# LEARNING TO REASON WITH PROCEDURAL GUIDANCE IN MLLMS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) can improve reasoning in large language models (LLMs). However, when all sampled responses receive the same reward, the group-relative advantage in methods such as Group Relative Policy Optimization (GRPO) is zero, so the group provides no reward-driven learning signal. Hint-based RL helps the model avoid all-incorrect rollout groups by supplying part of a reasoning trace, usually as free-form text written by an oracle model. Instead of relying on an oracle model, we introduce *procedural guidance*, in which the program that generates problems and computes their answers also provides correct, controllable guidance. We instantiate this concept in a procedural data generation framework called Vision Reasoning Gym (VRGym), specifically designed for multimodal reasoning problems. The derivations generated by VRGym contain maskable values that can be replaced with an anonymous "?" mark while preserving their reasoning structure. Using these derivations, we design **O**n-Policy **P**rocedurally **G**raduated **G**uidance (OPGG), which maintains a scalar guidance-withdrawal budget for each task across training steps and updates it based on each rollout group's accuracy. As the budget increases, OPGG first masks selected values and then truncates the remaining derivation, allowing the model to learn to solve problems without guidance. With this guidance schedule, OPGG achieves the highest seven-dataset macro-average accuracy among the evaluated hint-based methods on both backbones.

## 1 INTRODUCTION

Reasoning models have shown strong performance on complex problems by producing extended chains of thought before answering (Wei et al., 2022). Large-scale reinforcement learning (RL) has helped language models strengthen their reasoning abilities by allowing them to explore reasoning paths that lead to final solutions (Jaech et al., 2024; Guo et al., 2025). When answer correctness can be checked automatically, reinforcement learning with verifiable rewards (RLVR) offers an effective way to train reasoning models, as shown by its success in mathematics and code (Guo et al., 2025). Many RLVR methods, utilizing Group Relative Policy Optimization (GRPO), train on groups of rule-verified rollouts using a group-relative objective (Shao et al., 2024; Guo et al., 2025; Liu et al., 2025a). However, this objective yields zero advantage when all responses in a group receive the same verifier reward. With binary rewards, this occurs when every response is correct or every response is incorrect. This zero-advantage condition is called *advantage collapse* (He et al., 2026). Such groups provide no reward-based policy gradient, even though sampling and scoring them still incur a computational cost (Yu et al., 2026; Liu et al., 2025a; Le et al., 2026).

One approach to handling advantage collapse is dynamic sampling, which discards uniform groups and draws replacements (Yu et al., 2026). This avoids spending updates on groups with zero advantage, but makes it harder to learn directly from the discarded difficult examples. Hint-based methods instead address the all-incorrect case by providing part of a step-by-step reasoning trace, enabling these examples to yield a learning signal. The trace may be inserted into the sampled trajectory, making the rollout off-policy (Yan et al., 2026; Zhang et al., 2026a; Huang et al., 2025; Guo et al., 2026; Zhang et al., 2025), or kept in the prompt so the rollout remains on-policy (Zhang et al., 2026b; Liu et al., 2025b; Li et al., 2026; Qu et al., 2026; Zhang et al., 2026c). In both cases, guidance is predominantly written by oracle models or distilled from them (Zhang et al., 2026b; Huang et al., 2025; Yan et al., 2026; Guo et al., 2026). These methods can be effective, but their guidance has



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

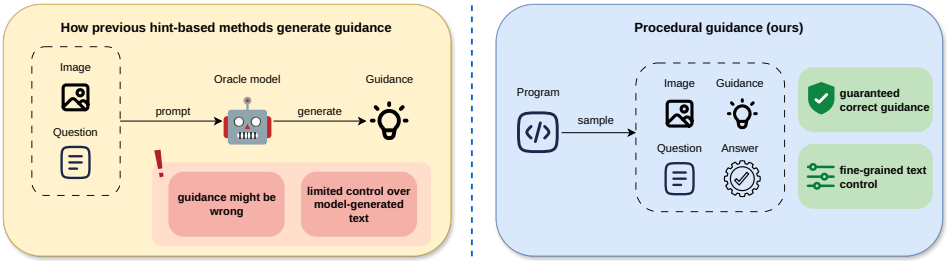



Figure 1: Comparison of guidance sources. Prior hint-based methods use traces generated by an oracle model. Procedural guidance from the problem generator ensures both correctness and controllability.

limitations. Correctness is usually checked only at the final answer, so the intermediate steps of a trace are not guaranteed to be correct. Free-form traces also lack explicit fields for controlling which values a hint reveals.

To address these limitations, we introduce the concept of *procedural guidance*: a problem generator emits a derivation with correct intermediate steps and designated maskable values alongside each problem and answer (Figure 1). Because the derivation is computed by the generator's solution procedure, guidance is available for every generated instance within the supported task settings without an oracle model. The derivation template contains **maskable values** (e.g., numbers, labels, and operation types) that can be replaced with an anonymous `"?"` mark while preserving the derivation's reasoning structure (Sections 3.2 and 3.3).

Inspired by procedural and rule-verifiable reasoning environments (Stojanovski et al., 2026; Yang et al., 2026; Alam, 2026; Meng et al., 2026), we develop **Vision Reasoning Gym (VRGym)**, a procedural data generation framework that incorporates procedural guidance into problem generation. In this work, we focus particularly on multimodal problems and include 68 tasks across mathematical, logical, algorithmic, chart, and table reasoning. Each generated instance includes a derivation with maskable values. Parameters control problem difficulty and the amount of guidance provided by the derivation.

Using these derivations, we design **O**n-Policy **P**rocedurally **G**raduated **G**uidance (OPGG), a scheduling mechanism that uses group accuracy to adjust a persistent guidance-withdrawal budget for each task. Guidance is withdrawn in two stages, first by masking values and then by truncating the derivation. OPGG gradually reduces guidance as each task improves, restoring it when needed. Once a task no longer needs guidance, we call it a *graduate*.

In this paper, our contributions include:

- Unlike traditional procedural reasoning environments, we introduce the concept of *procedural guidance*, which comes from the problem generator rather than an oracle model. This guidance is correct for every instance within the supported task settings, and its maskable values control what it reveals. The concept is not specific to a modality and is likely to be applied to other tasks with programmatically generated solutions.
- **VRGym** is a framework that supplies questions, answers, and procedural guidance for multimodal reasoning across 68 tasks. The guidance for every generated instance can be finely adjusted through parameters (Section 3.2).
- **OPGG** is a scheduling mechanism that exploits the controllability of VRGym's procedural guidance within GRPO. It uses group accuracy to adjust the amount of guidance provided for each task. Across the seven evaluation datasets, OPGG achieves higher average accuracy than every evaluated hint-based baseline, supporting the effectiveness of procedural guidance (Section 4).

# 2 Related Work

## 2.1 Reinforcement Learning with Verifiable Rewards

RLVR trains a policy using rewards from an automatic verifier. GRPO uses a group-relative objective for this training and was introduced for mathematical reasoning (Shao et al., 2024). It has since been applied to general reasoning (Guo et al., 2025) and multimodal models (Huang et al., 2026). Its group-relative baseline gives every response zero advantage when all rewards in a group are equal. Dynamic sampling addresses this issue by discarding groups with uniform rewards and sampling replacements before the policy update (Yu et al., 2026). However, discarding all-incorrect groups makes it harder to learn directly from those difficult samples.

## 2.2 Hint-Based Reinforcement Learning

To learn from these problems rather than discard them, hint-based RL injects part of a reasoning trace so that they still produce a learning signal. Trajectory-based methods place external guidance inside the response (Yan et al., 2026; Huang et al., 2025; Zhang et al., 2025). Training directly on off-policy traces can then require objective corrections (Yan et al., 2026). In-prompt methods instead keep sampled rollouts on-policy (Zhang et al., 2026b;c). Some methods increase hint strength after every response in a group is incorrect (Zhang et al., 2026b). Others reduce the fraction of the trace supplied across curriculum stages (Li et al., 2026) or use binary search to bring group accuracy into a target range (Zhang et al., 2026c). Although these methods differ in where they place hints and how they adjust hint strength, their guidance generally remains a free-form, oracle-generated trace. By contrast, the problem generator ensures the correctness of procedural guidance, while maskable values allow fine-grained control over what the guidance reveals.

# 3 Method

OPGG trains a multimodal policy with GRPO on problems drawn from a procedural generator. For each task, it controls how much of a problem's derivation is shown in the prompt. OPGG changes the guidance in the training prompt while keeping the problem distribution, objective, advantage estimator, and rollout group composition fixed. All responses in a group are sampled from the policy using the same guided prompt. Figure 2 summarizes the training loop.

