BOOTSTRAPPING EXPLORATION WITH GROUP-LEVEL NATURAL LANGUAGE FEEDBACK IN REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) for large language model (LLM) post-training relies on scalar outcome rewards that convey only *whether* the model succeeded, not *why* it failed, forcing blind exploration that stalls in failure-dominated regimes where advantages collapse. Yet rich natural language (NL) feedback, from execution traces to textual critiques, often arises as a byproduct of the reward system itself but remains largely discarded. We propose GOLF, an RL framework that leverages **GrO**up-level **L**anguage **F**eedback to operationalize refinement as a learnable *meta-capability*. Rather than refining each failure in isolation, GOLF aggregates external critiques and intra-group attempts into a unified context, surfacing systemic error patterns that broaden solution diversity. The resulting refinements are adaptively injected as off-policy scaffolds in failure-dominated regimes, while generation and refinement are jointly optimized so that improved self-refinement yields stronger scaffolds that accelerate exploration. Across verifiable and non-verifiable tasks, GOLF achieves the best average score in every setting: it surpasses the strongest NL-feedback baseline, Critique-GRPO, by +11.86 points on `Llama-3.1-8B-Instruct`, and its gains hold consistently as `Qwen3` policies scale from 4B to 14B. It also matches the baseline's final performance in up to $\sim$2.3$\times$ fewer steps, while sustaining broader exploration and cultivating test-time self-refinement that generation-only RL fails to develop.

1 INTRODUCTION

Reinforcement learning has emerged as the central paradigm for large language model (LLM) post-training, supporting both alignment with human preferences (Ouyang et al., 2022) and improvement of reasoning capabilities (Lambert et al., 2025; Guo et al., 2025). Despite its broad success, exploration remains a core bottleneck: current RL methods rely on sparse outcome rewards that compress rich feedback into scalar signals, whether reducing diverse reasoning errors to binary correctness or collapsing multi-dimensional rubrics into a single score. Such rewards convey only *whether* the model succeeded, not *why* it failed or *how* to improve, forcing the policy into blind trial and error.

Yet, in practice, the supervision available to LLMs extends far beyond scalar outcomes. Execution traces from code environments and textual critiques from generative reward models (Ankner et al., 2024) each produce rich natural language (NL) feedback. Such feedback is often a byproduct of the reward system itself, yet remains largely unused by current RL methods. In failure-dominated regimes, where scalar rewards yield uninformative advantages that stall learning, NL feedback can diagnose specific failure modes and provide the targeted signal needed to escape these dead zones.

In this work, we propose to treat refinement not as an ad-hoc correction mechanism but as a *meta-capability*: a learnable ability to identify systemic error patterns and self-correct, jointly optimized with generation so that the two co-evolve (Figure 1). To provide targeted refinement guidance, we aggregate *group-level* NL feedback from two complementary sources: (i) *external critiques* that identify what went wrong in each attempt, and (ii) *intra-group attempts* that supply alternative partial solutions and diverse failure patterns across the rollout group. Our preliminary study (Appendix A) confirms that aggregating both sources produces higher-quality refinements than either alone.

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.

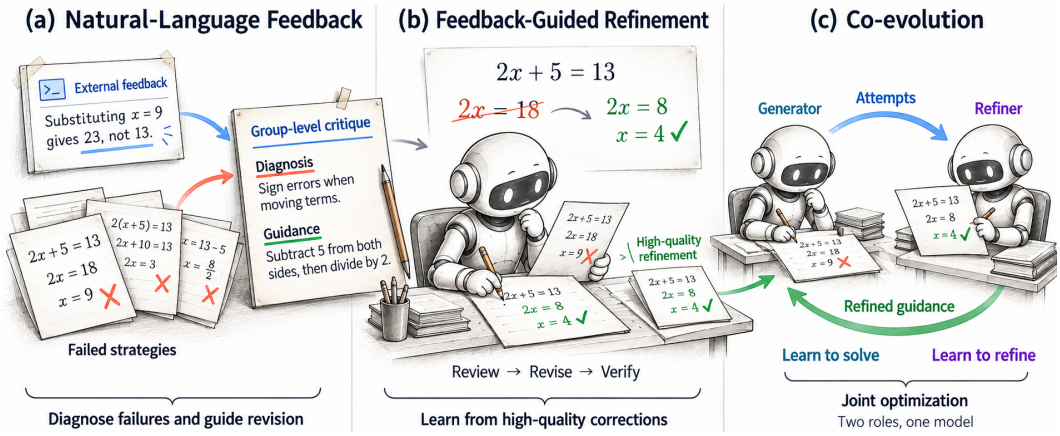


Figure 1: Overview of GOLF. (a) External NL feedback and the failed attempts of a rollout group are aggregated into a group-level critique that diagnoses shared error patterns and guides revision. (b) Conditioned on this critique, the policy refines a failed attempt into a high-quality correction, which is injected into the group as an off-policy scaffold. (c) One model plays both generator and refiner, and jointly optimizing the two roles lets them co-evolve.

Building on this insight, GOLF operationalizes this meta-capability through three coupled mechanisms: it aggregates failed attempts and their critiques into one context (**Group-Level Feedback Aggregation**), surfacing systemic error patterns; it *adaptively* injects the resulting refinements as off-policy scaffolds only in failure-dominated groups, where advantages would otherwise vanish (**Adaptive Guidance**); and it jointly optimizes generation and refinement within a single RL step (**Joint Optimization**), so that better self-refinement yields stronger scaffolds.

We evaluate GOLF on verifiable and non-verifiable tasks across model families and sizes. On four non-verifiable benchmarks, GOLF surpasses the strongest baseline by **+11.86** average points on `Llama-3.1-8B-Instruct` and matches the baseline's final performance in $\sim$2.3$\times$ fewer training steps. On verifiable tasks, GOLF consistently improves over GRPO across math reasoning, instruction following, and code generation, and raises Pass@k, indicating that feedback-guided refinement broadens solution coverage rather than collapsing onto a narrow set of strategies.

Our contributions are summarized as follows:

- We propose GOLF, an RL framework that operationalizes refinement as a learnable meta-capability through group-level NL feedback aggregation, adaptive off-policy guidance in failure-dominated regimes, and joint optimization of generation and refinement.
- On both verifiable and non-verifiable tasks, GOLF achieves the best average score in every setting while converging in up to $\sim$2.3$\times$ fewer steps. The gains hold over three random seeds and grow when scaling the policy to 14B, and ablations confirm that external critiques and intra-group attempts provide complementary supervision.
- Analysis shows that GOLF sustains broader exploration (higher policy entropy and Pass@k) and that joint optimization cultivates test-time self-refinement that generation-only RL lacks.

2 RELATED WORK

Optimization with Natural Language Feedback. At inference time, LLMs can use textual feedback to self-improve by turning it into self-reflective experiences (Shinn et al., 2023) or by iteratively refining previous attempts (Madaan et al., 2023; Kamoi et al., 2024). Another line of research learns from natural language feedback via imitation learning. For example, Chen et al. (2024) fine-tune models on high-quality refinements produced with textual feedback, while Wang et al. (2025) directly train models to imitate critiques. Recent work also explores incorporating natural language feedback into the RL process. Wang et al. (2026) and Cao et al. (2024) train reward models that convert textual critiques into token-level or span-level reward signals, enabling finer-grained

credit assignment during reinforcement learning. Most similar to our work, Critique-GRPO (Zhang et al., 2025) integrates critique-guided refinements into online RL optimization. We go beyond external critiques by also exploiting intra-group feedback from multiple attempts, which exposes diverse failure modes, and we also cover non-verifiable tasks.

Guiding Model Exploration in Reinforcement Learning. Exploration in RL for LLMs is bounded by the capabilities of the current policy (Yue et al., 2025), so recent work injects guidance when on-policy sampling fails. LUFFY (Yan et al., 2025) mixes expert demonstrations into the rollout group as off-policy samples; Guide (Nath et al., 2025) adds in-context hints for problems whose rollouts all fail and reweights the resulting trajectories with importance sampling; Scaf-GRPO (Zhang et al., 2026) provides tiered hints once learning plateaus; and Critique-GRPO (Zhang et al., 2025) refines individual failed responses with critiques. GOLF differs in three respects. First, its guidance is produced by the policy itself from the *group* of failed attempts and their critiques, rather than from external demonstrations or hints. Second, the refiner is trained jointly with the generator, so guidance improves as training proceeds, related in spirit to RL-trained self-correction (Kumar et al., 2025). Third, it applies to non-verifiable tasks, where feedback comes from a generative reward model. LANPO (Li et al., 2025a) also uses language feedback to guide exploration while numerical rewards drive optimization, but targets mathematical reasoning without a jointly trained refiner.

3 PRELIMINARIES

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is a widely used RL algorithm for training LLMs. It simplifies PPO (Schulman et al., 2017) by removing the value function. For each prompt $x \sim \mathcal{D}$, GRPO samples a group of N responses $\{y^{(i)}\}_{i=1}^{N}$ with $y^{(i)} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$. Each response is then scored by a reward function, yielding $r^{(i)}$. The group-relative advantage is computed as $A^{(i)} = \big(r^{(i)} - \text{mean}(\{r^{(j)}\}_{j=1}^{N})\big)/\text{std}(\{r^{(j)}\}_{j=1}^{N})$. GRPO then optimizes the clipped surrogate objective:

$$\mathcal{J}_{\text{GRPO}}(\theta; x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|y^{(i)}|}\sum_{t=1}^{|y^{(i)}|}\text{CLIP}\Big(\rho_t^{(i)}(\theta),\, A^{(i)},\, \varepsilon\Big) - \beta\, D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})\,, \tag{1}$$

where $\text{CLIP}(\rho, A, \varepsilon) = \min\big(\rho A,\ \text{clip}(\rho,\, 1{-}\varepsilon,\, 1{+}\varepsilon)\, A\big)$, $\text{clip}(\rho,\, a,\, b) \equiv \max(a, \min(b, \rho))$ denotes elementwise clipping, ε and β are hyperparameters controlling the clipping range and the KL penalty, respectively, and $\rho_t^{(i)}(\theta) = \frac{\pi_\theta(y_t^{(i)} | x, y_{<t}^{(i)})}{\pi_{\theta_{\text{old}}}(y_t^{(i)} | x, y_{<t}^{(i)})}$ is the importance sampling ratio.

4 METHODOLOGY

GOLF consists of three tightly coupled components, as illustrated in Figure 2.

4.1 GROUP-LEVEL FEEDBACK AGGREGATED REFINEMENT

For each prompt $x \sim \mathcal{D}$, we sample a group of N responses $\mathcal{G}_{\text{gen}}(x) = \{y^{(i)}\}_{i=1}^{N}$, and query the reward model for the scalar reward $r^{(i)} = R(x, y^{(i)}) \in \{0, 1\}$ and the corresponding critique $c^{(i)}$. We consider two types of NL feedback: (1) **External feedback** refers to the critique $c^{(i)}$ associated with a specific response $y^{(i)}$; (2) **Intra-group feedback** refers to alternative responses in the group $\mathcal{G}_{\text{gen}}(x)$, which often contain complementary partial ideas.

Instead of refining each failure in isolation, we *aggregate* multiple failed attempts and their critiques into a single refinement context, exposing diverse failure modes. Concretely, we collect the failure set $\mathcal{F}(x) = \{(y^{(i)}, c^{(i)}) \mid r^{(i)} = 0\}$, and construct the aggregated refinement prompt $p_{\text{agg}}(x) = \text{Concat}\big(x,\ \mathcal{F}(x)\big)$ (prompts in Appendix B). To respect the maximum prompt length, we fill the budget left after the template with complete failed attempts and their critiques in ascending order of length (Appendix C). Conditioned on $p_{\text{agg}}(x)$, we sample a refinement group $\mathcal{G}_{\text{ref}}(x) = \{\tilde{y}^{(j)}\}_{j=1}^{N}$ with $\tilde{y}^{(j)} \sim \pi_{\theta_{\text{old}}}(\cdot \mid p_{\text{agg}}(x))$ and score each refinement by $\tilde{r}^{(j)} = R(x, \tilde{y}^{(j)})$. This aggregation allows the model to identify mistakes, fill gaps, and synthesize diverse partial solutions across attempts, producing refinements that surpass any single attempt.

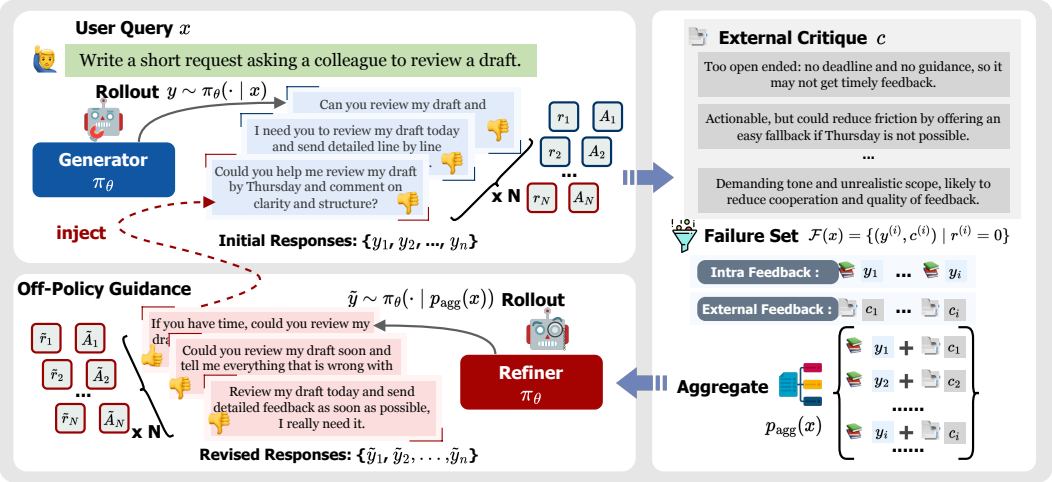


Figure 2: The GOLF framework. Failed responses and their critiques are aggregated into group-level feedback that conditions refinement; successful refinements are adaptively injected into failure-dominated groups as off-policy guidance, and generation and refinement are optimized jointly.

