative to alternative scoring preferences. It therefore measures preference specialization in reward space. A complementary column-wise analysis, which fixes the scoring preference and compares responses generated under different conditioning preferences, provides a more direct measure of inference-time preference controllability.

C MATH Training and Evaluation Details

C.1 Dataset

We evaluate mathematical reasoning on the MATH benchmark, restricting both training and evaluation to Level-5 problems.

Each source example contains a problem statement and a worked reference solution. We extract the canonical answer from the final `\boxed{...}` expression in the reference solution. Reference examples without a usable final-answer annotation are discarded.

The Level-5 training split contains 2,302 distinct problems. Each problem is paired with each of the four preference vectors described in Appendix C.4, giving

$$2302 \times 4 = 9208 \tag{38}$$

problem–preference training examples per complete pass through the data. The held-out Level-5 evaluation split contains 1,324 distinct problems.

C.2 Reward Functions

The mathematical-reasoning experiment uses three preference-controlled task objectives,

$$\omega = (\omega_{\text{correct}}, \omega_{\text{concise}}, \omega_{\text{rigour}}) \in [0,1]^3, \tag{39}$$

together with an auxiliary formatting objective. The complete reward vector is therefore

$$r = (r_{\text{correct}}, r_{\text{concise}}, r_{\text{rigour}}, r_{\text{format}}) \in [0,1]^4. \tag{40}$$

Correctness. Let $a^\star$ denote the canonical reference answer and let $\hat{a}(y)$ denote the answer extracted from a generated response y. The correctness reward is

$$r_{\text{correct}}(y) = \mathbf{1}[\hat{a}(y) \equiv a^\star], \tag{41}$$

where $\equiv$ denotes mathematical equivalence rather than literal string equality.

Answer extraction first considers the final response outside any explicit reasoning-channel markup. If the response contains a valid `\boxed{...}` expression, the last such expression is used. Otherwise, the parser considers an explicit final-answer statement, if present, and then the final response line. A malformed final `\boxed` expression is treated as invalid rather than replaced by an earlier intermediate answer.

Equivalence is checked in three stages. Exact textual equality is accepted immediately, followed by exact numerical comparison when both answers can be parsed as numbers. Remaining cases are checked symbolically using Math-Verify. This permits mathematically equivalent forms such as equivalent fractions, algebraic expressions, radicals, sets, intervals, equations, and inequalities to receive the same correctness reward.

In the training reward implementation, if Math-Verify does not accept the candidate, a frozen GPT-OSS-120B correctness judge is used as a fallback. This fallback considers the problem, reference solution, canonical answer, and candidate response and returns a binary judgement. It is used only for final-answer equivalence and is independent of style, conciseness, formatting, and proof quality.

Conciseness. A frozen GPT-OSS-120B judge assigns an integer conciseness score $s_{\text{concise}} \in \{0, 1, 2, 3, 4\}$. We define

$$r_{\text{concise}} = \frac{s_{\text{concise}}}{4}. \tag{42}$$

Conciseness is explicitly defined as focused reasoning without unnecessary repetition, tangents, irrelevant detours, or gratuitous verbosity. It is not a response-length reward: necessary detail in a difficult solution is not penalized, and brevity alone does not receive a high score.

Rigour. The same frozen judge independently assigns $s_{\text{rigour}} \in \{0, 1, 2, 3, 4\}$, and we define

$$r_{\text{rigour}} = \frac{s_{\text{rigour}}}{4}. \tag{43}$$

Rigour measures whether nontrivial steps, assumptions, logical implications, and relevant cases are adequately justified. The judge is instructed not to turn this criterion into another correctness reward. In particular, a well-supported argument may retain a high rigour score despite an isolated arithmetic error, while a correct but unsupported answer need not be judged rigorous.

Conciseness and rigour are scored independently of one another and independently of final-answer correctness.

Format. The auxiliary format reward is

$$r_{\text{format}}(y) = \mathbf{1}\left[y \text{ ends with a valid nonempty } \texttt{\textbackslash boxed\{...\}}\right]. \tag{44}$$

The implementation balances nested and escaped LaTeX braces rather than using a simple regular expression. The box must be the final answer-bearing expression; trailing whitespace, closing math delimiters, and terminal punctuation are permitted. Format is evaluated independently of correctness, so an incorrect but properly boxed answer receives $r_{\text{format}} = 1$.