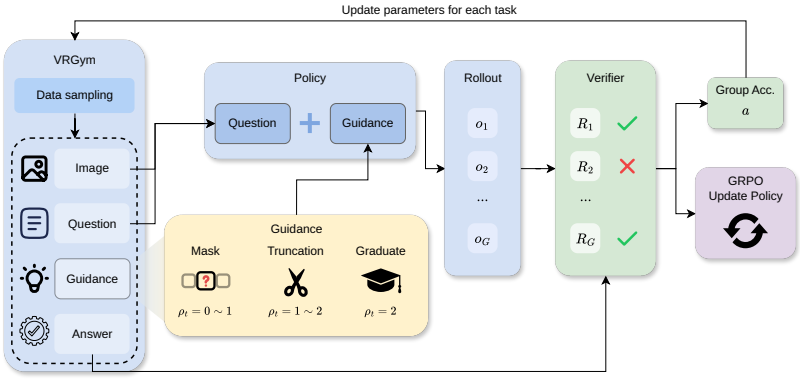



Figure 2: Overview of OPGG. VRGym supplies each problem and its verified derivation. A task-specific guidance-withdrawal budget determines how much guidance appears in the prompt. Group accuracy is used to update this budget at each training step.

## 3.1 Preliminaries

GRPO (Shao et al., 2024) replaces a learned value function with a baseline computed from the rewards of responses to the same prompt. For a prompt $q$, it samples $G$ responses (rollouts)

$\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q)$. In our RLVR setting, a rule-based verifier assigns each response a binary reward $R(o_i) \in \{0, 1\}$. The policy maximizes

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{k=1}^{|o_i|}\Big(\min\big(w_{i,k}\hat{A}_i,\ \text{clip}(w_{i,k}, 1-\epsilon, 1+\epsilon)\hat{A}_i\big) - \beta\mathcal{K}_{i,k}\Big)\right], \tag{1}$$

The expectation is over sampled prompts and rollouts. Here $w_{i,k} = \dfrac{\pi_\theta(o_{i,k} \mid q, o_{i,<k})}{\pi_{\theta_{\text{old}}}(o_{i,k} \mid q, o_{i,<k})}$ is the token-level importance ratio, and $\epsilon$ is the clipping parameter. The term $\mathcal{K}_{i,k}$ estimates the per-token KL divergence from the reference policy $\pi_{\text{ref}}$, with coefficient $\beta$. With outcome rewards, the advantage $\hat{A}_i$ is shared by every token of $o_i$:

$$\hat{A}_i = \frac{R(o_i) - \text{mean}(\{R(o_j)\}_{j=1}^{G})}{\text{std}(\{R(o_j)\}_{j=1}^{G}) + \varepsilon}. \tag{2}$$

The stabilizer $\varepsilon > 0$ keeps this expression defined when all rewards are equal.

When all $G$ rewards in a group are identical, every advantage $\hat{A}_i$ is exactly zero. The clipped surrogate term in Equation 1 then vanishes for that group, while its KL penalty may still influence the update. We call this *advantage collapse*. Let $a(q) = \frac{1}{G}\sum_i R(o_i)$ denote the group accuracy for prompt $q$. With binary rewards, advantage collapse occurs exactly when $a(q) = 0$ or $a(q) = 1$. Groups with $0 < a(q) < 1$ contain both correct and incorrect responses and have non-zero reward-derived advantages.

### 3.2 Derivations from a procedural generator

Controllable guidance requires derivations with correct intermediate steps and designated maskable values across the supported difficulty range. We generate these derivations using the procedure that computes each answer.

**Vision Reasoning Gym (VRGym)** generates multimodal reasoning problems and supplies a maskable derivation for each one. It contains 68 tasks across mathematical, logical, algorithmic, chart, and table reasoning. Each task has its own integer difficulty range. Appendix C lists all tasks, and the supplementary material provides reference examples for each task at every difficulty level.

Given a task $t$, difficulty $d$, and seed $s$, the generator samples a problem, renders its image, and computes its answer using the task's solver. VRGym supports changes in difficulty, but we keep the training difficulty distribution fixed across methods to isolate the effect of guidance. Each task defines a derivation template $T_t$ with **maskable values** $V = \{v_1, \ldots, v_K\}$. These include numbers and labels read from the image, such as a plotted value, a node's color, or a cell's coordinates. They also include computed intermediate results and the final answer.

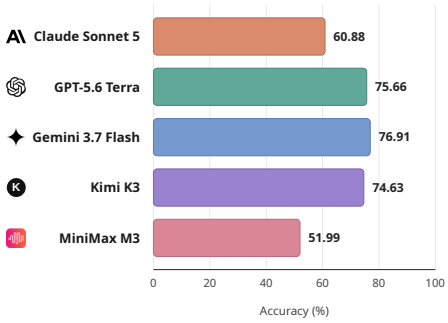



Figure 3: Frontier multimodal models on a fixed VRGym test split with reserved seeds. Prompts contain no guidance, and reasoning effort is fixed to medium for every model.

We write $T_t(V)$ for the full derivation obtained by filling the template with these values. Masking selected values before rendering the template produces partial guidance. Section 3.3 defines how much of this guidance is shown. Each generated sample is a tuple $(x, y, T_t, V)$, where $x$ contains the image and question, $y$ is the answer, $T_t$ is the derivation template, and $V$ contains its values. Samples are generated on request rather than retrieved from a fixed corpus.

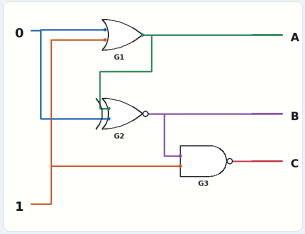
The generator computes each derivation step directly, so guidance does not require a separate oracle model. The resulting problems are not trivial: frontier multimodal models achieve between 52% and 77% accuracy on a fixed VRGym test split without any guidance (Figure 3).

**Question** `logic/logic_circuit`, difficulty 1

The image shows a combinational logic circuit using standard logic symbols: AND, OR, XOR, and NOT. A NOT is shown as the small inversion bubble at the right output end of a gate. Each left-side input wire is labeled directly with its Boolean value. The output wires on the right are labeled A, B, C. Evaluate the circuit from left to right and return the output values for (A, B, C) in that order as a tuple of 0/1 values, for example (0, 0, 0). *Answer:* (1, 0, 1).

**Full derivation** $T_t(V)$ (***bold** = values eligible for additional masking, $V \setminus A$)*

The left-side input signal values are ordered from top to bottom as I1 through I2; the same input signal may branch to multiple gate inputs. Start with these labeled inputs: I1 = **0**; I2 = **1**. Evaluate each gate only after its two input signals are known: G1: I1(**0**) **OR** I2(**1**) = 1; G2: G1(1) **XOR** I1(**0**) = 1, then **NOT** gives 0; G3: G2(0) **AND** I2(**1**) = 0, then **NOT** gives 1. Then read the right-side outputs in order: A follows G1, so A = 1; B follows G2, so B = 0; C follows G3, so C = 1. Therefore (A, B, C) = (1, 0, 1).




| $\rho_t$ | **Guidance block rendered into the prompt** |
|---|---|
| 0.5 | The left-side input signal values are ordered from top to bottom as I1 through I2; the same input signal may branch to multiple gate inputs. Start with these labeled inputs: I1 = 0; I2 = ?. Evaluate each gate only after its two input signals are known: G1: I1(0) ? I2(?) = ?; G2: G1(?) XOR I1(0) = ?, then ? gives ?; G3: G2(?) AND I2(?) = ?, then ? gives ?. Then read the right-side outputs in order: A follows G1, so A = ?; B follows G2, so B = ?; C follows G3, so C = ?. Therefore (A, B, C) = ?. |
| 1.5 | The left-side input signal values are ordered from top to bottom as I1 through I2; the same input signal may branch to multiple gate inputs. Start with these labeled inputs: I1 = ?; I2 = ?. Evaluate each gate only after its two input signals are known: G1: I1(?) ? I2(?) = ?; |
| 2.0 | *(no guidance block; the prompt is the question alone)* |

Figure 4: One generated problem with guidance masking ($\rho_t = 0.5$), truncation (1.5), and no guidance at all (2.0). For this problem, maskable values such as gate types, input bits, and computed results can be masked with ?. Appendix B shows additional budget settings.

### 3.3 A TWO-PHASE GUIDANCE SCHEDULE

Given a sample $(x, y, T_t, V)$, we place guidance below the question in the training prompt. Appendix A gives the full prompt template. A task-specific *guidance-withdrawal budget* $\rho_t \in [0, 2]$ controls two phases of guidance withdrawal. Larger values of $\rho_t$ mean less guidance. When $\rho_t$ reaches 2, the task *graduates* and receives no further guidance.

**Phase 1. Guidance masking.** Let $A \subseteq V$ contain the final answer and all computed intermediate results. We call $A$ the answer-bearing subset and always mask its values. The function $\mathrm{mask}(V, m)$ replaces every value in $A$ with `?`. It also replaces a randomly selected fraction $m$ of the remaining values with `?`.