4.2 ADAPTIVE GUIDANCE VIA MIXED POLICY OPTIMIZATION

In low-reward regimes, on-policy groups often contain only zero-reward samples, yielding weak group-relative advantages and slow policy improvement. We therefore treat high-quality refinements as off-policy scaffolds and inject them into the sampled group to restore informative advantages. The policy is then optimized with a mixed objective that combines on-policy and off-policy trajectories.

Adaptive Injection. For each prompt x, we compute the average reward $s(x) = \frac{1}{N}\sum_{y \in \mathcal{G}_{\text{gen}}(x)} R(x, y)$. We trigger injection only in failure-dominated regimes, *i.e.*, when $s(x)$ falls below a threshold η. We set $\eta = 1/N$, so that with binary rewards injection is triggered exactly when all N rollouts fail. In that case, we form the set of *successful* refinements $\mathcal{S}_{\text{ref}}(x) = \{\tilde{y} \in \mathcal{G}_{\text{ref}}(x) \mid R(x, \tilde{y}) = 1\}$. If $\mathcal{S}_{\text{ref}}(x) \neq \emptyset$, we randomly select $\tilde{y}^\star \in \mathcal{S}_{\text{ref}}(x)$ and replace one randomly chosen failed response in $\mathcal{G}_{\text{gen}}(x)$ with it. The gate follows from the advantages defined below: with k successes, a success receives $\hat{A} = (N-k)/N$. For $k = 0$ all advantages vanish and an injected refinement creates a signal of $(N-1)/N$; for $k \geq 1$ injection only shrinks the policy's own successes to $(N-k-1)/N$ (Appendix C). We ablate different thresholds in Table 22.

Mixed Policy Optimization. Let $\mathcal{G}_{\text{aug}}(x) = \mathcal{G}_{\text{on}}(x) \cup \mathcal{G}_{\text{off}}(x)$, where $\mathcal{G}_{\text{on}}(x) \subset \mathcal{G}_{\text{gen}}(x)$ are the $N_{\text{on}} = |\mathcal{G}_{\text{on}}(x)|$ un-replaced on-policy samples and $\mathcal{G}_{\text{off}}(x)$ contains the $N_{\text{off}} = |\mathcal{G}_{\text{off}}(x)|$ injected off-policy refinements, with $N_{\text{on}} + N_{\text{off}} = N$. We optimize the policy π_θ over $\mathcal{G}_{\text{aug}}(x)$ with the mixed objective of LUFFY (Yan et al., 2025), using a cross-context ratio for the off-policy term:

$$\mathcal{J}_{\text{Mixed}}(\theta) = \frac{1}{Z}\left[\underbrace{\sum_{i=1}^{N_{\text{on}}}\sum_{t=1}^{|y^{(i)}|} \text{CLIP}\left(\rho_t^{(i)}(\theta),\, \hat{A}^{(i)},\, \varepsilon\right)}_{\text{on-policy objective}} + \underbrace{\sum_{j=1}^{N_{\text{off}}}\sum_{t=1}^{|\tilde{y}^{(j)}|} f(\tilde{\rho}_t^{(j)}(\theta))\,\hat{A}^{(j)}}_{\text{off-policy objective}}\right], \tag{2}$$

where $Z = \sum_{i=1}^{N_{\text{on}}} |y^{(i)}| + \sum_{j=1}^{N_{\text{off}}} |\tilde{y}^{(j)}|$ normalizes by the total number of tokens, and

$$\rho_t^{(i)}(\theta) = \frac{\pi_\theta(y_t^{(i)} \mid x,\, y_{<t}^{(i)})}{\pi_{\theta_{\text{old}}}(y_t^{(i)} \mid x,\, y_{<t}^{(i)})}, \quad \tilde{\rho}_t^{(j)}(\theta) = \frac{\pi_\theta(\tilde{y}_t^{(j)} \mid x,\, \tilde{y}_{<t}^{(j)})}{\pi_{\theta_{\text{old}}}(\tilde{y}_t^{(j)} \mid p_{\text{agg}}(x),\, \tilde{y}_{<t}^{(j)})}. \tag{3}$$

The two contexts in $\tilde{\rho}$ are deliberate. The denominator is the distribution the refinement was actually sampled from, while the numerator conditions on the plain prompt x, the only input available at test time; positive-advantage refinements therefore raise $\pi_\theta(\cdot \mid x)$ and are distilled into direct generation.

Table 1: Results on non-verifiable benchmarks (WildBench scores range from -100 to 100). Best and second-best results per column are in **bold** and underlined.

Method	AlpacaEval v2.0 LC win rate (%)	WildBench Score	ArenaHard v2.0 Win rate (%)	CreativeWriting v3 Score	Avg.
Llama-3.1-8B-Instruct					
Base model	31.79	−8.25	5.57	53.96	20.77
+ Likert-GRPO	34.98	13.48	11.73	64.10	31.07
+ Pairwise-GRPO	43.19	25.54	13.30	62.95	36.25
+ Rubric-GRPO	36.12	26.51	15.57	**68.12**	36.58
+ Refinement-FT	33.21	12.60	9.60	60.43	28.96
+ Critique-GRPO	43.31	25.09	13.73	65.76	36.97
+ GOLF (ours)	**69.67**	**34.42**	**25.03**	66.21	**48.83**
Qwen3-8B					
Base model	52.60	48.05	33.90	63.27	49.46
+ Likert-GRPO	61.06	58.01	41.70	69.56	57.58
+ Pairwise-GRPO	68.34	67.77	50.10	68.08	63.57
+ Rubric-GRPO	68.88	67.09	50.08	69.21	63.82
+ Refinement-FT	63.23	54.16	38.07	65.65	55.28
+ Critique-GRPO	69.82	64.84	49.63	67.30	62.90
+ GOLF (ours)	**71.94**	**68.16**	**52.00**	**70.78**	**65.72**

This departs from the same-context ratio of LUFFY and is similar in spirit to the importance weighting of Guide (Nath et al., 2025) (Appendix C). Following Dr. GRPO (Liu et al., 2025), we remove standard-deviation normalization and compute advantages within the augmented group $\mathcal{G}_{\text{aug}}(x)$: $\hat{A}^{(i)} = R(x, y^{(i)}) - \frac{1}{|\mathcal{G}_{\text{aug}}(x)|} \sum_{y \in \mathcal{G}_{\text{aug}}(x)} R(x, y)$. Since we set $\beta = 0$ in all experiments (Appendices E and F), the KL term of Equation (1) is omitted. We further apply the reshaping function $f(u) = u/(u + \lambda)$ with $\lambda = 0.1$ (Yan et al., 2025) to off-policy ratios and omit the clip operation to emphasize low-probability yet effective actions from injected refinements. Both λ and $\eta = 1/N$ are shared across all tasks and backbones without per-task tuning (Appendix C).

4.3 Joint Optimization for Self-Refinement

Standard RL post-training optimizes problem solving but not test-time self-refinement, and we observe that its gains may not transfer to refinement (Section 6.3). We therefore train the LLM to improve both direct problem solving and feedback-conditioned refinement within one RL process.

Concretely, for each prompt x we optimize the generation group $\mathcal{G}_{\text{gen}}(x)$ with adaptive injection and Equation (2), and the refinement group $\mathcal{G}_{\text{ref}}(x)$ with GRPO using advantages computed within $\mathcal{G}_{\text{ref}}(x)$; both are updated jointly within a single RL step.

Jointly optimizing these two behaviors forms a positive feedback loop: as self-refinement improves, it produces higher-quality refinement trajectories that serve as stronger off-policy scaffolds for mixed-policy optimization, increasing the likelihood of discovering rewarding trajectories.

5 Experiments

5.1 Non-verifiable Tasks

Setup. We consider two model families, `Llama-3.1-8B-Instruct` (Grattafiori et al., 2024) and `Qwen3-8B` (Yang et al., 2025). All models are trained on 7,500 conversational prompts from WildChat-IF[1]. We evaluate on four widely used alignment benchmarks: AlpacaEval v2.0 (Dubois et al., 2024), ArenaHard v2.0 (Li et al., 2025b), WildBench (Lin et al., 2025), and CreativeWriting v3 (Paech, 2025). For AlpacaEval v2.0 and ArenaHard v2.0, we follow official evaluation protocols and report length-controlled (LC) win rate and win rate, respectively. For WildBench and

[1] https://huggingface.co/datasets/allenai/tulu-3-wildchat-if-on-policy-8b

CreativeWriting v3, we follow Bhaskar et al. (2025) and report the LLM-as-a-judge scores. We use GPT-4o (`gpt-4o-2024-05-13`) (OpenAI, 2024) as the judge across all benchmarks. Detailed benchmark descriptions and evaluation protocols are in Appendix D.

Baselines. We compare GOLF against five baselines spanning two categories. **(a) Scalar-Reward RL**: all methods build upon GRPO, differing only in how the scalar reward is computed. *Likert-GRPO* prompts the judge for a 1–10 Likert score; *Pairwise-GRPO* derives a binary reward from pairwise comparison against a GPT-4o reference; *Rubric-GRPO* (Gunjal et al., 2026) scores responses against prompt-specific rubrics generated by DeepSeek-V3.2 (DeepSeek-AI, 2025). **(b) Learning from NL Feedback**: *Refinement-FT* (Chen et al., 2024) fine-tunes the policy on refined responses generated using external critiques; *Critique-GRPO* (Zhang et al., 2025) incorporates critiques into the RL loop to guide individual response refinement, built on the same pairwise reward as GOLF. All methods use `Qwen3-235B-A22B-Instruct-2507`[2] as the generative reward model, which jointly produces scalar rewards and natural language critiques. Because all methods share this reward model and the same evaluation judge, judge-specific biases affect all methods alike (details in Appendix E).

Main Results. Table 1 reports results across four non-verifiable benchmarks. GOLF achieves the highest average score on both model families, surpassing the strongest baseline by **+11.86** points on `Llama-3.1-8B-Instruct` (48.83 vs. 36.97); over three random seeds, GOLF scores 47.78 ± 1.19 versus 36.52 ± 0.71 for Pairwise-GRPO (Appendix M.1). On the stronger `Qwen3-8B`, methods are closer, yet GOLF ranks first on all four benchmarks (+16.26 over the base model).

Among baselines, structured rewards (rubrics, reference-anchored comparisons) beat Likert scores; GOLF extends this trend with **more diagnostic**, **group-aggregated** feedback, improving over Critique-GRPO by **+11.86** (`Llama`) and **+2.82** (`Qwen`) without relying on longer responses (Appendix K).

Robustness to a Weaker Reward Model. To test whether the gains depend on a frontier-scale reward model, we rerun GOLF and Pairwise-GRPO on `Llama-3.1-8B-Instruct` with the much smaller `Qwen3-30B-A3B` in place of `Qwen3-235B-A22B` as the reward model. Both methods degrade, but the margin of GOLF is preserved (Figure 3a; Appendix J), suggesting that group-level aggregation adds redundancy across noisy per-rollout judgments.

5.2 VERIFIABLE TASKS

Setup. We experiment with `Qwen3-4B` and `Qwen3-8B` (Yang et al., 2025). For mathematical reasoning, following Critique-GRPO (Zhang et al., 2025), we train on 4,000 problems from OpenR1-Math (Bakouch et al., 2025) and evaluate on AIME24, AIME25, and AMC23 (LI et al., 2024); as in Critique-GRPO, the math critique states the ground-truth answer, yet a blind LLM critic that never sees it still improves over GRPO by +4.58 points (Appendix F). For instruction following, we follow the recipe of IFBench (Pyatkin et al., 2025), using RL prompts from IFTrain[3], each with up to 5 constraints sampled from IFEval (Zhou et al., 2023) and IFTrain. We further filter out prompts with conflicting constraints and evaluate on IFEval and IFBench.

Baselines. In addition to the baselines in non-verifiable settings, we include *Critique-FT* (Wang et al., 2025), which trains the policy to internalize critical thinking for reasoning tasks using critique-augmented data. For GRPO baselines, we adopt the Clip-Higher strategy (Yu et al., 2025) with $\varepsilon_{\text{high}} = 0.28$ to strengthen them. Implementation details are provided in Appendix F.