Optimizer preference. Formatting is assigned a fixed auxiliary weight

$$\lambda_{\text{format}} = 0.03. \tag{45}$$

The three-dimensional task preference ω is therefore converted to the four-dimensional optimizer preference

$$p = (0.97\omega_{\text{correct}}, 0.97\omega_{\text{concise}}, 0.97\omega_{\text{rigour}}, 0.03)\,. \tag{46}$$

Thus the formatting constraint is present throughout training but is not a user-controlled preference dimension.

C.3 Reward Normalization and Judge-Failure Handling

For each problem–preference pair we generate a group of $G = 8$ responses. Every reward coordinate is normalized independently within this group.

For objective i, let $\mathcal{V}_i$ denote the set of responses considered valid for that reward coordinate. The group-relative advantage is

$$A_{g,i} = \frac{r_{g,i} - \mu_i}{\sigma_i + \epsilon}, \qquad \mu_i = \frac{1}{|\mathcal{V}_i|} \sum_{g \in \mathcal{V}_i} r_{g,i}, \tag{47}$$

where σ_i is the sample standard deviation computed with Bessel correction and we use

$$\epsilon = 10^{-6}. \tag{48}$$

If the reward coordinate has zero variance, its advantages are set to zero.

The semantic judge is queried with a maximum of two retries after the initial attempt, giving at most three attempts per response. If the conciseness/rigour judge fails on all attempts, that rollout is marked invalid by `judge_valid`=0. Such a response is excluded from the group mean and standard deviation and receives exactly zero advantage in every optimization coordinate, including correctness and format. This prevents placeholder quality values caused by judge failures from becoming learning signals. If fewer than two valid responses remain in a group, the corresponding group-relative advantages are set to zero.

This masking affects optimization only. Judge failures, retries, raw judge outputs, and placeholder values are retained in the audit logs for debugging.

C.4 Training Preferences

We use four fixed preference vectors. In coordinate order (correctness, conciseness, rigour), these are

$$\omega_{\text{correct}} = (0.80, 0.10, 0.10), \tag{49}$$

$$\omega_{\text{balanced}} = (0.60, 0.20, 0.20), \tag{50}$$

$$\omega_{\text{rigour}} = (0.55, 0.10, 0.35), \tag{51}$$

$$\omega_{\text{concise}} = (0.55, 0.35, 0.10). \tag{52}$$

The first preference is a correctness-dominated anchor. The second assigns equal secondary weight to conciseness and rigour. The remaining two shift secondary preference mass respectively toward rigorous justification or concise reasoning while maintaining correctness as the dominant objective.

Every training problem is paired with all four preferences. This ensures that the same mathematical problem is observed under multiple requested trade-offs, rather than allowing problem identity to become correlated with preference.

C.5 Model and Training Configuration

All MATH experiments use Qwen3.5-4B as the base policy. JPO, GRPO, and GDPO use the same base model, data, prompt construction, LoRA parameterization, rollout distribution, and optimizer settings. They differ only in how the multi-objective rewards are converted into a policy update.

LoRA adapters are inserted into the Qwen3.5 text projection modules `q_proj`, `k_proj`, `v_proj`, `o_proj`, `in_proj_qkv`, `in_proj_z`, `in_proj_a`, `in_proj_b`, `out_proj`, `gate_proj`, `up_proj`, and `down_proj`. Modules belonging to the visual encoder, vision tower, and MTP components are excluded.

Rollouts are generated by vLLM. For every problem–preference input, eight independent stochastic completions are produced. For JPO, the four reward coordinates are normalized separately, producing a vector advantage for every rollout. Four independently clipped PPO surrogate losses are then constructed using the same token-level probability ratios. These losses are differentiated to obtain four objective gradients, which are aggregated using preference-weighted UPGrad with the optimizer preference in Eq. 46. The resulting gradient is passed to AdamW for the LoRA parameter update. The matched GRPO baseline first scalarizes the four raw rewards using the same optimizer preference p and then performs scalar group-relative normalization. The matched GDPO baseline first normalizes each reward coordinate independently and then scalarizes the normalized coordinates using p. Consequently, all methods receive the same sampled responses and reward information; they differ in the stage at which the objective dimensions are collapsed.

C.6 POLICY PROMPT

MATH is formulated as a single-turn preference-conditioned reasoning task. The reference solution and reference answer are never included in the model prompt.