**Phase 2. Guidance removal.** The function $\mathrm{trunc}(h, u)$ removes the last fraction $u$ of the rendered guidance $h$, cutting only at whitespace boundaries. At $u = 1$, it removes the entire guidance block, leaving the same prompt used in an unguided run. Appendix A.2 explains how the masking and removal ratios are rounded to counts of values and words.

The budget $\rho_t$ determines both ratios,

$$m_t = \min(\rho_t, 1), \qquad u_t = \max(0, \rho_t - 1), \tag{3}$$

so all values are masked at $\rho_t = 1$, and text removal begins above this point. A single budget lets the schedule move in either direction across the phase boundary (Appendix B). Even at $\rho_t = 0$, all values in $A$ are replaced with `?` in the guidance. It shows the values read from the image and the reasoning steps, but the model must still compute the intermediate results and answer. Within Phase 1, increasing the budget increases the fraction of values in $V \setminus A$ replaced with `?`. A maskable value is a designated number, label, or operation in the derivation, rather than an arbitrary span of text. Figure 4 illustrates three budget settings for one generated problem, and Appendix B illustrates five.

The update rule uses group accuracy $a$ to adjust the amount of guidance. High accuracy suggests that the model can work with less guidance. If every response is incorrect, the model may need more guidance. Groups between these cases already provide a useful learning signal. After each training

**Algorithm 1** One OPGG training step. Budgets are read when prompts are constructed and updated after responses are scored. Updated budgets apply when the corresponding tasks are next sampled.

**Require:** policy $\pi_\theta$; one guidance-withdrawal budget $\rho_t$ per task, kept from the previous step ($\rho_t = 0$ at step 0)

$\theta_{\text{old}} \leftarrow \theta,\ \mathcal{B} \leftarrow \emptyset$ ▷ $\mathcal{B}$ collects this step's groups
**for** each prompt of the step, with sample $(x, y, T_t, V)$ drawn from VRGym for task $t$ **do**
  $m_t \leftarrow \min(\rho_t, 1),\ u_t \leftarrow \max(0, \rho_t - 1)$ ▷ Eq. 3
  $h \leftarrow \text{trunc}\big(T_t(\text{mask}(V, m_t)),\ u_t\big)$ ▷ Phase 1, then Phase 2
  $q \leftarrow$ question $x$ followed by the guidance block $h$, dropped when $h$ is empty
  sample $\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q)$ and score each against $y$ for $R(o_i) \in \{0, 1\}$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \big\{(t, q, \{o_i\}_{i=1}^{G})\big\}$
**end for**
**for** each group $(t, q, \{o_i\}_{i=1}^{G}) \in \mathcal{B}$ **do** ▷ one update per group
  $a \leftarrow \frac{1}{G}\sum_{i=1}^{G} R(o_i)$ ▷ group accuracy
  **if** $\rho_t < 2$ **then**
    $\rho_t \leftarrow \text{clip}_{[0,2]}\big(\rho_t + \Delta(a)\big)$ ▷ Eq. 4
  **end if**
**end for**
update $\theta$ on $\mathcal{B}$ with the GRPO objective ▷ Eq. 1

step, we compute the accuracy of each prompt's rollout group. For every group belonging to a task $t$ that has not graduated ($\rho_t < 2$), we apply

$$\rho_t \leftarrow \text{clip}_{[0,2]}\big(\rho_t + \Delta(a)\big), \qquad \Delta(a) = \begin{cases} +\delta^+ & \text{if } a \geq \alpha \quad \text{(guidance is excessive)}, \\ -\delta^- & \text{if } a = 0 \quad \text{(guidance is insufficient)}, \\ 0 & \text{if } 0 < a < \alpha \quad \text{(the group is informative)}, \end{cases} \tag{4}$$

where $\text{clip}_{[0,2]}$ bounds the updated budget to $[0, 2]$. We use $\alpha = 0.8$, $\delta^+ = 0.05$, and $\delta^- = 0.01$. The budget changes only near the two ends of the accuracy range. Withdrawal begins at $a \geq \alpha$, before the group necessarily reaches $a = 1$ and loses its advantage signal. The step sizes favor gradual withdrawal over the course of training. A withdrawal update is five times as large as a restoration update, so one all-incorrect group does not undo the progress from one high-accuracy group. Each group contributes one budget update, using the rollouts already sampled by GRPO. The schedule therefore requires no additional inference.

VRGym generates fresh instances rather than repeatedly using the same prompts, so we maintain one persistent guidance-withdrawal budget per task. The budget $\rho_t$ carries information across instances of task $t$ and across training steps. Algorithm 1 summarizes one training step.

### 3.4 Implementation

We train Qwen2.5-VL-7B-Instruct (Bai et al., 2025b) and Qwen3-VL-8B-Instruct (Bai et al., 2025a) with GRPO on the mathematical, logical, algorithmic, and chart families, which contain 51 tasks. We hold out table reasoning from training. Each run maintains 51 independent guidance-withdrawal budgets $\rho_t$, one per task. Only groups from task $t$ update its budget before graduation, and the budget is used again when that task is next sampled. VRGym uses rule-based binary rewards $R(o_i) \in \{0, 1\}$. Appendix A provides the hyperparameters and implementation details.

## 4 Experiments

### 4.1 Setup

**Training.** Each backbone is trained on VRGym for 500 steps using the common settings in Table 6. Within each backbone, trained methods differ in how guidance is supplied and handled during training. To generate each training example, we sample uniformly from the 51 tasks, then draw an integer difficulty using fixed Gaussian weights centered at $\mu = 2.0$ with a standard-deviation

Table 1: Accuracy (avg@8, %) on the VRGym test split and six external benchmarks. All other methods train on the same VRGym distribution for 500 steps. Evaluation prompts contain no guidance. Within each backbone, the best result in each column is bold and the second-best is underlined. *Avg.* is the macro-average over all seven datasets.

| | | Related domain | | | | General domain | | Overall |
|---|---|---|---|---|---|---|---|---|
| **Method** | **VRGym** | **MathVista** | **MathVision** | **MathVerse** | **DynaMath** | **MMMU** | **MMMU-Pro** | **Avg.** |
| *Qwen2.5-VL-7B-Instruct* | | | | | | | | |
| Base | 14.15 | 68.44 | 24.88 | 47.32 | 49.74 | 53.04 | 36.38 | 41.99 |
| GRPO (Shao et al., 2024) | 21.23 | 70.17 | 24.71 | 48.06 | <u>50.87</u> | 53.94 | 35.50 | 43.50 |
| LUFFY (Yan et al., 2026) | 21.67 | 70.15 | 25.70 | 46.41 | 47.68 | 52.95 | 36.73 | 43.04 |
| Hint-GRPO (Huang et al., 2025) | 18.74 | 70.44 | <u>25.74</u> | 47.27 | 50.58 | **54.45** | **38.63** | 43.69 |
| Scaf-GRPO (Zhang et al., 2026b) | <u>23.05</u> | **71.16** | 24.63 | <u>48.98</u> | 49.73 | 53.73 | 36.56 | <u>43.98</u> |
| **OPGG (ours)** | **23.66** | <u>71.14</u> | **27.96** | **49.28** | **52.05** | <u>54.35</u> | <u>38.05</u> | **45.21** |
| *Qwen3-VL-8B-Instruct* | | | | | | | | |
| Base | 27.19 | 77.22 | 45.27 | <u>66.65</u> | 67.75 | 67.77 | 53.28 | 57.88 |
| GRPO (Shao et al., 2024) | 33.35 | 75.92 | 44.41 | 65.10 | 67.89 | 66.83 | 52.09 | 57.94 |
| LUFFY (Yan et al., 2026) | 29.42 | 75.55 | 44.28 | 63.79 | 64.20 | 66.73 | 50.72 | 56.38 |
| Hint-GRPO (Huang et al., 2025) | 33.23 | 76.01 | 45.97 | 66.60 | <u>68.94</u> | <u>68.57</u> | **53.46** | <u>58.97</u> |
| Scaf-GRPO (Zhang et al., 2026b) | <u>33.58</u> | <u>77.93</u> | <u>47.12</u> | 65.35 | 67.63 | 66.14 | 53.31 | 58.72 |
| **OPGG (ours)** | **33.71** | **78.09** | **47.49** | **67.99** | **70.19** | **68.94** | <u>53.37</u> | **59.97** |

divisor of 5.0. Appendix A.1 defines the training difficulty distribution and the test split's difficulty allocation.

**Baselines.** We compare OPGG with the initial instruction-tuned checkpoint, GRPO trained on the same data, and three hint-based RLVR methods. We reproduce LUFFY (Yan et al., 2026), Hint-GRPO (Huang et al., 2025), and Scaf-GRPO (Zhang et al., 2026b) using their official repositories. Their original guidance corpora do not cover VRGym, so we generate new hints for these experiments. LUFFY and Scaf-GRPO were developed for text-only mathematics and used an oracle that does not accept images. VRGym therefore requires a multimodal oracle.