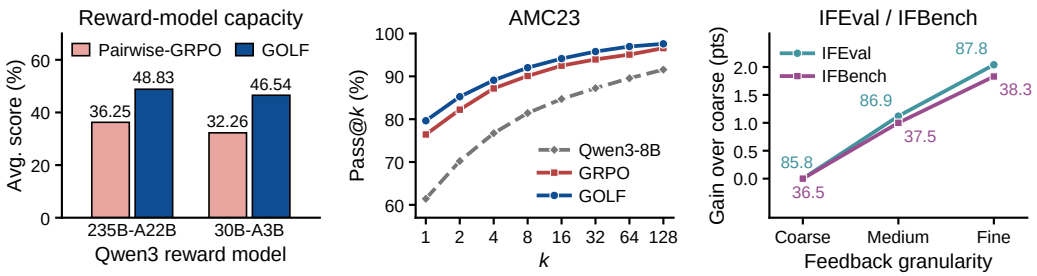
Main Results. Table 2 shows that GOLF achieves the best results across all verifiable benchmarks. GOLF improves the average over GRPO by **+3.17** (`Qwen3-4B`) and **+2.98** (`Qwen3-8B`), whereas Critique-GRPO gains only +1.18 and +0.15; over three seeds on math, GOLF scores 60.14 ± 0.73 versus 57.77 ± 0.71 for Critique-GRPO (Appendix M.1). We attribute this to jointly optimizing generation and refinement, which lets the policy internalize self-refinement. Since IFBench constraints are out of domain for training, the gains there also suggest generalization to unseen constraints.

[2]Chosen for its high pairwise agreement with our evaluation judge, GPT-4o (Appendix I).

[3]https://huggingface.co/datasets/allenai/IF_multi_constraints_upto5

Table 2: Results on verifiable benchmarks. Best and second-best are in **bold** and underlined.

Method	Mathematical reasoning			Instruction following		Avg.
	AIME24	AIME25	AMC23	IFBench	IFEval	
Qwen3-4B						
Base model	22.53	18.55	59.41	23.67	81.52	41.14
+ Refinement-FT	31.67	21.25	64.06	30.44	83.73	46.23
+ Critique-FT	34.58	24.58	65.94	31.67	82.63	47.88
+ GRPO	42.72	35.42	76.85	33.33	84.45	54.55
+ Critique-GRPO	45.72	35.89	76.14	35.67	85.21	55.73
+ GOLF (ours)	**49.18**	**38.10**	**77.15**	**37.67**	**86.51**	**57.72**
Qwen3-8B						
Base model	27.97	19.60	61.32	27.00	83.55	43.89
+ Refinement-FT	42.08	27.50	67.81	34.33	84.29	51.20
+ Critique-FT	46.75	28.75	70.31	33.60	84.45	52.77
+ GRPO	55.05	38.02	78.61	35.65	84.76	58.42
+ Critique-GRPO	55.49	37.86	77.58	36.33	85.58	58.57
+ GOLF (ours)	**58.49**	**41.65**	**80.74**	**38.33**	**87.80**	**61.40**



(a) Weaker reward model (b) Pass@k on AMC23 (c) Feedback granularity

Figure 3: (a) With a smaller reward model, GOLF keeps its margin over Pairwise-GRPO (`Llama-3.1-8B-Instruct`, average of four benchmarks). (b) GOLF keeps a Pass@k advantage for all k (`Qwen3-8B`). (c) Finer feedback yields larger gains (`Qwen3-8B`; labels: absolute scores).

Exploration Diversity. On AMC23 (Figure 3b), GOLF keeps its Pass@k advantage over the base model and GRPO for k up to 128, indicating broader solution coverage (AIME25 in Figure 8b). On non-verifiable tasks, GOLF also maintains higher policy entropy than the baseline (Appendix N.3).

Effect of Feedback Granularity. Instruction following lets us control feedback granularity, since each constraint is checked by an independent Python function. We vary the feedback given to refinement across three levels, keeping the scalar reward identical: **fine** (a critique of which constraints are violated and why), **medium** (only the identities of failed constraints), and **coarse** (only the overall pass rate). Performance improves monotonically from coarse to fine on both IFEval and IFBench (Figure 3c), showing that GOLF benefits from richer feedback (details in Appendix L).

Comparison with On-Policy Distillation. SDPO (Hübotter et al., 2026) exploits execution feedback in code generation by using successful rollouts as hindsight for credit assignment, whereas GOLF aggregates feedback across diverse failures. Following its protocol on LiveCodeBench v6 (Jain et al., 2025) with `Qwen3-8B` (Appendix G), GOLF slightly outperforms SDPO (Avg@4: 47.71 vs. 47.52) and both improve over GRPO (44.08; Figure 5b).

Scaling to Larger Policies. To test whether the benefits persist beyond 8B, we train `Qwen3-14B` with the recipes, hyperparameters, and protocols of Tables 1 and 2, without re-tuning (Figure 4). The margin of GOLF does not shrink with scale. On non-verifiable tasks, it grows over Pairwise-GRPO from +2.15 (8B) to +3.50 (14B) and remains stable over Critique-GRPO (+2.82 to +2.87); on verifiable tasks, the gain over GRPO is +3.17, +2.98, and +4.04 at 4B, 8B, and 14B, and GOLF ranks first on all nine benchmarks at 14B. We attribute the growing margin to the stronger backbone's

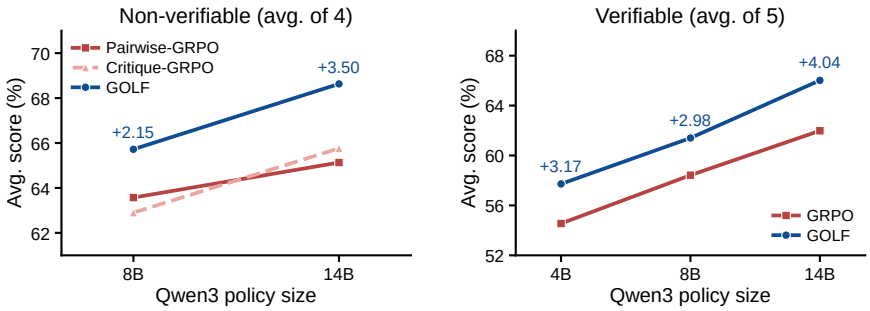
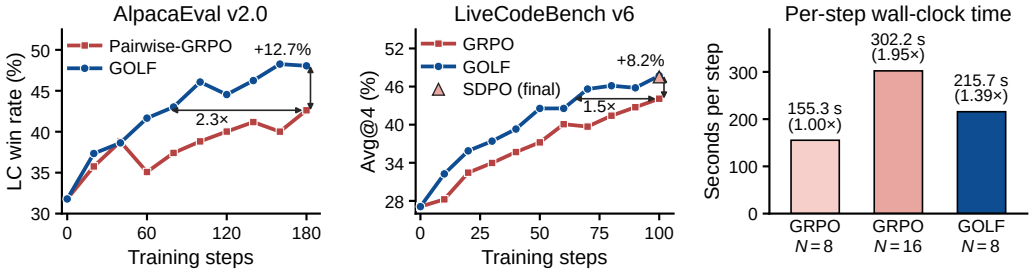


Figure 4: Scaling the `Qwen3` policy with an unchanged recipe. Numbers above GOLF give its gain over Pairwise-GRPO (left) and GRPO (right). The 4B and 8B points are the single-run results of Tables 1 and 2; the 14B points are averaged over three random seeds. At 14B, GOLF ranks first on all nine benchmarks; per-benchmark results are in Appendix M.2.



(a) AlpacaEval (`Llama-3.1-8B`) (b) LiveCodeBench (`Qwen3-8B`) (c) Training cost per step

Figure 5: GOLF converges faster and reaches a higher plateau. (a, b) GOLF matches the baseline's final score after ∼80 and ∼65 steps, and on LiveCodeBench v6 it also slightly surpasses SDPO (Hübotter et al., 2026). (c) Per-step wall-clock time on `Llama-3.1-8B-Instruct`. Curves evaluate intermediate checkpoints; the main tables report the final checkpoint. More curves are in Appendix N.4.

greater capacity for refinement, which turns the same aggregated feedback into better scaffolds; Section 6.3 evaluates this refinement capability directly.

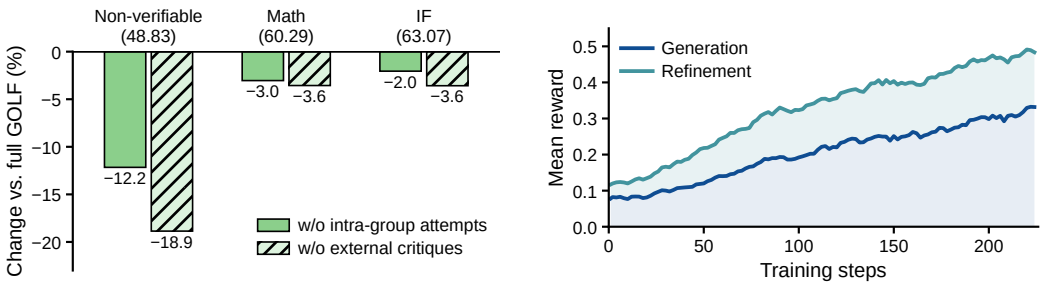
6 ANALYSIS

We analyze training efficiency, the contribution of each feedback source, and joint optimization (Sections 6.1 to 6.3); ablations of the adaptive-guidance design (mixed-policy RL vs. SFT, adaptive vs. always injection) are in Appendix N.2 and a case study in Appendix N.6.

6.1 STEP EFFICIENCY

Group-level NL feedback also accelerates convergence. Figures 5a and 5b show that GOLF matches the baseline's final score in roughly 80 and 65 training steps on AlpacaEval v2.0 and LiveCodeBench v6, yielding ∼2.3× and ∼1.5× step efficiency, and reaches a notably higher plateau (+12.7% and +8.2%, respectively). The same pattern holds on the remaining benchmarks (Appendix N.4).

The refinement stage doubles the rollouts per prompt, so we also compare with rollout-matched GRPO (N=16): doubling the rollouts gains GRPO only +2.15 points versus +12.58 for GOLF (Table 23), so the improvement stems from feedback-guided refinement rather than more sampling. Per step, GOLF costs 1.39× GRPO (N=8), less than rollout-matched GRPO (1.95×; Figure 5c, Table 24); combined with the 2.3× step efficiency, this reduces total training wall-clock time.



(a) Feedback-source ablation (b) Training dynamics (`Llama-3.1-8B`)

Figure 6: (a) Removing either feedback source lowers the average score on every task family (full-GOLF average in parentheses). (b) Generation and refinement rewards rise jointly.

Table 3: RefineBench pass rate (%) on `Llama-3.1-8B-Instruct` (pass: all checklist items satisfied). *Self*: the model refines its own response; *guided*: refinement is conditioned on the checklist.

	Self-refinement				Guided refinement			
Method	Biology	Chemistry	Law	**Avg.**	Biology	Chemistry	Law	**Avg.**
Base model	9.09	8.33	23.94	13.79	18.18	27.78	49.30	31.75
+ GRPO	22.73	27.77	22.24	24.25	27.78	55.55	45.07	42.80
+ GOLF (ours)	**27.27**	**33.33**	**24.64**	**28.41**	**54.55**	**66.67**	**50.00**	**57.07**

6.2 Effect of Group-level Feedback

We ablate the two sources of group-level feedback: **w/o external critiques** builds refinement prompts from failed attempts only, while **w/o intra-group attempts** refines a single sampled response using its critique. Figure 6a shows that removing either component lowers the average score on every task family. On non-verifiable tasks, removing intra-group attempts and critiques reduces the average score by **12.2%** and **18.9%**, respectively; on mathematical reasoning (AIME24, AIME25, AMC23) by **3.0%** and **3.6%**; and on instruction following by **2.0%** and **3.6%**. At the level of individual benchmarks (Appendix N.1), the only exceptions are AMC23 and WildBench, where removing intra-group attempts is slightly better. The two sources are complementary: critiques provide targeted revision signals, whereas alternative attempts supply reusable partial ideas and diverse failure patterns.

6.3 Effect of Joint Optimization for Self-Refinement

Figure 6b tracks the generation and refinement rewards of GOLF during training: both rise in tandem, so joint optimization improves the two capabilities together rather than trading them off.

Table 3 evaluates GOLF on RefineBench (Lee et al., 2025) under *self-refinement* and *guided refinement*, reporting Law as an open-ended domain and Biology and Chemistry as out-of-domain STEM domains. GOLF consistently outperforms GRPO, raising the average pass rate by **+4.16** and **+14.27** in the two settings, and it also leads in every one of the 11 RefineBench domains (40.3 vs. 30.7 on average; Appendix N.5). Generation-only GRPO can even hurt refinement (*e.g.*, Law in both settings), whereas GOLF gains in every domain. The benefit also grows with scale: moving the `Qwen3` base model from 8B to 14B improves self-refinement by only +2.22, whereas GOLF training adds +11.29 at 8B and +13.17 at 14B (Table 17 in Appendix M.2).

7 Conclusion

We introduced GOLF, which aggregates group-level NL feedback, adaptively injects refinements into failure-dominated groups, and jointly optimizes generation and refinement. Across verifiable and non-verifiable tasks, GOLF achieves the best average score in every setting and converges in up to $\sim$2.3$\times$ fewer steps, while cultivating test-time self-refinement and sustaining broader exploration; ablations confirm that the two feedback sources are complementary.