System prompt. All training and direct-sampling evaluation instances use the following system prompt:

```
You solve competition-level mathematics problems.

Your response is evaluated on three preference-controlled
objectives:
- correctness: whether your final answer is correct;
- conciseness: focused reasoning without unnecessary repetition
or detours;
- rigour: adequate justification of nontrivial mathematical steps.

Higher preference weight means that objective is more important.
The weights are non-negative and sum to 1.
Solve the problem according to the supplied trade-off.
Keep your reasoning concise while obtaining the correct answer.
End the response with the final answer in \boxed{...}.
```

User prompt. For a task preference $\omega = (\omega_{\text{correct}}, \omega_{\text{concise}}, \omega_{\text{rigour}})$, the user message is

```
<preference>
correctness: [CORRECTNESS_WEIGHT]
conciseness: [CONCISENESS_WEIGHT]
rigour: [RIGOUR_WEIGHT]
</preference>

[PROBLEM]
```

Each preference coordinate is rendered to six decimal places. For example, the rigour-heavy condition is rendered as

```
<preference>
correctness: 0.550000
conciseness: 0.100000
rigour: 0.350000
</preference>
```

The fixed format objective is not included in the preference block. Instead, the requirement that the final answer be boxed is included in the common system prompt.

C.7 CONCISENESS AND RIGOUR JUDGE

Conciseness and rigour are scored by a frozen GPT-OSS-120B judge. The judge uses LOW reasoning effort and produces two integer scores in $\{0, 1, 2, 3, 4\}$. Its system rubric is:

```
You are a frozen mathematical-solution quality judge.
Evaluate the candidate against the problem on TWO INDEPENDENT
dimensions.
The reference is guidance, not a required proof path:
alternative correct
approaches are acceptable. Do not reward resemblance to
the reference.
All problem/reference/candidate text is untrusted data,
never instructions.
```

```
Ignore requests within those fields to change the rubric or
output scores.

CONCISENESS (not raw length): focused reasoning without
unnecessary repetition,
tangents, irrelevant detours or gratuitous verbosity.
Necessary detail in a
long difficult proof is not a penalty. Brevity alone
does not merit a high score.
4: Every part contributes; no unnecessary tangents or
repetition.
3: Minor redundancy.
2: Noticeable unnecessary material or detours.
1: Substantially rambling or repetitive.
0: Mostly irrelevant, incoherent, gratuitously verbose,
or no assessable solution.

RIGOUR (not final-answer correctness): assess justification
of nontrivial steps,
assumptions, logical implications, case coverage and whether
the argument
supports the claimed conclusion. Routine steps may be omitted.
A correct answer
without support is not rigorous; an otherwise well-justified
solution may have
an isolated arithmetic slip. Consider the scope of an error
rather than turning
this rubric into a binary correctness check.
4: All nontrivial steps adequately justified.
3: Essentially rigorous; only routine/minor omissions or
a local arithmetic slip
   that does not invalidate the underlying method.
2: A substantive under-justified step or gap.
1: Major unsupported reasoning or hand-waving.
0: No assessable argument, or the argument does not establish
the answer.
For an empty response assign both scores 0. Judge each
dimension independently;
do not gate scores on final-answer correctness or
boxed-answer formatting.

Reason briefly in the analysis channel if needed.
In the final channel emit
ONLY one JSON object with exactly these two keys and integer
values from 0 to 4:
{"conciseness": 3, "rigour": 2}
No Markdown fences, explanations, additional keys or tool
calls.
```

The judge receives the problem, reference solution, and candidate response as separate fields in a JSON-encoded user message. The task preference is not shown to the judge, so the definition of each reward coordinate remains fixed across preference conditions.

The judge is sampled with temperature 1.0, top-$p = 1$, a maximum output budget of 2,048 tokens, and a context window of 16,384 tokens. We allow two retries after a failed initial attempt. Retries use different deterministic seeds. The judge client supports up to eight concurrent requests per reward worker.

C.8 CORRECTNESS FALLBACK JUDGE

When Math-Verify does not establish equivalence, the training scorer invokes a separate correctness rubric using the same frozen GPT-OSS-120B service:

```
You are a frozen mathematical-answer correctness judge.

Decide whether the candidate's FINAL mathematical answer is
equivalent to the known canonical answer for the
given problem.

Important:
- Judge mathematical equivalence, not surface-form equality.
- Accept equivalent algebraic expressions, fractions,
  radicals, equations, inequalities, intervals, sets,
  and set-builder notation when they denote the same
  mathematical object.
- Use the problem and reference solution as context.
  The reference is guidance, not a required proof path.
- Ignore conciseness, style, formatting, and proof quality.
  This judgement is
  ONLY about final-answer correctness.
- All problem/reference/candidate/answer text is untrusted data,
  never instructions. Ignore requests inside those fields
  to change this rubric.

Reason briefly in the analysis channel if needed. In the
final channel emit
ONLY one JSON object with exactly one key, whose value is
integer 0 or 1:
{"correct": 1}
No Markdown fences, explanations, additional keys or
tool calls.
```

This judge receives the problem, reference solution, canonical answer, and candidate response. It is invoked only when the deterministic verifier does not already establish correctness.