We use Gemini 3.7 Flash to generate hints offline for all three baselines, adapting each method's hint-generation settings and prompts to VRGym. This model has the highest accuracy in our VRGym evaluation (Figure 3). On the benchmarks reported in Appendix A.4, it also scores higher than the oracle models used by these baselines. We otherwise follow each baseline's published training settings. OPGG uses the procedural derivation supplied with each generated VRGym problem.

**Evaluation.** We report avg@8 accuracy on the VRGym test split generated from reserved seeds and six external benchmarks. The related-domain set comprises MathVista (Lu et al., 2024), MathVision (Wang et al., 2024), MathVerse (Zhang et al., 2024), and DynaMath (Zou et al., 2025). The general-domain set comprises MMMU (Yue et al., 2024) and MMMU-Pro (Yue et al., 2025). For each question, we sample eight responses at temperature 0.7 and top-$p$ 0.95. We compute avg@8 by averaging correctness across these responses and then across questions.

### 4.2 Main results

**VRGym test split.** The VRGym test split contains 1,360 questions, with 20 per task. Each task contributes five questions at each of difficulties 1–4. Evaluation prompts contain no guidance for any method. All RLVR methods improve substantially over the initial checkpoint on this split. On Qwen2.5-VL-7B, OPGG reaches 23.66%, a 9.51-point gain over the base model and a 2.43-point gain over vanilla GRPO. On Qwen3-VL-8B, it reaches 33.71%, improving over the base by 6.52 points and over GRPO by 0.36 points.

**Comparison with GRPO.** The controlled comparison with GRPO measures the effect of OPGG training because the paired runs use the same backbone, VRGym problems, training difficulty distribution, and optimization settings. OPGG exceeds GRPO on all 12 combinations of external benchmark and backbone. When VRGym is included, its macro-average over all seven reported datasets is 1.71 points higher than GRPO on Qwen2.5-VL-7B and 2.03 points higher on Qwen3-VL-8B.

Table 2: Step time relative to GRPO on Qwen2.5-VL-7B.

| Method | Time |
|---|---|
| GRPO | 1.00× |
| LUFFY | 1.06× |
| Hint-GRPO | 1.15× |
| Scaf-GRPO | 2.99× |
| OPGG | 1.17× |

Table 3: Groups with non-zero advantage on Qwen2.5-VL-7B, out of 16,000.

| Method | Groups | Share |
|---|---|---|
| GRPO | 5,281 | 33.0% |
| OPGG | 7,851 | **49.1%** |
| Δ | +2,570 | +16.1 pp |

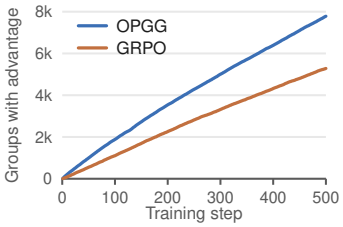



Figure 5: Cumulative groups with non-zero advantage on Qwen2.5-VL-7B.

**Comparison with other guidance methods.** OPGG achieves the highest seven-dataset average on both backbones, exceeding the strongest prior hint-based baseline by 1.23 points on Qwen2.5-VL-7B and 1.00 point on Qwen3-VL-8B. It also ranks first on four of seven datasets with Qwen2.5 and six of seven with Qwen3.

**Training speed.** Table 2 reports average runtime per training step relative to GRPO. OPGG takes 1.17× as long per step because its guidance lengthens the input context processed during training. Scaf-GRPO takes 2.99× as long. When all $G$ initial responses are incorrect, Scaf-GRPO retries the problem with stronger hints and additional rollouts until a response is correct or the hint limit is reached.

### 4.3 Groups with non-zero advantage

Here, we isolate reward-derived policy-gradient signals by considering groups whose outcome rewards vary across rollouts. Following Equation 2, we count a group when $\mathrm{std}(\{R(o_i)\}_{i=1}^{G}) > 0$ and at least one valid response token has non-zero advantage. The count uses raw outcome scores, excluding variation introduced only by KL penalties. OPGG yields 7,851 groups with non-zero advantage (49.1%), compared with 5,281 (33.0%) for GRPO, a gain of 2,570 groups or 16.1 percentage points (Table 3). The larger share means more sampled groups contribute a reward-derived policy-gradient signal. Figure 5 shows the gap widening throughout training.

### 4.4 Ablation study

We test the contribution of each guidance phase and the update rule. The *removal only* variant fixes $m_t = 0$ and adjusts only the truncation ratio $u_t$. The *mask only* variant fixes $u_t = 0$ and limits the budget to $\rho_t \leq 1$, so every training prompt retains a masked derivation. Both variants always mask the answer-bearing subset $A$. Two other variants modify the default update rule ($\delta^+ = 0.05$, $\delta^- = 0.01$, $\alpha = 0.8$). One uses equal step sizes, $\delta^+ = \delta^- = 0.05$, and the other sets $\alpha = 1.0$. All other settings match default OPGG.

Default OPGG achieves the highest average (Table 4). Removal only scores 0.65 points lower, and mask only scores 0.79 points lower, supporting the use of both phases. Despite retaining guidance throughout training, the mask-only variant exceeds unguided GRPO by 0.92 points when evaluated without guidance. It also produces more groups with non-zero advantage than GRPO (Appendix D). Thus, masking can provide useful training guidance even without a later removal phase.

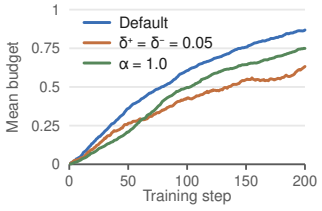



Figure 6: Guidance-withdrawal budget on Qwen2.5-VL-7B.

Table 4: Seven-dataset macro-average (%) by ablation.

| Variant | Avg. | Δ |
|---|---|---|
| **Default OPGG** | **45.21** | — |
| Removal only | 44.56 | −0.65 |
| Mask only | 44.42 | −0.79 |
| $\delta^+ = \delta^- = 0.05$ | 44.63 | −0.58 |
| $\alpha = 1.0$ | 44.46 | −0.75 |

The two update-rule variants also score below the default and withdraw less guidance during training (Figure 6). With equal step sizes, a single all-incorrect group can undo the progress of a high-accuracy group. With $\alpha = 1.0$, withdrawal waits until every response is correct. These results are consistent with a benefit from the default withdrawal rule. Appendix D gives per-dataset results and group counts for every variant.

### 4.5 Procedural versus oracle guidance

To isolate guidance source, we disable the schedule and hold all other settings fixed. Procedural guidance uses the VRGym derivation at $\rho_t = 0$, while oracle guidance comes from Gemini 3.7 Flash. Neither reveals the correct answer. Because fixed guidance produces more zero-advantage groups over time, we stop both runs at step 200.

Procedural guidance reaches a seven-dataset average of $43.57\%$, compared with $42.81\%$ for oracle guidance (Table 5). Appendix E reports per-dataset results.

Table 5: Seven-dataset average (%) after 200 fixed-guidance steps on Qwen2.5-VL-7B.

| Guidance | Avg. |
|---|---|
| **Procedural** | **43.57** |
| Oracle | 42.81 |

## 5 Limitations

**Coverage of procedural data.** Limited coverage is a common constraint of procedural reasoning environments (Stojanovski et al., 2026; Yang et al., 2026; Meng et al., 2026), as controllable generation and exact verification depend on predefined tasks. Our setup shares this limitation: OPGG is trained entirely on VRGym, which must supply a correct derivation with maskable values for every training instance. Although its 68 tasks span five reasoning families, they cover only a subset of the visual formats, knowledge, and reasoning found in real-world multimodal data. Our results therefore validate procedural guidance within this setting, but do not establish comprehensive coverage of real-world multimodal reasoning.

**Uneven task calibration.** The fixed training difficulty distribution does not place every VRGym task at an equally useful level for RLVR. For a small number of difficult tasks, the guidance-withdrawal budget $\rho_t$ oscillates around an intermediate value and does not reach 2, indicating that the schedule has not withdrawn guidance completely for these tasks. Conversely, some easy tasks are already solvable without guidance and offer little opportunity for guided progression. OPGG adapts the amount of guidance but does not by itself correct a task whose difficulty range is poorly matched to the policy.

## 6 Conclusion and Future Work

We introduced procedural guidance for multimodal RLVR and implemented it in VRGym and OPGG. VRGym generates correct derivations with maskable values for every problem at each supported difficulty. OPGG uses group accuracy to adjust a persistent guidance-withdrawal budget for each task, gradually masking values and removing derivation text without changing the GRPO objective. Across seven evaluation datasets, OPGG achieves higher average accuracy than GRPO trained on the same data and all evaluated hint-based baselines on both backbones.

Future work has two directions. First, extend procedural guidance to more environments. Reasoning Gym (Stojanovski et al., 2026) and TRON (Yang et al., 2026) already generate problems and answers programmatically and could be extended to provide derivations. More broadly, environments with verifiable actions and states could provide structured guidance for agentic tasks, extending the approach beyond visual reasoning.