AI USE STATEMENT

Apart from their role as reward models and evaluation judges in our experiments (Section 5), we used large language models only to polish the writing and to check the formatting of this paper. We did not use generative AI tools for research ideation, experimental design, or data analysis. The authors reviewed all AI-assisted edits and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work aims to make reinforcement learning of LLMs more sample-efficient by learning from natural language feedback. All training and evaluation data are publicly released datasets and benchmarks used under their original licenses, and the study involves no human subjects. Stronger refinement may also make models more effective at producing persuasive or strategically optimized content, and biases of LLM-based judges or critics may be reinforced during training. We encourage bias audits of judge feedback and careful evaluation before deploying such methods.

REPRODUCIBILITY STATEMENT

Training hyperparameters for non-verifiable tasks, mathematical reasoning, instruction following, and code generation are listed in Tables 5, 6, 8 and 9. Refinement prompts are in Appendix B, reward-model and judge prompts in Appendix H, and benchmark protocols in Appendix D. Baselines are described in Appendices E and F. Datasets are public; code will be released upon publication.

REFERENCES

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D. Chang, and Prithviraj Ammanabrolu. Critique-out-loud reward models. *CoRR*, abs/2408.11791, 2024. doi: 10.48550/ARXIV.2408.11791. URL https://doi.org/10.48550/arXiv.2408.11791.

Elie Bakouch, Leandro von Werra, and Lewis Tunstall. Open-R1: A fully open reproduction of DeepSeek-R1. https://huggingface.co/blog/open-r1, 2025.

Adithya Bhaskar, Xi Ye, and Danqi Chen. Language models that think, chat better. *CoRR*, abs/2509.20357, 2025. doi: 10.48550/ARXIV.2509.20357. URL https://doi.org/10.48550/arXiv.2509.20357.

Meng Cao, Lei Shu, Lei Yu, Yun Zhu, Nevan Wichers, Yinxiao Liu, and Lei Meng. Enhancing reinforcement learning with dense rewards from language model critic. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 9119–9138. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.515. URL https://doi.org/10.18653/v1/2024.emnlp-main.515.

Angelica Chen, Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Samuel R. Bowman, Kyunghyun Cho, and Ethan Perez. Learning from natural language feedback. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=xo3hI5MwvU.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025. URL https://arxiv.org/abs/2512.02556.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled AlpacaEval: A simple way to debias automatic evaluators. In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2404.04475.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2507.17746.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, et al. Deepseek-r1 incentivizes reasoning in LLMs through reinforcement learning. *Nat.*, 645(8081):633–638, 2025. doi: 10.1038/S41586-025-09422-Z. URL https://doi.org/10.1038/s41586-025-09422-z.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, et al. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs *Actually* correct their own mistakes? A critical survey of self-correction of llms. *Trans. Assoc. Comput. Linguistics*, 12:1417–1440, 2024. doi: 10.1162/TACL_A_00713. URL https://doi.org/10.1162/tacl_a_00713.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, et al. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CjwERcAU7w.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, et al. Tülu 3: Pushing frontiers in open language model post-training. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2411.15124.

Young-Jun Lee, Seungone Kim, Byung-Kwan Lee, Minkyeong Moon, Yechan Hwang, Jong Myoung Kim, Graham Neubig, Sean Welleck, and Ho-Jin Choi. Refinebench: Evaluating refinement capability of language models via checklists, 2025. URL https://arxiv.org/abs/2511.22173.

Ang Li, Yifei Wang, Zhihang Yuan, Stefanie Jegelka, and Yisen Wang. LANPO: bootstrapping language and numerical feedback for reinforcement learning in llms. *CoRR*, abs/2510.16552, 2025a. doi: 10.48550/ARXIV.2510.16552. URL https://doi.org/10.48550/arXiv.2510.16552.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, et al. NuminaMath. https://github.com/project-numina/aimo-progress-prize, 2024.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-Hard and BenchBuilder pipeline. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025b. URL https://proceedings.mlr.press/v267/li25h.html.

Bill Yuchen Lin, Yuntian Deng, Khyathi Raghavi Chandu, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking LLMs with challenging tasks from real users in the wild. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=MKEHCx25xp.

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. In *Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=5PAF7PAY2Y.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, et al. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Vaskar Nath, Elaine Lau, Anisha Gunjal, Manasi Sharma, Nikhil Baharte, and Sean Hendryx. Adaptive guidance accelerates reinforcement learning of reasoning models. *arXiv preprint arXiv:2506.13923*, 2025.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Samuel J Paech. Eq-bench creative writing benchmark v3. https://github.com/EQ-bench/creative-writing-bench, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=yfYgwjj5F8.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL https://doi.org/10.48550/arXiv.2303.11366.

Hanyang Wang, Lu Wang, Chaoyun Zhang, Tianjun Mao, Si Qin, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Text2Grad: Reinforcement learning from natural language feedback. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2505.22338.

Yubo Wang, Xiang Yue, and Wenhu Chen. Critique fine-tuning: Learning to critique is more effective than learning to imitate. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=vTAz44GgOA.

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 *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=vO8LLoNWWk.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=2a36EMSSTp.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2504.13837.

Xiaoying Zhang, Yipeng Zhang, Hao Sun, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-GRPO: Advancing LLM reasoning with natural language and numerical feedback. *arXiv preprint arXiv:2506.03106*, 2025. URL https://arxiv.org/abs/2506.03106.

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 *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.19807.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatGPT interaction logs in the wild. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bl8u7ZRlbM.

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

A PRELIMINARY STUDY: THE VALUE OF DIVERSE FEEDBACK

A core motivation of GOLF is that NL feedback encompasses more than explicit critiques. In practice, alternative attempts within a rollout group can also serve as implicit guidance by exposing diverse failure modes and partial solution strategies. To validate this intuition, we conduct a preliminary study addressing two hypotheses: **(H1)** external critiques improve test-time self-refinement; **(H2)** combining intra-group attempts with critiques yields additional gains by helping the model escape local optima and explore more diverse repair directions.

A.1 EXPERIMENTAL SETUP

We focus on mathematical reasoning and construct a challenging subset from OpenR1 (Bakouch et al., 2025). Specifically, we select 500 problems on which `Qwen3-8B` achieves zero Pass@4 accuracy—that is, the model fails to produce any correct solution across four independent samples per problem. For each problem, we perform refinement under different feedback conditions and evaluate using two metrics: **Pass@4**, the fraction of problems solved by at least one refined sample (out of four), and **Acc**, the overall accuracy across all 2,000 refined outputs (500 problems $\times$ 4 samples).

A.2 FEEDBACK CONDITIONS

We compare four conditions with increasing information richness:

Simple. The model receives only a binary signal indicating the current solution is incorrect, with no diagnostic information.

Intra-Feedback. The model is provided with other failed attempts from the same rollout group, enabling it to reuse partial reasoning steps and avoid repeated mistakes, but without explicit error analysis.

External-Feedback. The model receives a critique for the current response, generated by `Qwen3-235B-A22B-Instruct-2507`, which pinpoints concrete errors and suggests revisions.

Mixed Feedback. We combine intra-group attempts with external critiques, providing both targeted diagnostics and alternative reasoning traces.

A.3 RESULTS AND ANALYSIS

Table 4: Test-time refinement results on the 500 hardest math problems for `Qwen3-8B` under four feedback conditions. **Bold** marks the best per column.

Feedback type	Pass@4 (%)	Acc. (%)
Simple	0.00	0.00
Intra	18.80	6.65
External	27.60	17.00
Mixed	**30.40**	**17.55**

Table 4 summarizes the results. External feedback yields substantial improvements, lifting Pass@4 from 0% to 27.60% and Acc from 0% to 17.00%, confirming **(H1)**: explicit critiques provide strong corrective signals. Intra-group feedback alone also improves Pass@4 to 18.80%, though the gain in Acc is more modest (6.65%). This asymmetry suggests that alternative attempts primarily help broaden the search space rather than directly fixing specific errors—models may generate more diverse solutions but still struggle to identify which partial ideas are correct.

Crucially, mixed feedback achieves the best performance (30.40% Pass@4, 17.55% Acc), outperforming external feedback alone by 2.80 and 0.55 points in Pass@4 and Acc, respectively. This supports **(H2)**: the two sources are complementary. External critiques provide targeted revision directions, while intra-group attempts supply diverse reasoning fragments that help escape unproductive trajectories. Notably, the improvement in Acc (not just Pass@4) indicates that combining both sources produces more consistently correct solutions, not merely more lucky guesses.

These findings motivate our design choice to aggregate both external critiques and intra-group attempts at the group level, jointly leveraging their complementary strengths to generate higher-quality training data for policy learning.

B PROMPTS FOR SELF-REFINEMENT

In this section, we present the prompts used for model self-refinement during RL training. Depending on the task and NL feedback, we employ different prompts.

B.1 NON-VERIFIABLE TASKS

Self-Refinement Prompt for WildChat

```
Given the following inputs:

**Problem**: {original_prompt}

**Candidate Responses with Feedback**:

--- Candidate Response 1 (Score: {score_1}) ---
Response:
{response_1}

Feedback:
{critique_1}

--- Candidate Response 2 (Score: {score_2}) ---
Response:
{response_2}

Feedback:
{critique_2}

--- Candidate Response 3 (Score: {score_3}) ---
Response:
{response_3}

Feedback:
{critique_3}

... (additional candidates if provided)

Please synthesize an improved response by:

- Learning from the mistakes identified in the critiques - avoid
repeating the same errors.
- Incorporating the strengths and good aspects mentioned in the
critiques.
- Synthesizing the best parts from all candidates while addressing
their individual weaknesses.
- Fully satisfying the user instruction and meeting all requirements.

CRITICAL OUTPUT REQUIREMENTS:
- Provide ONLY the synthesized response itself, nothing more.
- DO NOT start with any meta phrases like "Improved Response:", "Here
is the synthesized response:", or similar introductory text.
- DO NOT end with any meta commentary, notes, or explanations such as
"Note: This response meets all requirements...", "This addresses the
user's needs...", or any other additional remarks.
- Your output should be the pure, direct response to the user's
original instruction - as if you were directly answering them without
any wrapper text or self-commentary.
```

B.2 VERIFIABLE TASKS

Self-Refinement Prompt for Instruction Following

```
Given the following inputs:

**User Instruction**: {original_prompt}

**Candidate Responses with Feedback**:

--- Candidate Response 1 (Score: {score_1}) ---
Response:
{response_1}

Feedback:
{critique_1}

--- Candidate Response 2 (Score: {score_2}) ---
Response:
{response_2}

Feedback:
{critique_2}

--- Candidate Response 3 (Score: {score_3}) ---
Response:
{response_3}

Feedback:
{critique_3}

... (additional candidates if provided)

Please synthesize an improved response by:

- Learning from the mistakes identified in the critiques - avoid
repeating the same errors.
- Incorporating the strengths and good aspects mentioned in the
critiques.
- Synthesizing the best parts from all candidates while addressing
their individual weaknesses.
- Fully satisfying the user instruction and meeting all requirements.

CRITICAL OUTPUT REQUIREMENTS:
- Provide ONLY the synthesized response itself, nothing more.
- DO NOT start with any meta phrases like "Improved Response:", "Here
is the synthesized response:", or similar introductory text.
- DO NOT end with any meta commentary, notes, or explanations such as
"Note: This response meets all requirements...", "This addresses the
user's needs...", or any other additional remarks.
- Your output should be the pure, direct response to the user's
original instruction - as if you were directly answering them without
any wrapper text or self-commentary.
```

Refinement Prompt for Mathematical Reasoning

```
Given the following inputs:

**Problem**: {original_prompt}

**Solution Attempts with Feedback**:

--- Solution Attempt 1 (Score: {score_1}) ---
{solution_1}
```

```
Feedback: The answer is incorrect, the ground_truth is {ground_truth}

--- Solution Attempt 2 (Score: {score_2}) ---
{solution_2}
Feedback: The answer is incorrect, the ground_truth is {ground_truth}

--- Solution Attempt 3 (Score: {score_3}) ---
{solution_3}
Feedback: The answer is incorrect, the ground_truth is {ground_truth}

... (additional attempts if provided)

Please synthesize an improved solution by:

- Carefully analyze each attempt and use the feedback to locate where
the reasoning goes wrong.
- Keep any valid steps and calculations, but fix incorrect steps with
your own correct reasoning.
- Build a complete, coherent, self-contained solution with
step-by-step derivations and necessary calculations.

CRITICAL REQUIREMENTS:
- You MUST derive the solution through genuine mathematical reasoning.
- Do NOT work backwards from the ground truth or force the final
answer.
- Every step must follow logically from the previous one and use valid
mathematical operations.

OUTPUT FORMAT:
- Start solving immediately (no preface or meta commentary).
- Do NOT mention or allude to attempts, candidates, or feedback.
- End with the final answer formatted as \boxed{}.
- Output ONLY the solution itself. Do not add any notes after the
boxed answer.
```

C Design Details of GOLF

Aggregated prompt under a token budget. The aggregated refinement prompt $p_{\text{agg}}(x)$ must fit within the maximum prompt length (Tables 5, 6, 8 and 9). We first subtract the length of the fixed template, *i.e.*, the header containing the original prompt and the instruction tail, from the maximum prompt length. We then sort the failed attempts in $\mathcal{F}(x)$ by length in ascending order and append each attempt together with its critique while the complete pair still fits into the remaining budget; attempts are never truncated. Filling the budget with the shortest attempts first maximizes the number of distinct failure patterns in the context.