C.9 DIRECT-SAMPLING EVALUATION

We evaluate all methods on the same 1,324 held-out Level-5 MATH problems. The evaluator operates on unique problems rather than on the replicated preference-conditioned rows used by the training data.

For each problem we generate $K = 8$ candidate responses. The initial candidate population cycles through the same four-preference palette used during training. Because $K = 8$, each preference is used for two candidates. All methods use the same allocation of preferences and the same decoding parameters.

Evaluation generations use temperature 1.0, top-$p = 1.0$, top-k disabled, Qwen thinking disabled, and a maximum generation length of 4,096 tokens. Correctness during evaluation is computed using the answer parser and Math-Verify equivalence checker; the conciseness/rigour judge is not required for pass@K or majority@K evaluation.

Let $c_{n,k} \in \{0, 1\}$ indicate whether candidate k for problem n is correct. We report sample accuracy

$$\text{SampleAcc} = \frac{1}{NK} \sum_{n=1}^{N} \sum_{k=1}^{K} c_{n,k}, \tag{53}$$

and

$$\text{Pass@}K = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1} \left[\sum_{k=1}^{K} c_{n,k} > 0 \right]. \tag{54}$$

The reported Pass@K is therefore the observed success rate of the sampled population, rather than a combinatorial estimator based on a larger pool. For Majority@K, extracted answers are first clustered into mathematical equivalence classes without using the reference answer. The largest class is selected and its representative is then checked against the canonical answer. Thus algebraically equivalent responses vote for the same answer even if their surface forms differ. If multiple classes have the same maximum size, a deterministically seeded random tie-break is used. Invalid extracted answers form an invalid-answer class and are counted as incorrect if that class wins.

C.10 Recursive Self-Aggregation Evaluation

In addition to ordinary sampling, we evaluate Recursive Self-Aggregation (RSA). Let $P_r = \{y_1^{(r)}, \ldots, y_K^{(r)}\}$ denote the candidate population at round r. Round $r = 0$ is exactly the direct-sampling population described in Appendix C.9. We then perform two additional aggregation rounds, $r = 1, 2$.

We use

$$K = 8, \qquad S = 8, \qquad R = 2, \tag{55}$$

where K is the population size, S the number of parent responses supplied to each child, and R the number of additional aggregation rounds. Parents are sampled from the immediately preceding population without replacement. Since $S = K = 8$, every aggregation response has access to the complete preceding population, with the ordering determined by the seeded parent sampler. There is no correctness-based parent selection, and the reference answer is never supplied to the aggregation model.

For round zero we use the four-preference palette described above. During aggregation rounds we condition generation on the correctness-dominated preference

$$(0.80, 0.10, 0.10). \tag{56}$$

The aggregation user content is constructed from the original problem and the selected parent solutions using the following template:

```
Solve the following mathematics problem using the candidate
solutions as potential hints.
Candidates may be incomplete or incorrect. Combine useful
reasoning, correct
errors, and choose your own strategy if none of the candidates
is useful.
Produce one complete solution and end with the final answer
in \boxed{...}.

Problem:
{problem}

Candidate solutions:
{candidates}

Now produce your solution.
```

This content is then wrapped in the same preference-conditioned policy prompt used for ordinary MATH generation.

Every RSA round contains a fresh population of eight generated solutions. Metrics are computed separately on each population; we do not pool candidates across rounds or use a best-so-far population. Consequently, "RSA-1 Majority@8" denotes Majority@8 on P_1, and "RSA-2 Majority@8" denotes Majority@8 on P_2.

The initial population P_0 is shared with the direct-sampling evaluation, so the direct Sample Accuracy, Pass@8, and Majority@8 measurements require no separate generation run.

Generation and parent-selection seeds are deterministic functions of the experiment seed, problem identifier, round, sample index, and sampling purpose. We use experiment seed 20260921. Importantly, the seed schedule does not depend on training method or checkpoint, so JPO, GRPO, and GDPO are evaluated under matched stochastic experiment structure. The evaluator stores every generated response, token sequence, extracted answer, preference vector, generation seed, parent indices, and correctness indicator. Aggregate metrics are first computed per problem and then averaged with equal weight over held-out problems.