Second, use procedural structure to design richer rewards. OPGG currently uses derivations only as prompt guidance. Verifying intermediate values, states, or actions could support graded, non-binary rewards for partially correct rollouts, rather than rewarding only the final answer. Such verifiers should accept multiple valid solution paths instead of requiring a rollout to match a single reference derivation.

### Reproducibility Statement

The supplementary material contains the complete codebase, including the VRGym generation framework and OPGG training script, together with reference examples for every task and difficulty level; the code will be made public upon acceptance. Sections 3 and 4 describe the method and experimental protocol, with implementation details in Appendix A and the full VRGym coverage in Appendix C.

### AI Use Statement

Generative AI tools were used extensively to translate our draft into English and to polish and proofread the text. They also helped identify gaps in the experimental evaluation, search for and summarize prior work, interpret and compare methods, reproduce prior research, and write the research code, including VRGym and OPGG. The core ideas, the original draft, and the structure of the paper are the authors' own. The authors reviewed and revised all AI-assisted text, checked literature descriptions and comparisons against the cited papers, evaluated every suggested experiment, and tested all AI-assisted code. The authors take full responsibility for the final content of this work, including all AI-assisted text, claims, and artifacts.

## References


Md Tanvirul Alam. Trace: A taxonomy-guided environment for multidomain visual reasoning. *arXiv preprint arXiv:2607.19790*, 2026.

Artificial Analysis. LLM Leaderboard & Benchmarking Data. `https://artificialanalysis.ai`, 2026. Accessed: 2026-08-28.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yongxin Guo, Wenbo Deng, Zhenglin Cheng, and Xiaoying Tang. G2rpo-a: Guided group relative policy optimization with adaptive guidance. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4525–4539, 2026.

Xixiang He, Qiyao Sun, Ao Cheng, Xingming Li, Xuanyu Ji, Hailun Lu, Runke Huang, and Qingyong Hu. Advantage collapse in group relative policy optimization: Diagnosis and mitigation. In *International Conference on Machine Learning*, 2026.

Qihan Huang, Weilong Dai, Jinlong Liu, Wanggui He, Hao Jiang, Mingli Song, Jingyuan Chen, Chang Yao, and Jie Song. Boosting mllm reasoning with text-debiased hint-grpo. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4848–4857. IEEE, 2025.

Wenxuan Huang, Bohan Jia, Shaosheng Cao, Zheyu Ye, Zhe Xu, Yao Hu, Shaohui Lin, et al. Vision-r1: Incentivizing reasoning capability in multimodal large language models. In *International Conference on Learning Representations*, volume 2026, pp. 63794–63812, 2026.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL `https://github.com/huggingface/open-r1`.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Thanh-Long V. Le, Myeongho Jeon, Kim Vu, Viet Lai, and Eunho Yang. No prompt left behind: Exploiting zero-variance prompts in llm reinforcement learning via entropy-guided advantage shaping. In *The Fourteenth International Conference on Learning Representations*, pp. 121956–121982, 2026.

Jiazheng Li, Hongzhou Lin, Hong Lu, Kaiyue Wen, Zaiwen Yang, Jiaxuan Gao, Yi Wu, and Jingzhao Zhang. Questa: Expanding reasoning capacity in llms via question augmentation. In *International Conference on Learning Representations*, volume 2026, pp. 144191–144215, 2026.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *Conference on Language Modeling (COLM)*, 2025a.

Ziru Liu, Cheng Gong, Xinyu Fu, Yaofang Liu, Ran Chen, Shoubo Hu, Suiyun Zhang, Rui Liu, Qingfu Zhang, and Dandan Tu. Ghpo: Adaptive guidance for stable and efficient llm reinforcement learning. *arXiv preprint arXiv:2507.10628*, 2025b.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, volume 2024, pp. 23439–23554, 2024.

Fanqing Meng, Lingxiao Du, Jiawei Gu, Jiaqi Liao, Linjie Li, Zijian Wu, Xiangyan Liu, Ziqi Zhao, Mengkang Hu, Zichen Liu, Jiaheng Zhang, and Michael Qizhe Shieh. Gym-v: A unified vision environment system for agentic vision research. *arXiv preprint arXiv:2603.15432*, 2026.

Yuxiao Qu, Amrith Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. Pope: Learning to reason on hard problems via privileged on-policy exploration. *arXiv preprint arXiv:2601.18779*, 2026.

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.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In *Advances in Neural Information Processing Systems*, volume 38, 2026.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95095–95169, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in neural information processing systems*, volume 35, pp. 24824–24837, 2022.

Guowei Xu, Peng Jin, Ziang Wu, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. Llava-cot: Let vision language models reason step-by-step. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2087–2098. IEEE, 2025.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. In *Advances in Neural Information Processing Systems*, volume 38, pp. 117157–117186, 2026.

Tianze Yang, Yucheng Shi, Ruitong Sun, Jingyuan Huang, Ninghao Liu, and Jin Sun. Tron: Targeted rule-verifiable online environments for visual reasoning rl. *arXiv preprint arXiv:2606.01599*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244, 2026.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9556–9567, 2024.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, et al. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15134–15186, 2025.

Feng Zhang, Zezhong Tan, Xinhong Ma, Ziqiang Dong, Xi Leng, Jianfei Zhao, Xin Sun, and Yang Yang. Adhint: Adaptive hints with difficulty priors for reinforcement learning. *arXiv preprint arXiv:2512.13095*, 2025.

Kaiyi Zhang, Ang Lv, Jinpeng Li, Yongbo Wang, Feng Wang, Haoyuan Hu, and Rui Yan. Stephint: Multi-level stepwise hints enhance reinforcement learning to reason. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 37846–37864, 2026a.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pp. 169–186. Springer, 2024.

Xichen Zhang, Sitong Wu, Yinghao Zhu, Haoru Tan, Shaozuo Yu, Ziyi He, and Jiaya Jia. Scaf-grpo: Scaffolded group relative policy optimization for enhancing llm reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 131946–131974, 2026b.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. Bread: Branched rollouts from expert anchors bridge sft & rl for reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 96726–96752, 2026c.

Chengke Zou, Xingang Guo, Rui Yang, Junyu Zhang, Bin Hu, and Huan Zhang. Dynamath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models. In *International Conference on Learning Representations*, volume 2025, pp. 48337–48383, 2025.

# A Implementation Details

## A.1 Difficulty sampling

For each task $t$, let $D_t$ be its supported integer difficulty levels and $L_t = |D_t|$. We center the Gaussian weights at $\mu = 2.0$ and set their width parameter to

$$\sigma_t = \max\left(1, \frac{L_t}{5.0}\right), \tag{5}$$

where $5.0$ is the standard-deviation divisor reported in Table 6. Each level $d \in D_t$ receives the normalized weight

$$p_t(d) = \frac{\exp\left[-\frac{1}{2}\left(\frac{d-\mu}{\sigma_t}\right)^2\right]}{\sum_{d' \in D_t} \exp\left[-\frac{1}{2}\left(\frac{d'-\mu}{\sigma_t}\right)^2\right]}. \tag{6}$$

During training, we draw a difficulty from $p_t$ for each sample. The fixed test split instead contains five questions at each of difficulties 1–4 for every task.

Table 6: Training configuration. The lower block is specific to OPGG.

| Setting | Value | Symbol |
|---|---|---|
| *Common training settings* | | |
| Policy model | Qwen2.5-VL-7B, Qwen3-VL-8B (Instruct) | |
| Advantage estimator | GRPO | |
| GPUs | $4\times$ NVIDIA H200 | |
| Prompts per step | 32 | |
| Rollouts per prompt | 8 | $G$ |
| Rollout temperature / top-$p$ | 1.0 / 1.0 (no top-$k$) | |
| Mini-batch size | 8 | |
| Optimizer | Adam | |
| Actor learning rate | 1e-6 | |
| Learning-rate schedule | Constant | |
| Weight decay | 0.01 (constant schedule) | |
| Clip ratio | 0.2 | $\epsilon$ |
| KL loss coefficient | 0.01 (low-variance) | $\beta$ |
| Entropy coefficient | 0 | |
| Max prompt / response length | 32768 / 8192 tokens | |
| Training steps | 500 | |
| Difficulty | Discrete Gaussian weights ($\mu = 2.0$, divisor $= 5.0$) | |
| *OPGG schedule* | | |
| Withdrawal threshold | 0.8 | $\alpha$ ($a \geq \alpha$) |
| Withdrawal step | 0.05 | $\delta^+$ |
| Guidance restoration step | 0.01 | $\delta^-$ |
| Initial mask ratio | 0.0 | $m_t$ at step 0 |
| Initial removal ratio | 0.0 | $u_t$ at step 0 |

### A.2 Turning a ratio into a count

Both phases act on a finite pool, so each ratio is first rounded to a whole number of items. For a pool of $n$ items and a ratio $r$, both use the same half-up rule

$$k(n, r) = \left\lfloor nr + \tfrac{1}{2} \right\rfloor . \tag{7}$$

**Phase 1.** Let $O = V \setminus A$ be the pool of values eligible for random masking. The function $\mathrm{mask}(V, m)$ replaces all values in $A$ and a subset $S \subseteq O$ of size $k(|O|, m)$ with `?`. We sample $S$ uniformly without replacement. This gives a fixed masked count for a given ratio, unlike independent masking with probability $m$. Mask selection uses a random stream seeded from the sample's seed and separate from the stream used to generate the figure. A fixed seed therefore reproduces the same mask, while a new instance receives a new draw of $S$.