Why injection is gated on all-failed groups. With binary rewards and the Dr. GRPO advantage $\hat{A} = R - \frac{1}{N}\sum R$, a group with k successes among N rollouts assigns $\hat{A} = (N-k)/N$ to each success and $\hat{A} = -k/N$ to each failure.

- $k = 0$: every advantage is zero and the on-policy term provides no gradient. Replacing one failure with a verified refinement gives the refinement $\hat{A} = (N-1)/N$ and each remaining failure $\hat{A} = -1/N$, so a learning signal is created where none existed.
- $k \geq 1$: the group already provides a signal. After replacing one failure with a refinement, the group has $k+1$ successes, and the advantage of each of the policy's own successes shrinks from $(N-k)/N$ to $(N-k-1)/N$. Injection then shifts credit from on-policy successes to an off-policy sample and dilutes the policy's own learning signal.

This motivates $\eta = 1/N$, which triggers injection exactly in the $k = 0$ case, and is consistent with the drop observed when refinements are always injected (Table 21).

Cross-context importance ratio. In Equation (3), the denominator of $\tilde{\rho}$ is the distribution from which the refinement was sampled, $\pi_{\theta_{\text{old}}}(\cdot \mid p_{\text{agg}}(x))$, as required by importance sampling. The numerator conditions on the plain prompt x because the quantity we aim to improve is the policy used at test time, which never sees feedback. Refinement tokens with positive advantage therefore increase $\pi_\theta(\cdot \mid x)$, distilling feedback-conditioned refinements into direct generation. Because the two contexts differ, $\tilde{\rho}$ can be far from one; the shaping function $f(u) = u/(u+\lambda)$ is bounded in $[0, 1)$, which keeps the off-policy term stable, and its derivative $\lambda/(u+\lambda)^2$ is largest for small u, which amplifies the gradient on tokens that are unlikely under $\pi_\theta(\cdot \mid x)$ (Yan et al., 2025).

Hyperparameters. GOLF introduces two scalars beyond the RL backbone: $\lambda = 0.1$, inherited from LUFFY (Yan et al., 2025), and the injection threshold $\eta = 1/N$. The same values are used for all task families and backbones without per-task tuning.

D BENCHMARKS

We provide detailed descriptions of the benchmarks used in our experiments.

- **AlpacaEval v2.0** (Dubois et al., 2024) contains 805 user prompts paired with reference responses. It reports a head-to-head win rate computed by a generative judge. Following the recommended protocol, we use the length-controlled win rate, while replacing the default judge with GPT-4o.
- **WildBench** (Lin et al., 2025) measures open-domain conversational ability using 1,024 user prompts, including a subset of multi-turn interactions. It is scored with instance-level rubrics that are manually checked, which helps reduce judge shortcutting. For each prompt, a candidate response is compared against a GPT-4 reference and assigned a discrete score in $\{-100, -50, 0, 50, 100\}$. We report the mean score over all instances.
- **ArenaHard v2.0** (Li et al., 2025b) contains 750 prompts in two categories: *Hard Prompts* (500) covering challenging real-world queries and *Creative Writing* (250). Evaluation follows a pairwise LLM-as-a-judge protocol, using `o3-mini-2025-01-31` and `gemini-2.0-flash-001` as reference models for the two categories, respectively. We use a GPT-4o judge with style control to mitigate potential bias.
- **CreativeWriting v3** (Paech, 2025) evaluates long-form writing under explicit constraints using 96 creative writing prompts. We compute an absolute score between 0 and 100 using GPT-4o as the judge.
- **IFEval** (Zhou et al., 2023) comprises 25 verifiable instruction types (length, keyword, format, language) with over 500 prompts. Verification is automated with Python functions, supporting strict and loose accuracy metrics.
- **IFBench** (Pyatkin et al., 2025) introduces 58 diverse constraints, curated for their novelty and coverage; all have corresponding verification code. It explicitly targets generalization by constructing evaluation prompts and constraints out of domain relative to the training data, revealing severe overfitting in prior models.

E TRAINING DETAILS FOR NON-VERIFIABLE TASKS

We provide the training hyperparameters in Table 5. Below we describe the implementation details of each baseline.

Likert-GRPO. The generative reward model is prompted to evaluate each response on a 1–10 Likert scale. The resulting scalar score is used directly as the reward for GRPO. The scoring prompt is provided in §H.3.

Pairwise-GRPO. For each prompt, we first generate a high-quality reference response using GPT-4o. The generative reward model then compares each candidate response against this reference via pairwise comparison and returns a binary reward (1 if the candidate wins, 0 otherwise). To mitigate potential position bias in the generative judge, we randomly shuffle the presentation order of the

Table 5: Key hyperparameters used for RL training on non-verifiable tasks in the veRL (Sheng et al., 2025) framework.

Category	Hyperparameter	Value
Data	Train file	WildChat-IF
	Train size	7,500
	Max prompt length	8,192
	Max response length	4,096
	Filter overlong prompts	True
Actor Model	Base model 1	`Llama-3.1-8B-Instruct`
	Base model 2	`Qwen3-8B (non-thinking)`
	LR	1×10^{-6}
	KL loss coefficient β	0.00
	Use dynamic batch size	True
Rollout	Rollout engine	vllm
	GPU mem utilization	0.8
	Train rollout n	8
	Temperature	1.0
Reward Model	RM model	`Qwen3-235B-A22B-Instruct-2507`
Trainer	PPO Mini Batch size	32
	PPO Train Batch size	64
	Critic Warmup	0
	GPUs per node	8
	Nodes	4
	Total epochs	2

candidate and reference responses for each comparison. The pairwise scoring prompt is provided in §H.1.

Rubric-GRPO. Following Gunjal et al. (2026), we first generate prompt-specific evaluation rubrics using DeepSeek-V3.2 (DeepSeek-AI, 2025), one of the strongest open-source models available at the time of our experiments. Each rubric comprises 5–10 weighted criteria spanning correctness, relevance, completeness, style, and safety, tailored to the specific prompt. The generative reward model then evaluates each response holistically against these criteria and produces a single 1–10 score, which is used as the scalar reward for GRPO. The rubric generation and scoring prompts are provided in §H.2.

Refinement-FT. Following Chen et al. (2024), we implement a two-stage pipeline: (1) for each prompt, the policy generates initial responses and receives external critiques from the generative reward model; (2) the policy generates refined responses conditioned on the original response and the critique. The model is then fine-tuned on successful refinements, i.e., those that win the pairwise comparison against a GPT-4o reference (using the same evaluation protocol as Pairwise-GRPO), with a standard supervised learning objective.

Critique-GRPO. Following Zhang et al. (2025), each failed response receives an external critique from the generative reward model. The policy then generates a refined response conditioned on the original response and the critique. The refinement is incorporated into the GRPO rollout group as an additional sample, using the same pairwise reward as Pairwise-GRPO.

F Training Details for Verifiable Tasks

This section describes the training details for verifiable tasks. Table 6 lists the training hyperparameters for mathematical reasoning tasks, and Table 8 lists the training hyperparameters for instruction-following tasks.

Table 6: Key hyperparameters used for RL training on the mathematical reasoning task in the veRL (Sheng et al., 2025) framework.

Category	Hyperparameter	Value
Data	Train file	OpenR1-filtered
	Train size	4,000
	Max prompt length	8,192
	Max response length	6,144
	Filter overlong prompts	True
Actor Model	Base model 1	`Qwen3-4B (non-thinking)`
	Base model 2	`Qwen3-8B (non-thinking)`
	LR	1×10^{-6}
	KL loss coefficient β	0.00
	Use dynamic batch size	True
Rollout	Rollout engine	vllm
	GPU mem utilization	0.8
	Train rollout n	8
	Temperature	1.0
Reward Model	RM model	`Math-Verify`
Trainer	PPO Mini Batch size	64
	PPO Train Batch size	128
	Critic Warmup	0
	GPUs per node	8
	Nodes	4
	Total epochs	10

In addition to the baselines shared with the non-verifiable setting, we include the following baseline.

Critique-FT. Following Wang et al. (2025), for each problem, the base model first generates initial responses, and the reward system produces natural language critiques for each response (using the same external critique source as GOLF). The policy is then fine-tuned to maximize the likelihood of generating these critiques conditioned on the problem and the initial response, training the model to internalize critical thinking.

For mathematical reasoning and verifiable instruction-following tasks, feedback sources differ from the non-verifiable setting, which relies on a generative reward model.

For mathematical reasoning tasks, obtaining accurate critiques of intermediate reasoning steps is both challenging and prohibitively expensive. Following Zhang et al. (2025), we therefore convert the ground-truth answer into the format of an external critique, which both signals whether the prediction is correct and provides the target answer for the model to reflect on its reasoning errors. To minimize the risk of the model taking shortcuts by directly copying the ground truth from the context during refinement, we carefully optimize the refinement prompt to discourage such behavior (see Appendix B).

LLM-Generated Critique. To examine whether replacing ground-truth answers with LLM-generated critiques preserves training effectiveness, we conduct an additional experiment using `Qwen3-235B-A22B-Instruct-2507` as a generative reward model (GRM) to produce step-level critiques for mathematical reasoning. The GRM receives the problem, the model's solution, and the ground-truth answer, and is prompted to generate a detailed critique identifying specific reasoning errors. The critique prompt is shown below.

Critique Prompt for LLM-Generated Feedback (Mathematical Reasoning)

```
You are a mathematics expert tasked with critiquing a student's
solution.

## Problem
```

```
{problem}

## Student's Solution
{solution}

## Ground-Truth Answer
{ground_truth}

## Instructions
Analyze the student's solution step by step. Identify the first
reasoning
error or logical gap. Provide a concise critique that:
1. Points out the specific step where the reasoning goes wrong.
2. Explains why the step is incorrect.
3. Suggests the correct direction without giving the full solution.

Your critique should help the student reflect on their mistake and
correct
their reasoning, not simply provide the answer.
```

Table 7 compares GOLF trained with ground-truth critiques against GOLF trained with LLM-generated critiques on three mathematical reasoning benchmarks. LLM-generated critiques consistently outperform the ground-truth baseline across all three benchmarks (+2.34 on AIME24, +1.68 on AIME25, +3.95 on AMC23), suggesting that step-level diagnostic feedback from a capable LLM provides richer supervision than simply revealing the target answer. This result indicates that the ground-truth critique used in our main experiments is a conservative choice, and that GOLF can benefit further from higher-quality NL feedback.

Blind critiques without the ground truth. To test whether GOLF depends on access to the ground-truth answer, we remove the ground-truth field from the critique prompt above, so the critic (`Qwen3-235B-A22B-Instruct-2507`) sees only the problem and the failed attempts and must diagnose the errors on its own. All other settings are unchanged, and the scalar reward is still computed by Math-Verify, as for all methods. Without ever seeing the ground truth, the blind critic still improves over GRPO by **+4.58** average points and exceeds the ground-truth-answer feedback used in our main experiments (61.81 vs. 60.29); giving the critic the ground truth adds only +1.14 (Table 7).

Table 7: Critique sources for mathematical reasoning with `Qwen3-8B`. The LLM critic is `Qwen3-235B-A22B-Instruct-2507`; the blind critic does not see the ground-truth answer. The blind-critique results are averaged over three random seeds; † marks the single-run results of Table 2.

Critique source	AIME24	AIME25	AMC23	Avg.
GRPO (no critique)†	55.05	38.02	78.61	57.23
GOLF w/ ground-truth answer†	58.49	41.65	80.74	60.29
GOLF w/ LLM critique (with ground truth)	**60.83**	**43.33**	**84.69**	**62.95**
GOLF w/ blind LLM critique (no ground truth)	59.17	42.50	83.75	61.81

Table 8: Key hyperparameters used for RL training on the instruction-following task in the veRL (Sheng et al., 2025) framework.

Category	Hyperparameter	Value
Data	Train file	IFTrain-filtered
	Train size	3,798
	Max prompt length	8,192
	Max response length	4,096
	Filter overlong prompts	True
Actor Model	Base model 1	`Qwen3-4B (non-thinking)`
	Base model 2	`Qwen3-8B (non-thinking)`
	LR	1×10^{-6}
	KL loss coefficient β	0.00
	Use dynamic batch size	True
Rollout	Rollout engine	vllm
	GPU mem utilization	0.8
	Train rollout n	8
	Temperature	1.0
Reward Model	RM model	`Verification functions in Python`
Trainer	PPO Mini Batch size	256
	PPO Train Batch size	256
	Critic Warmup	0
	GPUs per node	8
	Nodes	4
	Total epochs	15

For instruction-following tasks, the verification process is implemented through code-based constraint checking. We convert the execution results of these verification functions into natural language feedback. Below, we provide an illustrative example: the `check_following` function performs automated constraint verification, and based on its pass or fail outcome, the corresponding `get_critique` function returns explicit natural language feedback to guide the model during training.