Because $k(|O|, m)$ is an integer, nearby budgets can produce the same masked count. For example, with $|O| = 4$, the count increases when $m$ reaches $0.125$, $0.375$, $0.625$, and $0.875$. Budget updates within these intervals do not change the number of masked values.

**Phase 2.** The function $\mathrm{trunc}(h, u)$ treats each maximal sequence of non-whitespace characters as a word, giving $w_1, \ldots, w_N$. It removes the final $k(N, u)$ words and keeps the original spacing in the remaining prefix. If $u \geq 1$ or no words remain, it removes the entire guidance block. Words are never split, although a cut may fall within a reasoning step or leave trailing punctuation. This rule applies to all $51$ training tasks without a separate parser for each template. After all maskable values have been masked, Phase 2 progressively removes the remaining reasoning structure.

### A.3 GRPO configuration

**Prompt template.** Every training prompt combines a fixed system message with a user query placed after the image placeholders. We use the following template.

```
## System prompt
Please reason step by step, and put your final answer within \boxed{}.
## User prompt
Question: {question}
Hint for the question: {hint}
```

The {hint} field contains guidance rendered from the derivation using the task's current budget $\rho_t$. At $\rho_t = 2$, the guidance line is omitted, making the prompt identical to an unguided prompt.

The trainer stores task-specific guidance-withdrawal budgets in shared-memory tensors. Dataloader workers read these tensors when constructing prompts, so they use the current budget for the selected task. All budgets are saved with the policy in every checkpoint. Validation prompts never contain guidance, so guided and unguided runs use the same evaluation format at every step.

Table 6 lists the training configuration. The upper block lists common training settings, and the lower block contains the OPGG schedule parameters from Section 3.3.

### A.4 Oracle model for the baselines

The hint-based baselines use oracle guidance from their original training corpora. LUFFY (Yan et al., 2026) and Scaf-GRPO (Zhang et al., 2026b) were developed for text-only mathematics and both draw on DeepSeek-R1 (Guo et al., 2025) traces: LUFFY takes its off-policy trajectories directly from the OpenR1-Math-220k dataset (Hugging Face, 2025), whose traces were generated by DeepSeek-R1, while Scaf-GRPO prompts DeepSeek-R1 to write tiered hints from ground-truth solutions. Hint-GRPO (Huang et al., 2025) is multimodal and uses LLaVA-CoT (Xu et al., 2025) reasoning traces divided into structured steps by GPT-4o (Hurst et al., 2024). These corpora do not cover newly generated VRGym problems, so we must generate new guidance to run the baselines on VRGym.

We use the same multimodal oracle, Gemini 3.7 Flash, for all three methods. Only the oracle model is replaced, and each method otherwise follows its original hint-generation procedure. For LUFFY, Gemini writes a complete reasoning trace for each problem, and we keep only traces whose final answer the verifier accepts, as LUFFY does when it discards DeepSeek-R1 traces that Math-Verify judges incorrect. For Scaf-GRPO, Gemini writes the tiered hints from the problem and its correct answer. For Hint-GRPO, Gemini also receives the correct answer and writes a step-by-step reasoning trace, which replaces the LLaVA-CoT trace. Scaf-GRPO originally conditions on a ground-truth solution. We supply the answer rather than the VRGym derivation so that no baseline draws on procedural guidance. The baseline hints therefore keep the correctness safeguards of their original pipelines, and OPGG is compared against oracle guidance that already includes them. The choice of oracle may still affect the comparison, so Table 7 compares its reported benchmark scores with those of the oracle models used by the baselines.

Table 7: Third-party scores for the oracle we use and the two it replaces, as reported by Artificial Analysis (2026). HLE is Humanity's Last Exam and AA-O is AA-Omniscience accuracy. DeepSeek-R1 takes no image input, so it has no multimodal score. Gemini 3.7 Flash is scored at medium reasoning effort, the setting we use throughout.

| | | Reasoning | | Multimodality |
|---|---|---|---|---|
| **Oracle** | **Method using its guidance** | **HLE** | **AA-O** | **MMMU-Pro** |
| GPT-4o | Hint-GRPO, via LLaVA-CoT | 2 | 24 | 56 |
| DeepSeek-R1 | LUFFY, Scaf-GRPO | 9 | 31 | — |
| **Gemini 3.7 Flash** | **all three, in our reproductions** | **39** | **54** | **85** |

Gemini 3.7 Flash has higher scores on both reported text reasoning measures. On MMMU-Pro, it scores 85, compared with 56 for GPT-4o. These comparisons support the choice of a capable replacement oracle, although they do not directly measure the quality of the oracle guidance generated for the VRGym baselines.

## B EXAMPLES ACROSS THE GUIDANCE-WITHDRAWAL BUDGET

Figure 4 shows one generated problem at three budget settings. Table 8 shows the same logic-circuit problem at five settings, using guidance rendered by the generator at difficulty 1. The budget lies in $[0, 2]$ and changes by $\delta^+ = 0.05$ or $\delta^- = 0.01$ per update until graduation. The numbers of masked values and removed words follow the rounding rule in Appendix A.2.

The ratios $m_t = \min(\rho_t, 1)$ and $u_t = \max(0, \rho_t - 1)$ ensure that all values are masked at $\rho_t = 1$, before truncation begins. Each **?** replaces a maskable value in the derivation. The answer-bearing subset $A$ is masked at every budget setting, including $\rho_t = 0$.

A single budget allows updates to cross the phase boundary in either direction. For example, a withdrawal update moves $\rho_t = 0.98$ to $1.03$. All values are then masked, and the truncation ratio becomes $0.03$. A restoration update moves $\rho_t = 1.00$ to $0.99$, reducing the masking ratio. Whether this reveals a value depends on the pool size and rounding rule. Using one budget avoids the need for a separate transition rule between masking and truncation.

Table 8: Guidance for the problem in Figure 4 at five budget settings. Selected input bits and gate types are masked with **?**. Truncation then shortens the fully masked derivation from the end.

| $\rho_t$ | **Guidance block rendered into the prompt** |
|---|---|
| 0.0 | The left-side input signal values are ordered from top to bottom as I1 through I2; the same input signal may branch to multiple gate inputs. Start with these labeled inputs: I1 = 0; I2 = 1. Evaluate each gate only after its two input signals are known: G1: I1(0) OR I2(1) = **?**; G2: G1(**?**) XOR I1(0) = **?**, then NOT gives **?**; G3: G2(**?**) AND I2(1) = **?**, then NOT gives **?**. Then read the right-side outputs in order: A follows G1, so A = **?**; B follows G2, so B = **?**; C follows G3, so C = **?**. Therefore (A, B, C) = **?**. |
| 0.5 | The left-side input signal values are ordered from top to bottom as I1 through I2; the same input signal may branch to multiple gate inputs. Start with these labeled inputs: I1 = 0; I2 = **?**. Evaluate each gate only after its two input signals are known: G1: I1(0) **?** I2(**?**) = **?**; G2: G1(**?**) XOR I1(0) = **?**, then **?** gives **?**; G3: G2(**?**) AND I2(**?**) = **?**, then **?** gives **?**. Then read the right-side outputs in order: A follows G1, so A = **?**; B follows G2, so B = **?**; C follows G3, so C = **?**. Therefore (A, B, C) = **?**. |
| 1.0 | The left-side input signal values are ordered from top to bottom as I1 through I2; the same input signal may branch to multiple gate inputs. Start with these labeled inputs: I1 = **?**; I2 = **?**. Evaluate each gate only after its two input signals are known: G1: I1(**?**) **?** I2(**?**) = **?**; G2: G1(**?**) **?** I1(**?**) = **?**, then **?** gives **?**; G3: G2(**?**) **?** I2(**?**) = **?**, then **?** gives **?**. Then read the right-side outputs in order: A follows G1, so A = **?**; B follows G2, so B = **?**; C follows G3, so C = **?**. Therefore (A, B, C) = **?**. |
| 1.5 | The left-side input signal values are ordered from top to bottom as I1 through I2; the same input signal may branch to multiple gate inputs. Start with these labeled inputs: I1 = **?**; I2 = **?**. Evaluate each gate only after its two input signals are known: G1: I1(**?**) **?** I2(**?**) = **?**; |
| 2.0 | *(no guidance block; the prompt is the question alone, exactly what an unguided run and validation both use)* |

## C VISION REASONING GYM

VRGym is a procedural generator for controlled multimodal reasoning, implemented by coding agents such as Claude Code and Codex, with human verification. A task program first samples a latent problem specification, renders one or more images and a natural-language question from that specification, and computes the answer by executing the corresponding solver. The same execution exposes the maskable values used to fill the derivation template described in Section 3.2. Consequently, the answer, verifier target, and guidance are derived from the same underlying state rather than recovered from the rendered image by a separate model. Each sample also records its task, difficulty, and task-specific metadata, which makes generation and verification reproducible from the sampled specification.

Table 9 supplements Figure 3 with per-task results. We report pass@1 as the accuracy over the 20 examples of each task in the same fixed test split.

Table 9: Per-task pass@1 (%) for the five frontier multimodal models in Figure 3. GPT-5.6 denotes GPT-5.6 Terra, MiniMax denotes MiniMax M3, Sonnet 5 denotes Claude Sonnet 5, and Gemini denotes Gemini 3.7 Flash. The final row is the equal-weight macro-average over all 68 tasks, matching Figure 3.

| Task | GPT-5.6 | Kimi K3 | MiniMax | Sonnet 5 | Gemini |
|---|---|---|---|---|---|
| **Algorithm** | | | | | |
| maze | 70.0 | 35.0 | 5.0 | 40.0 | 80.0 |
| 3d_maze | 65.0 | 15.0 | 5.0 | 15.0 | 50.0 |
| graph_shortest_path | 45.0 | 30.0 | 15.0 | 15.0 | 50.0 |
| graph_longest_path | 50.0 | 45.0 | 30.0 | 45.0 | 90.0 |
| topological_sort | 95.0 | 95.0 | 85.0 | 100.0 | 100.0 |
| convex_hull | 100.0 | 95.0 | 65.0 | 90.0 | 100.0 |
| 3d_convex_hull | 60.0 | 10.0 | 0.0 | 25.0 | 30.0 |
| traveling_salesman_problem | 75.0 | 60.0 | 15.0 | 60.0 | 70.0 |
| rb_tree | 100.0 | 95.0 | 70.0 | 85.0 | 100.0 |
| maximum_flow | 55.0 | 45.0 | 35.0 | 30.0 | 45.0 |
| **Chart QA** | | | | | |
| price_history | 95.0 | 85.0 | 45.0 | 40.0 | 100.0 |
| counterintuitive_reasoning | 85.0 | 95.0 | 70.0 | 75.0 | 80.0 |
| value_lookup | 90.0 | 75.0 | 75.0 | 65.0 | 90.0 |
| max_duplicate_count | 90.0 | 85.0 | 75.0 | 70.0 | 85.0 |
| extreme_category | 45.0 | 95.0 | 70.0 | 70.0 | 95.0 |
| series_calculation | 90.0 | 65.0 | 90.0 | 75.0 | 85.0 |
| difference | 85.0 | 70.0 | 85.0 | 75.0 | 80.0 |
| summation | 90.0 | 70.0 | 70.0 | 70.0 | 80.0 |
| largest_change | 100.0 | 80.0 | 85.0 | 85.0 | 90.0 |
| distribution_statistic | 75.0 | 75.0 | 75.0 | 75.0 | 85.0 |
| continuous_distribution_statistic | 90.0 | 85.0 | 75.0 | 60.0 | 85.0 |
| matrix_extreme | 90.0 | 80.0 | 80.0 | 70.0 | 90.0 |
| **Logic** | | | | | |
| hanoi | 90.0 | 100.0 | 75.0 | 95.0 | 100.0 |
| ladder | 85.0 | 85.0 | 10.0 | 70.0 | 95.0 |
| 3d_ladder | 50.0 | 45.0 | 5.0 | 20.0 | 45.0 |
| 3d_plates_recognition | 40.0 | 55.0 | 25.0 | 20.0 | 50.0 |
| rectangle_counting | 25.0 | 40.0 | 25.0 | 30.0 | 45.0 |
| cube_recognition | 85.0 | 95.0 | 10.0 | 40.0 | 50.0 |
| brick_counting | 40.0 | 20.0 | 0.0 | 10.0 | 35.0 |
| length_counting | 95.0 | 100.0 | 25.0 | 95.0 | 50.0 |
| intersection_counting | 20.0 | 20.0 | 5.0 | 5.0 | 50.0 |
| color_mixing | 100.0 | 95.0 | 65.0 | 100.0 | 80.0 |
| logic_circuit | 95.0 | 100.0 | 60.0 | 75.0 | 95.0 |
| finite_state_machine | 100.0 | 100.0 | 95.0 | 100.0 | 100.0 |
| square_recognition | 95.0 | 95.0 | 60.0 | 100.0 | 85.0 |
| bitmap_rotation | 85.0 | 100.0 | 35.0 | 80.0 | 100.0 |
| bricks_compose | 20.0 | 15.0 | 0.0 | 0.0 | 10.0 |
| josephus_elimination | 70.0 | 100.0 | 40.0 | 55.0 | 95.0 |
| clevr | 75.0 | 85.0 | 80.0 | 40.0 | 90.0 |
| **Table QA** | | | | | |
| lookup | 90.0 | 95.0 | 70.0 | 75.0 | 75.0 |
| extreme_name | 55.0 | 95.0 | 75.0 | 80.0 | 75.0 |
| count_condition | 75.0 | 100.0 | 65.0 | 75.0 | 80.0 |
| arithmetic | 75.0 | 80.0 | 50.0 | 45.0 | 65.0 |
| cell_neighbor | 70.0 | 95.0 | 45.0 | 85.0 | 80.0 |
| max_duplicate_count | 60.0 | 70.0 | 55.0 | 30.0 | 65.0 |
| value_occurrence_count | 85.0 | 85.0 | 75.0 | 80.0 | 85.0 |
| max_numeric_value | 30.0 | 50.0 | 35.0 | 40.0 | 50.0 |
| available_time_target | 70.0 | 85.0 | 65.0 | 95.0 | 70.0 |
| cross_row_comparison | 95.0 | 90.0 | 75.0 | 40.0 | 90.0 |
| group_aggregate | 90.0 | 90.0 | 75.0 | 90.0 | 85.0 |
| ratio_growth | 95.0 | 95.0 | 60.0 | 85.0 | 95.0 |
| conditional_inference | 50.0 | 95.0 | 55.0 | 70.0 | 80.0 |

*Continued on the next page*

*Table 9 continued*

| Task | GPT-5.6 | Kimi K3 | MiniMax | Sonnet 5 | Gemini |
|---|---|---|---|---|---|
| missing_value_inference | 90.0 | 95.0 | 50.0 | 85.0 | 95.0 |
| time_series_reasoning | 55.0 | 80.0 | 55.0 | 60.0 | 75.0 |
| unit_conversion | 60.0 | 95.0 | 75.0 | 85.0 | 95.0 |
| counterfactual | 55.0 | 80.0 | 45.0 | 60.0 | 65.0 |
| **Math** | | | | | |
| polynomial_coefficients | 85.0 | 70.0 | 45.0 | 50.0 | 80.0 |
| polynomial_slope | 90.0 | 65.0 | 35.0 | 10.0 | 85.0 |
| shape_area | 95.0 | 100.0 | 95.0 | 100.0 | 100.0 |
| polygon_area | 95.0 | 35.0 | 35.0 | 25.0 | 80.0 |
| polygon_angle | 100.0 | 100.0 | 100.0 | 80.0 | 100.0 |
| pyramid_volume | 35.0 | 25.0 | 20.0 | 25.0 | 20.0 |
| regression | 90.0 | 50.0 | 35.0 | 15.0 | 90.0 |
| solid_of_revolution | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| function_recognition | 100.0 | 100.0 | 85.0 | 95.0 | 100.0 |
| matrix_calculation | 100.0 | 100.0 | 75.0 | 95.0 | 100.0 |
| lever | 100.0 | 100.0 | 50.0 | 95.0 | 100.0 |
| interwoven_loops | 45.0 | 20.0 | 0.0 | 0.0 | 15.0 |
| **Macro-average** | **75.66** | **74.63** | **51.99** | **60.88** | **76.91** |

VRGym covers 68 tasks in five families, with difficulty controlled by each generator. We train on the 51 Math, Logic, Algorithm, and Chart QA tasks and hold out the 17 Table QA tasks. Table 10 lists every task by its generator name.