Example: Code-Based Verification and Natural Language Critique (Unique Word Count)

```
def check_following(self, value):
    """Checks if the response contains the expected number of unique
    words."""
    words = value.lower().split()
    unique_words = set()
    for word in words:
        unique_words.add(word.strip(''.join(string.punctuation) + '
        '))
    # Convert to set to get unique words
    return len(unique_words) >= self._num_unique_words

def get_critique(self, value, passed):
    """Generate natural language feedback for unique word count
    constraint."""
    words = value.lower().split()
    unique_words = set()
    for word in words:
        unique_words.add(word.strip(''.join(string.punctuation) + '
        '))
    actual_unique = len(unique_words)
    if passed:
        return f"Constraint satisfied: The response contains
        {actual_unique} unique words (at least {self._num_unique_words}
        required)."
```

```
    else:
        return f"Constraint not satisfied: The response contains
        {actual_unique} unique words, but at least
        {self._num_unique_words} unique words are required."
```

G Training Details for Code Generation

We follow the evaluation protocol of Hübotter et al. (2026) for code generation experiments. We train on the LCBv6 subset of LiveCodeBench (Jain et al., 2025), which contains 131 contest-style coding problems released between February and May 2025. Following Hübotter et al. (2026), we use a 50%/50% public/private test split, where public tests are used for evaluation during training and private tests for validation. We report the average accuracy over 4 rollouts (Avg@4). Table 9 lists the training hyperparameters.

Table 9: Key hyperparameters used for RL training on the code generation task in the veRL (Sheng et al., 2025) framework.

Category	Hyperparameter	Value
Data	Train file	LCBv6 (LiveCodeBench)
	Train size	131
	Max prompt length	2,048
	Max response length	8,192
	Filter overlong prompts	True
Actor Model	Base model	`Qwen3-8B (non-thinking)`
	LR	1×10^{-6}
	KL loss coefficient β	0.00
	Use dynamic batch size	True
Rollout	Rollout engine	vllm
	GPU mem utilization	0.8
	Train rollout n	8
	Temperature	1.0
Reward Model	RM model	`Execution-based (public test cases)`
Trainer	PPO Mini Batch size	32
	PPO Train Batch size	32
	Critic Warmup	0
	GPUs per node	8
	Nodes	4
	Total steps	100

For the SDPO baseline, we follow the training protocol and hyperparameters described by Hübotter et al. (2026).

H LLM Judge Prompts

In this section, we present the generative reward model (GRM) scoring prompts used by different baselines on non-verifiable tasks, including pairwise comparison against a high-quality reference (§H.1), rubric-based scoring (§H.2), and Likert scoring (§H.3).

H.1 PAIRWISE SCORING PROMPT

Pairwise GRM Scoring Prompt

```
You are given a user question and two responses.
- Response A: a model-generated answer that may need improvement.
- Response B: a high-quality reference answer (for evaluation only).

Your task is to act as an impartial judge and decide which response is
better for the user.

First, think step by step and put your analysis in <reasoning> and
</reasoning> tags. In your reasoning:
- Identify the key requirements of the user question (instruction
following, relevance, completeness, factuality/safety, style/format).
- Compare Response A and Response B on these requirements.
- Response A may be better than Response B if it follows the user more
closely, is safer, or better matches the requested style.
- Avoid position bias and length bias.

Then, provide an actionable critique in <critique> and </critique>
tags **for the model-generated answer (Response A) only**. This
critique will be shown to another model that only sees that answer.
Therefore:
- Write the critique as if there is ONLY ONE answer.
- Do NOT mention or hint that there was another response, a reference
answer, "Assistant A/B", "the other answer", "the second answer", or
"the reference".
- Point out the current answer's strengths.
- Point out missing/incorrect/unsafe/irrelevant parts.
- Give concrete suggestions on what to add/remove/fix.
- Do NOT copy or paraphrase the content of Response B.

Finally, output your verdict in <answer> and </answer> tags:
- <answer> [[A]] </answer> if the model-generated answer (Response A)
is better
- <answer> [[B]] </answer> if the reference answer (Response B) is
better

Format your output EXACTLY like this:
<reasoning> your step-by-step comparison of A vs B </reasoning>
<critique>
your self-contained, neutral feedback for improving the
model-generated answer
</critique>
<answer> [[A]] or [[B]] </answer>

Below are the user's question and the two responses:

[User Question]
{instruction}

[The Start of Response A]
{response_a}
[The End of Response A]

[The Start of Response B]
{response_b}
[The End of Response B]
```

H.2 RUBRICS-AS-REWARD SCORING PROMPT

We present the prompts used for rubric generation, as well as the prompts for rubric-based scoring.

Rubric Generation Prompt

```
You are an expert rubric writer. Your job is to generate a
self-contained set of evaluation criteria (rubrics) for judging how
good a chatbot response is to a given user prompt in a WildChat-style
open-domain conversation. Rubrics can cover aspects of a response,
such as, but not limited to, factual correctness, relevance,
completeness, reasoning quality, clarity, tone, empathy, creativity
when appropriate, formatting, and safety and policy compliance. Each
rubric item must be self-contained so that a non-expert reader can
apply it without inferring hidden requirements or consulting external
information.

Input:
- prompt: The full text of the user prompt.

Total items:
- Choose 5 to 10 rubric items based on the complexity and risk of the
prompt.

Rubric item requirements:
- Each item must contain exactly three lines in the following format:
  1) <title> ... </title>
  2) <description> ... </description>
  3) <weight> ... </weight>

- title: 2 to 4 words.
- description: Exactly one sentence that begins with one of the
following category prefixes:
  - "Essential Criteria: ..."
  - "Important Criteria: ..."
  - "Optional Criteria: ..."
  - "Pitfall Criteria: Does not mention ..."
  - "Pitfall Criteria: Recommends ..."

- weight:
  - For Essential, Important, Optional use an integer 1 to 5 (5 = most
  important).
  - For Pitfall use -1 or -2.

Category guidance:
- Essential: Must have requirements or safety checks. If missing, the
response is invalid (weight 5).
- Important: Key reasoning, completeness, correctness, or clarity that
strongly affects quality (weight 3 to 4).
- Optional: Nice to have improvements in style or depth, not deal
breaking (weight 1 to 2).
- Pitfall: Common mistakes or omissions specific to this prompt. Each
Pitfall description must begin with "Pitfall Criteria: Does not
mention ..." or "Pitfall Criteria: Recommends ..." and use weight -1
or -2.

Prompt understanding and constraint extraction:
- Infer any explicit constraints from the prompt (for example: "give
me 10", "in Chinese", "be concise", "step by step", "only bullet
points") and turn them into checkable rubric items.
- Convert vague expectations into observable checks that can be
verified from the assistant response alone.
- Do not copy large blocks of the prompt into the rubric text.

Safety and policy guidance:
- If the prompt involves medical, legal, or financial decisions,
include at least one Essential item requiring appropriate caveats and
avoidance of definitive personalized high stakes directives.
```

```
- If the prompt involves self harm, violence, illegal activity, hate,
harassment, sexual content (especially minors), or privacy invasion,
include Essential items requiring refusal or safe redirection, and
Pitfall items penalizing disallowed compliance.
- If the prompt requests instructions enabling wrongdoing, include an
Essential item requiring refusal or safe alternatives.

Output format:
- Output only rubric items, with no header and no trailing commentary.
- Separate each rubric item with exactly one blank line (i.e., a
"\n\n" separator).
- Do not add extra fields or extra lines per item.

Now, given the prompt, generate the rubric as described.
```

Rubric-Based GRM Scoring Prompt

```
You are an expert evaluator. Given a user prompt, a generated
response, and a list of quality rubrics, please rate the overall
quality of the response on a scale of 1 to 10 based on how well it
satisfies the rubrics.
Consider all rubrics holistically when determining your score. A
response that violates multiple rubrics should receive a lower score,
while a response that satisfies all rubrics should receive a higher
score.
Start your response with <score> and ends with </score>. The value
should be an integer between 1 and 10.
Example response:
<score> your_integer_score_from_1-to-10 </score>

Given the following prompt, response, and rubrics, please rate the
overall quality of the response on a scale of 1 to 10 based on how
well it satisfies the rubrics.

<prompt>
{instruction}
</prompt>

<response>
{response}
</response>

<rubrics>
{rubric_list_string}
</rubrics>

Your evaluation:
```

H.3 LIKERT SCORING PROMPT

Likert GRM Scoring Prompt

```
You are given a user question and a single response from an AI
assistant. Your task is to act as an impartial judge and evaluate how
well the response fulfills the user's instructions.
```

```
Think carefully about how to assess the quality of the response, and
enclose your reasoning within <reasoning> and </reasoning> tags. Your
reasoning should include your evaluation criteria, a clear
understanding of what an ideal response would look like for this
particular question, and a concrete example of such an ideal or
reference answer if possible. Then compare the assistant's response to
your ideal or reference answer, explaining how it aligns with or
deviates from your expectations. Be specific and avoid vague or overly
general judgments. Remain as objective as possible. **Be critical and
rigorous in your evaluation-do not be lenient.**

In addition to your reasoning, provide a concise, actionable critique
of the assistant's response for improvement. The critique should (a)
highlight key strengths and weaknesses, (b) point out concrete errors,
omissions, safety or factuality issues, and (c) give clear, targeted
suggestions for fixing them. Enclose this critique within <critique>
and </critique> tags. Important: in the <critique> section, only give
analysis and modification suggestions (what to change and how to
change it). Do NOT rewrite the full answer, do NOT output a "revised"
or "improved" version of the response, and do NOT copy large spans of
the original answer.

Finally, assign the assistant's response a score from 1 to 10. **Use
strict standards and fully utilize the entire 1-10 scale.** Use
integers only (no decimals). The score distribution should be:
- 1-2: Fundamentally flawed, mostly irrelevant, or severely harmful
- 3-4: Significant issues that prevent it from being useful; major
gaps or errors
- 5-6: Partially helpful but with substantial room for improvement;
meets only basic requirements
- 7-8: Good quality with some noticeable issues or missing elements
- 9: Very good, minor issues only
- 10: Exceptional, comprehensive, and nearly perfect

**Important calibration notes:**
- **Fully utilize the 1-10 range**: Do not cluster scores in the 7-9
range. Spread your scores across the entire scale based on actual
quality.
- Scores of 9-10 should be rare and reserved for truly exceptional
responses
- Scores of 1-4 should be given when responses have fundamental
problems
- Be especially critical of: factual errors, incomplete answers, poor
reasoning, ignoring parts of the question, verbosity without
substance, lack of specificity
- Avoid grade inflation: if a response has clear deficiencies, assign
a correspondingly lower score

Choose the number that best matches your judgment after applying these
strict standards. Enclose the score within <score> and </score> tags.

Format your output like this:
<reasoning> your_thinking_process </reasoning>
<critique> your_critique (only what to fix and how to fix, no
rewritten answer) </critique>
<score> your_integer_score_from_1_to_10 </score>

Below are the user's question and the assistant's response:

[User Question]
{instruction}

[The Start of Assistant's Answer]
```

```
{response}
[The End of Assistant's Answer]
```

I Reward Model Selection: Agreement with GPT-4o

To justify our choice of `Qwen3-235B-A22B-Instruct-2507` as the generative reward model, we measure its pairwise preference agreement with GPT-4o (our main evaluation judge) on a held-out set of conversational prompts. The aim is to confirm that the reward signal during training is well-aligned with the eventual evaluation signal, while keeping the two judges separate (different model families) to avoid in-family bias.

Protocol. We sample 200 prompts from the held-out split of WildChat-IF (Zhao et al., 2024), disjoint from the training set. For each prompt, we generate one response from `Llama-3.1-8B-Instruct` and one from `Qwen3-8B`, yielding 200 preference pairs spanning a realistic quality range. Each candidate reward model and GPT-4o then independently judge every (response$_A$, response$_B$) pair using the same pairwise prompt template (Appendix H.1). We report two metrics with GPT-4o as the reference judge:

- **Pairwise agreement (%):** the fraction of pairs on which the candidate and GPT-4o pick the same winner (ties handled symmetrically).
- **Cohen's κ:** agreement after correcting for chance; $\kappa \in [0.41, 0.60]$ is conventionally considered moderate, $\kappa > 0.60$ substantial.

Results. Table 10 reports agreement for two candidate reward models from the Qwen3 family. `Qwen3-235B-A22B-Instruct-2507` achieves 85.7% pairwise agreement with GPT-4o ($\kappa = 0.71$, indicating substantial agreement), substantially outperforming the smaller `Qwen3-30B-A3B-Instruct-2507` (80.4%, $\kappa = 0.61$). This supports its use as a GPT-4o-aligned reward signal and suggests that reward-model capacity correlates with judge alignment in this regime.

Table 10: Pairwise preference agreement with GPT-4o on 200 held-out comparisons.

Reward model	Agreement (%) ↑	Cohen's κ ↑
`Qwen3-30B-A3B-Instruct-2507`	80.4	0.61
`Qwen3-235B-A22B-Instruct-2507` (used)	**85.7**	**0.71**

Implication. The agreement results validate two design choices: (i) using `Qwen3-235B-A22B-Instruct-2507` as a GPT-4o-aligned proxy reward during training is reasonable, and (ii) keeping the evaluation judge in a different family (GPT-4o) preserves an independent measurement channel rather than collapsing training-time and evaluation-time judges. Together, these reduce the risk of reward hacking and same-judge inflation in our reported main results.