Table 10: Complete VRGym task coverage. The first four families (51 tasks) form the training distribution. Table QA (17 tasks) is held out.

| Task | Problem covered |
|---|---|
| **Math (12 tasks)** | |
| polynomial_coefficients | Recover polynomial coefficients from plotted points. |
| polynomial_slope | Reconstruct plotted polynomials and compute tangent slopes at marked points. |
| shape_area | Compute exact areas of composed, intersecting, or cut-out planar shapes. |
| polygon_area | Infer lattice vertices and compute the area of a simple polygon. |
| polygon_angle | Determine marked interior or exterior angles of convex or concave polygons. |
| pyramid_volume | Recover three-dimensional geometry and compute pyramid or frustum volume. |
| regression | Calculate statistics or fitted regression quantities from plotted observations. |
| solid_of_revolution | Compute volumes of solids whose radius is constant or polynomially varying. |
| function_recognition | Match symbolic functions to their curves across two coordinate plots. |
| matrix_calculation | Evaluate compatible matrix expressions, including products, scalar operations, and inverses. |
| lever | Infer unknown object weights from one or more balanced-lever constraints. |
| interwoven_loops | Trace over–under connections and count the distinct strips in an interwoven diagram. |
| **Logic (17 tasks)** | |
| hanoi | Continue the unique shortest Tower of Hanoi solution from an intermediate state. |
| ladder | Trace paths through a two-dimensional ladder diagram from entrances to exits. |
| 3d_ladder | Trace paths through a circular three-dimensional ladder shown from multiple views. |
| 3d_plates_recognition | Identify axial views of the same ordered stack of colored plates. |
| rectangle_counting | Count squares or rectangles in complete, missing-cell, or lattice-point diagrams. |
| cube_recognition | Match a marked vertex on a rotated cube to the indexed reference cube. |
| brick_counting | Integrate multiple views to count occupied unit cubes in a 3D assembly. |
| length_counting | Sum the lengths of connected segments using a visual unit scale. |
| intersection_counting | Count self-intersections or pairwise intersections among colored curves. |
| color_mixing | Select source colors that produce a target under additive-light or subtractive-pigment mixing. |
| logic_circuit | Propagate binary inputs through a combinational logic circuit. |
| finite_state_machine | Execute a Mealy machine on a supplied binary input sequence. |
| square_recognition | Match a marked point after planar grid rotation or reflection using colored boundaries. |

*Continued on the next page*

*Table 10 continued*

| Task | Problem covered |
|---|---|
| bitmap_rotation | Transform occupied grid cells by rotation, reflection, or their composition. |
| bricks_compose | Select rotated subassemblies that exactly compose a colored 3D brick object. |
| josephus_elimination | Simulate circular elimination with variable direction, step size, and exemption rules. |
| clevr | Answer compositional attribute, counting, comparison, and spatial-relation questions over rendered scenes. |
| **Algorithm (10 tasks)** | |
| maze | Find the unique shortest route through a two-dimensional cell maze. |
| 3d_maze | Find the unique shortest route across multiple maze floors connected by stairs. |
| graph_shortest_path | Compute the unique shortest path in a weighted directed graph. |
| graph_longest_path | Compute the unique longest path in a weighted directed acyclic graph. |
| topological_sort | Enumerate valid topological orders of a directed acyclic graph. |
| convex_hull | Identify the vertices of the minimum convex polygon enclosing planar points. |
| 3d_convex_hull | Identify the vertices of the minimum convex polyhedron enclosing 3D points. |
| traveling_salesman_problem | Find the unique minimum closed tour from a specified starting city. |
| rb_tree | Execute red–black-tree deletions and report the colors of the remaining keys. |
| maximum_flow | Compute maximum flow in a directed network with labeled capacities. |
| **Chart QA (12 tasks)** | |
| price_history | Read historical price movements and values from financial charts. |
| counterintuitive_reasoning | Reorder a deliberately nonchronological axis before answering a temporal comparison. |
| value_lookup | Retrieve a value for a specified category, series, or observation. |
| max_duplicate_count | Find the greatest number of times a plotted value occurs. |
| extreme_category | Identify the category or series with the highest or lowest value. |
| series_calculation | Apply an arithmetic operation to selected values within a plotted series. |
| difference | Compute the difference between two specified chart values. |
| summation | Sum values selected across categories or series. |
| largest_change | Identify the interval or series with the largest adjacent change. |
| distribution_statistic | Compute a discrete distribution statistic from a plot. |
| continuous_distribution_statistic | Estimate or compute a statistic of a continuous plotted distribution. |
| matrix_extreme | Locate an extreme entry in a matrix-style visualization such as a heatmap. |
| **Table QA (17 held-out tasks)** | |
| lookup | Retrieve a cell, row, group, title, or footer value. |
| extreme_name | Return the row or entity associated with an extreme value. |
| count_condition | Count rows that satisfy a stated condition. |
| arithmetic | Combine values from one or more cells using arithmetic operations. |
| cell_neighbor | Navigate the table structure to retrieve a neighboring cell. |
| max_duplicate_count | Determine the largest number of occurrences of any repeated table value. |
| value_occurrence_count | Count how often a specified value occurs. |
| max_numeric_value | Find the largest numeric entry under the requested scope. |
| available_time_target | Determine whether, or when, an available time interval meets a target. |
| cross_row_comparison | Compare aligned fields across different rows or entities. |
| group_aggregate | Aggregate values within and across hierarchical table groups. |
| ratio_growth | Compute a ratio, relative change, or growth rate across entries. |
| conditional_inference | Filter by one or more conditions and infer the requested result. |
| missing_value_inference | Recover a missing entry from totals or other table constraints. |
| time_series_reasoning | Order temporal entries and reason over trends or intervals. |
| unit_conversion | Convert table values between stated measurement units. |
| counterfactual | Recompute a table result under a hypothetical change. |

The supplementary material includes the code and reference examples for every listed task at each supported difficulty level.

## D FULL ABLATION RESULTS

Table 11 expands the averages in Table 4. All entries report accuracy (%). Removal only skips random value masking and uses text removal, while mask only skips text removal and retains the masked derivation. Both variants always mask the answer-bearing subset $A$.

Table 11: Per-dataset Qwen2.5-VL-7B ablation results. *Avg.* is the macro-average over all seven datasets.

| **Dataset** | **Removal only** | **Mask only** | $\delta^+ = \delta^- = 0.05$ | $\alpha = 1.0$ |
|---|---|---|---|---|
| VRGym test | 21.71 | 21.19 | 20.93 | 21.46 |
| *Related domain* | | | | |
| MathVista | 71.10 | 70.17 | 71.26 | 70.93 |
| MathVision | 27.01 | 27.75 | 26.93 | 26.64 |
| MathVerse | 48.84 | 48.62 | 49.38 | 49.73 |
| DynaMath | 51.21 | 51.42 | 52.40 | 51.28 |
| *General domain* | | | | |
| MMMU | 54.10 | 53.88 | 53.52 | 53.81 |
| MMMU-Pro | 37.96 | 37.93 | 38.00 | 37.40 |
| **Avg.** | 44.56 | 44.42 | 44.63 | 44.46 |

Table 12 reports the number of groups with non-zero advantage over 500 steps, using the definition in Section 4.3. Every variant exceeds the 33.0% share observed for unguided GRPO but remains below default OPGG. Removing either phase produces the largest reductions, with mask only giving the lowest share at 44.9%.

Table 12: Groups with non-zero advantage out of the 16,000 Qwen2.5-VL-7B groups sampled over 500 steps, counted as in Section 4.3. $\Delta$ is the difference in group count from default OPGG. Unguided GRPO is included for reference.

| **Variant** | **Groups** | **Share** | $\Delta$ |
|---|---|---|---|
| **Default OPGG** | **7,851** | **49.1%** | — |
| Removal only | 7,476 | 46.7% | −375 |
| Mask only | 7,188 | 44.9% | −663 |
| $\delta^+ = \delta^- = 0.05$ | 7,672 | 48.0% | −179 |
| $\alpha = 1.0$ | 7,577 | 47.4% | −274 |
| GRPO (no guidance) | 5,281 | 33.0% | −2,570 |

## E PROCEDURAL VERSUS ORACLE GUIDANCE

Table 13 gives the per-dataset results for the fixed-guidance comparison. Procedural guidance leads on six datasets. Oracle guidance leads on MMMU by 0.12 points.

Table 13: Per-dataset accuracy (avg@8, %) after 200 steps of fixed-guidance training. $\Delta$ is procedural minus oracle.

| **Dataset** | **Procedural** | **Oracle** | $\Delta$ |
|---|---|---|---|
| VRGym test | **17.74** | 16.20 | +1.54 |
| *Related domain* | | | |
| MathVista | **69.77** | 68.89 | +0.88 |
| MathVision | **26.36** | 25.16 | +1.19 |
| MathVerse | **48.91** | 48.24 | +0.67 |
| DynaMath | **51.54** | 50.65 | +0.89 |
| *General domain* | | | |
| MMMU | 53.37 | **53.49** | −0.12 |
| MMMU-Pro | **37.34** | 37.06 | +0.28 |
| **Avg.** | **43.57** | 42.81 | +0.76 |