J Setup for Reward Model Capacity Sensitivity

Configuration. We rerun GOLF and Pairwise-GRPO on `Llama-3.1-8B-Instruct` with the reward model swapped from `Qwen3-235B-A22B-Instruct-2507` (∼22B active parameters, used in Section 5.1) to the substantially weaker `Qwen3-30B-A3B-Instruct-2507` (∼3B active parameters). All other hyperparameters—training data, batch size, learning rate, KL coefficient, number of training steps, evaluation protocol—are held identical to the main setting of Section 5.1.

Scope of modification. Both methods receive their scalar reward from the new (weaker) reward model. For GOLF, the same model also generates the per-rollout critique used by the group-level refinement step. Importantly, the evaluation judge (GPT-4o) is unchanged, so any drop in measured score reflects a weaker policy trained under the weaker reward model, not a different scoring scale.

Table 11: Full per-benchmark breakdown of the reward-model capacity ablation on `Llama-3.1-8B-Instruct`.

Method	AlpacaEval v2.0 LC win rate (%)	WildBench Score	ArenaHard v2.0 Win rate (%)	CreativeWriting v3 Score	Avg.
Reward model: `Qwen3-235B-A22B-Instruct-2507`					
Pairwise-GRPO	43.19	25.54	13.30	62.95	36.25
GOLF	**69.67**	**34.42**	**25.03**	**66.21**	**48.83**
Reward model: `Qwen3-30B-A3B-Instruct-2507`					
Pairwise-GRPO	39.83	21.16	9.27	58.78	32.26
GOLF	**65.84**	**33.17**	**22.43**	**64.72**	**46.54**

Limitations. This robustness check varies reward-model capacity within the same model family (Qwen3) to isolate capacity from judgment-style differences. Cross-family RM swaps (e.g., Llama- or Mistral-based judges) and significantly weaker RMs ($\leq$1B parameters) are left to future work.

K LENGTH BIAS ANALYSIS

Response-length inflation is a known confound in LLM-as-a-judge evaluation: a longer response can win simply by appearing more thorough, regardless of true quality. We verify that the gains reported in Section 5.1 are not driven by such length gaming.

Table 12: Length bias analysis on AlpacaEval v2.0 (`Llama-3.1-8B-Instruct`).

Method	Avg Tokens	Raw Win Rate (%)	LC Win Rate (%)
Pairwise-GRPO	939	45.47	43.19
GOLF	1,143	71.80	**69.67**

As shown in Table 12, GOLF produces responses that are 22% longer than Pairwise-GRPO (1,143 vs. 939 tokens). The length-controlled (LC) win rate on AlpacaEval v2.0 explicitly penalizes verbosity: a method that games length would see its LC win rate drop well below its raw win rate. Both methods exhibit only a modest LC penalty—Pairwise-GRPO drops from 45.47 to 43.19 (-2.28) and GOLF from 71.80 to 69.67 (-2.13)—indicating that the additional length is accompanied by proportional quality gains. Crucially, GOLF's LC win rate (69.67) still surpasses Pairwise-GRPO's raw win rate (45.47) by a wide margin, confirming that the gains are predominantly quality-driven.

L SETUP FOR FEEDBACK GRANULARITY ON INSTRUCTION FOLLOWING

Granularity levels. On instruction-following tasks (IFEval / IFBench), each prompt comes with a set of n verifiable constraints. The programmatic verifier returns, per constraint, a pass/fail bit and the verifier-internal state (e.g., observed counts, matched patterns). We define three levels of natural-language feedback constructed from this verifier output, which differ only in the refinement-time prompt fed to the policy in GOLF; the underlying scalar reward (used for advantage computation) is identical across all three settings.

- **Coarse.** The feedback is a single number: "k/n constraints satisfied."
- **Medium.** The feedback names which constraints failed but provides no diagnosis: e.g., "Failed: `constraint_3`, `constraint_5`."
- **Fine.** The feedback uses the code-to-NL conversion template in Appendix F to translate each failed constraint's verifier code into a natural-language explanation, e.g., "Constraint not satisfied: response contains X unique words but at least Y are required." This is the level used in the main GOLF experiments.

We illustrate the three granularity levels using the unique word count constraint from Appendix F. The `get_critique` function below accepts a `level` parameter and returns feedback of increasing

specificity: **coarse** provides no per-constraint detail (aggregated at the prompt level), **medium** gives a human-readable description of what is violated without specific counts, and **fine** includes the observed and required values.

Example: Multi-Granularity Critique for Unique Word Count Constraint

```
def get_critique(self, value, passed, level="fine"):
    """Generate natural language feedback at the specified
    granularity."""
    if level == "coarse":
        return None  # aggregated at prompt level as "k/n constraints
        satisfied"

    if level == "medium":
        if passed:
            return "Constraint satisfied: The response contains the
            expected number of unique words."
        else:
            return "Constraint not satisfied: The response does not
            contain the expected number of unique words."

    # level == "fine"
    words = value.lower().split()
    unique_words = set()
    for word in words:
        unique_words.add(word.strip(''.join(string.punctuation) + '
        '))
    actual_unique = len(unique_words)
    if passed:
        return f"Constraint satisfied: The response contains
        {actual_unique} unique words (at least {self._num_unique_words}
        required)."
    else:
        return f"Constraint not satisfied: The response contains
        {actual_unique} unique words, but at least
        {self._num_unique_words} unique words are required."
```

Configuration. We use `Qwen3-8B` as the policy, with the same IFTrain / IFEval / IFBench protocols as in Section 5.2, training each granularity setting from the same initialization and for the same number of steps. The reward model and KL coefficient are unchanged.

Results. Figure 3c reports IFEval / IFBench scores for GOLF under each granularity. Performance improves monotonically from coarse to fine (IFEval: 85.76 $\rightarrow$ 86.88 $\rightarrow$ 87.80; IFBench: 36.50 $\rightarrow$ 37.50 $\rightarrow$ 38.33), indicating that GOLF's group-level aggregation extracts proportionally more value from richer verifier signal.

M VARIANCE ACROSS SEEDS AND SCALING TO 14B

M.1 VARIANCE ACROSS RANDOM SEEDS

Given the many backbone–baseline combinations, Tables 1 and 2 report single training runs. For the key comparisons, we train with three random seeds using the same recipe and report mean $\pm$ std in Tables 13 and 14. GOLF outperforms every baseline on every benchmark in both regimes: on non-verifiable tasks it reaches 47.78 ± 1.19 versus 36.52 ± 0.71 for Pairwise-GRPO, and on mathematical reasoning 60.14 ± 0.73 versus 57.77 ± 0.71 for Critique-GRPO and 56.92 ± 0.27 for GRPO. Under seed averaging, Critique-GRPO also gains more over GRPO (+0.85) than in single runs.

Table 13: Key non-verifiable comparison (`Llama-3.1-8B-Instruct`; mean ± std over three seeds).

Method	AlpacaEval v2.0 LC win rate (%)	WildBench Score	ArenaHard v2.0 Win rate (%)	CreativeWriting v3 Score	Avg.
Pairwise-GRPO	44.84 ± 1.43	25.30 ± 2.67	12.02 ± 1.48	63.92 ± 1.60	36.52 ± 0.71
GOLF	**66.72** ± 2.71	**34.24** ± 2.36	**23.48** ± 1.36	**66.70** ± 1.90	**47.78** ± 1.19

Table 14: Key mathematical-reasoning comparison (`Qwen3-8B`; mean ± std over three seeds).

Method	AIME24	AIME25	AMC23	Avg.
GRPO	53.89 ± 1.05	39.17 ± 1.10	77.71 ± 0.90	56.92 ± 0.27
Critique-GRPO	56.25 ± 0.83	38.61 ± 0.64	78.44 ± 0.83	57.77 ± 0.71
GOLF	**58.33** ± 1.67	**40.42** ± 1.10	**81.67** ± 0.95	**60.14** ± 0.73

M.2 SCALING TO QWEN3-14B

To test whether the benefits persist beyond 4B–8B policies, we train `Qwen3-14B` with exactly the same recipes, hyperparameters, and evaluation protocols as in Tables 1 and 2, without re-tuning, and compare with the key baselines (mean over three random seeds; Tables 15 and 16). Figure 4 in the main text summarizes the trend across policy sizes. The margin does not shrink with scale: on non-verifiable tasks it grows over Pairwise-GRPO from +2.15 (8B) to +3.50 (14B) and stays stable over Critique-GRPO (+2.82 to +2.87); on verifiable tasks the margin over GRPO is +3.17 (4B), +2.98 (8B), and +4.04 (14B). At 14B, GOLF ranks first on all nine benchmarks.

We also evaluate self-refinement on RefineBench with the `Qwen3` backbones (Table 17). Scaling the base model from 8B to 14B improves self-refinement only marginally (+2.22), whereas GOLF training improves it by +11.29 at 8B and +13.17 at 14B. We attribute the growing margin to the stronger backbone's greater capacity for self-refinement: from the same aggregated critiques and attempts, it produces higher-quality refinements, which in turn serve as stronger scaffolds.

Table 15: Non-verifiable benchmarks with `Qwen3-14B` (same recipe and protocol as Table 1; mean over three random seeds). Best result per column in **bold**.

Method	AlpacaEval v2.0 LC win rate (%)	WildBench Score	ArenaHard v2.0 Win rate (%)	CreativeWriting v3 Score	Avg.
Pairwise-GRPO	70.63	68.52	51.24	70.12	65.13
Critique-GRPO	71.64	66.78	53.33	71.28	65.76
GOLF	**74.26**	**70.89**	**56.45**	**72.92**	**68.63**

Table 16: Verifiable benchmarks with `Qwen3-14B` (same recipe and protocol as Table 2; mean over three random seeds). Best result per column in **bold**.

	Mathematical reasoning			Instruction following		
Method	AIME24	AIME25	AMC23	IFBench	IFEval	Avg.
GRPO	57.10	42.91	82.50	38.67	88.73	61.98
GOLF	**61.67**	**47.08**	**86.56**	**42.33**	**92.46**	**66.02**

Table 17: RefineBench self-refinement pass rate (%) for `Qwen3` base models and GOLF (same protocol as Table 3; mean over three random seeds).

Model	Biology	Chemistry	Law	**Avg.**
`Qwen3-8B` (base)	15.72	12.33	25.68	17.91
`Qwen3-8B` + GOLF	29.78	30.25	27.56	29.20
`Qwen3-14B` (base)	18.23	14.15	28.02	20.13
`Qwen3-14B` + GOLF	**32.67**	**36.89**	**30.34**	**33.30**

N ADDITIONAL ANALYSES

This section presents additional analyses, including ablations on feedback sources (§N.1), adaptive guidance design choices (§N.2), entropy and exploration diversity (§N.3), per-benchmark step efficiency curves (§N.4), RefineBench on all domains (§N.5), and a case study (§N.6).

N.1 ABLATION STUDY ON FEEDBACK SOURCES

Table 18: Feedback-source ablation on non-verifiable benchmarks (`Llama-3.1-8B-Instruct`). Best and second-best results per column are in **bold** and underlined.

Method	**AlpacaEval v2.0** LC win rate (%)	**WildBench** Score	**ArenaHard v2.0** Win rate (%)	**CreativeWriting v3** Score	**Avg.**
GOLF	**69.67**	34.42	**25.03**	**66.21**	**48.83**
w/o intra-group attempts	52.70	**37.30**	20.60	60.94	42.89
w/o external critiques	51.00	33.91	17.67	55.91	39.62

Table 19: Feedback-source ablation on verifiable benchmarks (`Qwen3-8B`). Best and second-best results per column are in **bold** and underlined.

	Mathematical reasoning			**Instruction following**		
Method	AIME24	AIME25	AMC23	IFBench	IFEval	**Avg.**
GOLF	**58.49**	**41.65**	80.74	**38.33**	**87.80**	**61.40**
w/o intra-group attempts	53.02	40.30	**82.06**	36.67	86.88	59.79
w/o external critiques	55.64	39.37	79.43	35.67	85.95	59.21

N.2 ABLATION STUDY ON ADAPTIVE GUIDANCE

N.2.1 ABLATION FOR MIXED POLICY OPTIMIZATION

To assess the effectiveness of mixed-policy optimization, we compare *mixed-policy RL*—which injects high-quality refinements into the on-policy group $\mathcal{G}_{\text{gen}}(x)$ and optimizes the policy with Eq. 2—against *Supervised Fine-Tuning (SFT)*, which trains the policy to imitate the same refinement samples with a supervised learning objective. Experiments on non-verifiable benchmarks in Figure 7 (left) show that *mixed-policy RL* outperforms *SFT* by **+15.35** average points (48.83 vs. 33.48), winning on three of the four benchmarks and performing on par on WildBench (34.42 vs. 34.62; Table 20). Moreover, we track the fraction of all-zero-reward groups during training. As shown in Figure 7 (right), when applying mixed-policy optimization, GOLF markedly reduces the frequency of all-zero groups by 31.97 percentage points, thereby yielding usable policy gradients. In contrast, SFT improves imitation of refinement outputs but does not reliably propagate these improvements to on-policy exploration, leading to slower recovery from low-reward regimes.

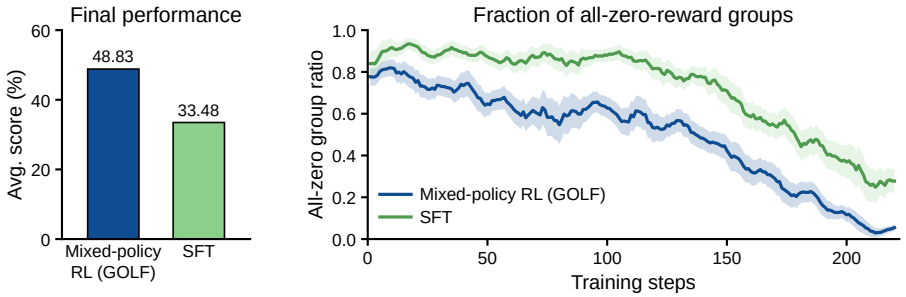


Figure 7: Left: mixed-policy RL vs. SFT on non-verifiable benchmarks (full results in Table 20). Right: fraction of all-zero-reward generation groups during training.

Table 20: Mixed-policy RL vs. SFT on the injected refinements (`Llama-3.1-8B-Instruct`). Best result per column in **bold**.

Method	AlpacaEval v2.0 LC win rate (%)	WildBench Score	ArenaHard v2.0 Win rate (%)	CreativeWriting v3 Score	Avg.
GOLF (mixed-policy RL)	**69.67**	34.42	**25.03**	**66.21**	**48.83**
GOLF (SFT, coef. 0.1)	39.03	**34.62**	15.87	44.41	33.48

N.2.2 Ablation for Adaptive Injection

A key design choice in GOLF is when to inject high-quality refinements derived from NL feedback into the rollout group. We compare our *adaptive injection* strategy, which is triggered only when the generation group $\mathcal{G}_{\text{gen}}$ is in a low-reward regime, with an *always injection* variant that injects the highest-reward refinements for each rollout group. The experiments in Table 21 are conducted on `Llama-3.1-8B-Instruct`, where "w/ adaptive" denotes the full GOLF configuration and "w/o adaptive" unconditionally selects the response with the highest reward from the refinement samples for injection at each step.

As shown in Table 21, adaptive injection yields the best results, outperforming the *always injection* strategy by **+15.29** points (48.83 vs. 33.54). Intuitively, it allocates off-policy guidance to the prompts where GRPO is most likely to suffer from collapsed group-normalized advantages (e.g., all-zero groups), thereby converting previously uninformative groups into ones with usable gradients. In contrast, always injecting may dilute the benefit of scaffolding, leading to weaker overall gains.

Table 21: Adaptive vs. always injection (`Llama-3.1-8B-Instruct`); best per column in **bold**.

Method	AlpacaEval v2.0 LC win rate (%)	WildBench Score	ArenaHard v2.0 Win rate (%)	CreativeWriting v3 Score	Avg.
GOLF (adaptive injection)	**69.67**	**34.42**	**25.03**	**66.21**	**48.83**
GOLF (always inject)	40.73	23.78	15.80	53.84	33.54

Intermediate thresholds. We further train GOLF with $\eta \in \{2/N, 3/N\}$ under the setting of Table 21. Performance decreases monotonically as η grows (Table 22), consistent with the advantage analysis in Appendix C: at $\eta = 1/N$, injection only occurs in all-zero-reward groups, where it creates a learning signal from nothing, whereas a larger η increasingly injects into groups that already contain successes and dilutes the policy's own signal. All gated settings still outperform Pairwise-GRPO (36.25); only ungated always-injection falls below it, so GOLF does not hinge on precise tuning of η.

Table 22: Sensitivity to the injection threshold η: average score over the four non-verifiable benchmarks on `Llama-3.1-8B-Instruct`. Results for $\eta \in \{2/N, 3/N\}$ are averaged over three random seeds; † marks the single-run results of Table 21 (the three-seed mean of the default is 47.78 ± 1.19, Table 13). Pairwise-GRPO scores 36.25.

η	$1/N$ (default)†	$2/N$	$3/N$	always†
Avg. score	**48.83**	45.90	43.39	33.54

N.2.3 Ablation for Efficiency

Joint optimization over generation and refinement doubles the number of rollouts per prompt in GOLF, increasing training-time compute. We therefore compare against a rollout-matched baseline (N=16) to match the total number of sampled trajectories. Table 23 shows that GOLF remains better, suggesting the improvement comes from feedback-guided refinement rather than additional sampling.

Table 23: Rollout-matched comparison with `Llama-3.1-8B-Instruct` on non-verifiable benchmarks. Pairwise-GRPO with N=16 matches the rollouts of GOLF with N=8; best per column in **bold**.

Method	**AlpacaEval v2.0** LC win rate (%)	**WildBench** Score	**ArenaHard v2.0** Win rate (%)	**CreativeWriting v3** Score	**Avg.**
Pairwise-GRPO (N=8)	43.19	25.54	13.30	62.95	36.25
Pairwise-GRPO (N=16)	46.66	28.62	14.10	64.21	38.40
GOLF (N=8)	**69.67**	**34.42**	**25.03**	**66.21**	**48.83**

Per-step training cost. Table 24 reports the per-step wall-clock time on `Llama-3.1-8B-Instruct`, measured end to end on the same hardware. GOLF (N=8) incurs a 1.39× per-step overhead relative to GRPO (N=8), which is smaller than the 1.95× cost of the rollout-matched GRPO (N=16). Combined with the 2.3× step efficiency shown in Section 6.1, GOLF reaches the baseline's final performance with less total wall-clock time.

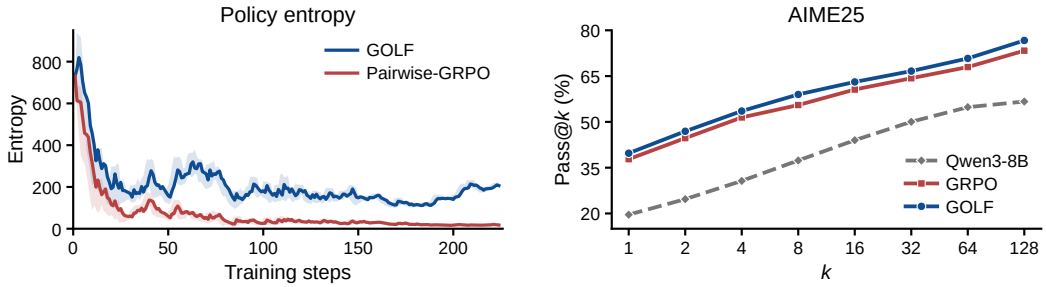
Table 24: Per-step wall-clock time on `Llama-3.1-8B-Instruct` with WildChat prompts, measured end to end on the same hardware. GOLF samples N generation and N refinement rollouts.

Method	**Rollouts per prompt**	**Seconds per step**	**Relative cost**
GRPO (N=8)	8	155.3	1.00×
GRPO (N=16)	16	302.2	1.95×
GOLF (N=8)	8 + 8	215.7	1.39×

N.3 Entropy Analysis and Exploration Diversity

Entropy has long been regarded as a proxy for exploration in policy optimization. To assess the impact of GOLF on exploratory behavior, we track the policy entropy of `Llama-3.1-8B-Instruct` throughout RL training on non-verifiable tasks. Figure 8a shows that GOLF maintains consistently higher entropy than the Pairwise-GRPO baseline. While the baseline exhibits a rapid entropy collapse, GOLF stabilizes at a substantially higher level and displays recurrent entropy surges over training, suggesting sustained exploration rather than premature mode collapse. This trend aligns with our design: aggregating intra-group attempts exposes complementary partial ideas and diverse failure modes, helping the policy escape local optima and continue exploring diverse trajectories.

Figure 8b further reports Pass@k on AIME25 for `Qwen3-8B`. Consistent with the AMC23 results in Section 5.2, GOLF maintains a clear advantage over GRPO across all k, confirming that group-level feedback broadens the set of correct solution trajectories discovered during exploration.



(a) Policy entropy (`Llama-3.1-8B-Instruct`) (b) Pass@k on AIME25 (`Qwen3-8B`)

Figure 8: GOLF maintains higher entropy and broader exploration than the baselines. (a) GOLF sustains higher policy entropy than Pairwise-GRPO throughout training. (b) GOLF covers more correct solutions for all k.

N.4 PER-BENCHMARK STEP EFFICIENCY CURVES

Figure 5a presents step-efficiency curves on AlpacaEval v2.0 and LCBv6 as representative benchmarks. For completeness, Figure 9 reports the same comparison on the remaining benchmarks. The top row shows non-verifiable tasks (WildBench and ArenaHard v2.0) on `Llama-3.1-8B-Instruct`; the bottom row shows verifiable tasks (AIME24 and IFBench) on `Qwen3-8B`. The trend is consistent across all benchmarks: GOLF converges faster and reaches a higher final plateau. As in Figure 5, curves show intermediate checkpoints; the main tables report the final one.

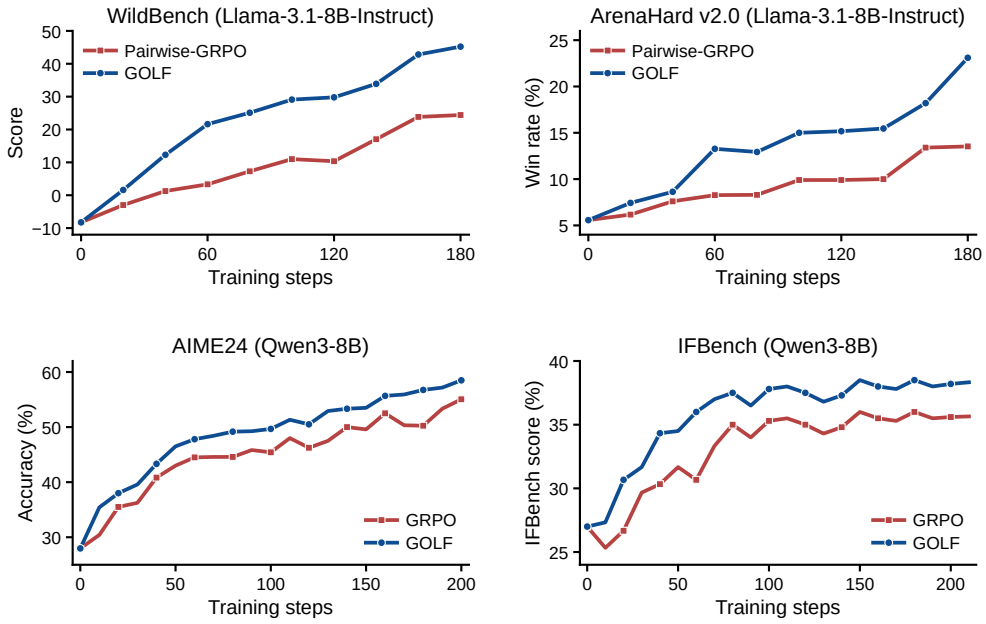


Figure 9: Step-efficiency curves on additional benchmarks. Top row: WildBench and ArenaHard v2.0 (non-verifiable, `Llama-3.1-8B-Instruct`). Bottom row: AIME24 and IFBench (verifiable, `Qwen3-8B`).

N.5 REFINEBENCH ON ALL DOMAINS

Table 3 reports three domains because of evaluation cost: Law as a representative open-ended domain, and Biology and Chemistry as out-of-domain STEM domains. Table 25 extends the guided-refinement evaluation to all 11 RefineBench domains. GOLF improves over the base model and outperforms

GRPO in every domain (average 40.3 vs. 30.7 for GRPO and 19.1 for the base model). As on Law, GRPO also falls slightly below the base model on Math and Statistics, which may partly reflect that our training prompts concentrate on non-verifiable conversational domains.

Table 25: Guided refinement on all 11 RefineBench domains with `Llama-3.1-8B-Instruct` (pass rate, %). n: problems per domain; Biology, Chemistry, and Law are the domains of Table 3. Best per row in **bold**.

Domain	n	**Base**	**+ GRPO**	**+ GOLF**
Math	321	7.3	6.8	**12.5**
Humanities / social science	185	10.4	29.8	**41.7**
Statistics	163	10.3	9.8	**14.5**
Law	142	49.3	45.1	**50.0**
Physics	69	9.3	15.2	**17.6**
CS / AI	51	15.8	25.6	**29.3**
Chemistry	18	27.8	55.5	**66.7**
Other	17	25.6	50.9	**66.3**
Engineering	14	13.6	25.8	**30.5**
Biology / medicine	11	18.2	27.8	**54.5**
Economics	9	22.8	45.7	**59.8**
Average		19.1	30.7	**40.3**

N.6 CASE STUDY

Case Study: Group-Level NL Feedback for Refinement

User prompt: Give me 10 reasons why AI should not replace humans.

Candidate Response 1 (intra-sample).

1. Loss of human touch and empathy.
2. Complexity and nuance of human judgment.
3. Creativity and originality.
4. Job diversity and adaptability.
5. Emotional labor and social roles.
6. Error tolerance and human oversight.
7. Learning and improveability.
8. Contextual understanding and common sense (“hallway corridor” problems).
9. Value alignment and ethics.
10. Breadth of human capabilities.

Critique for Candidate 1. Strong coverage of empathy, creativity, judgment, and ethics, but several points overlap (judgment, common sense, ethics) and others are redundant (adaptability vs learning). The “hallway corridor” reference is unclear. Ensure each reason is distinct and use clearer examples.

Candidate Response 2 (intra-sample).

1. Lack of emotional intelligence.
2. Creativity and innovation.
3. Contextual understanding.
4. Decision-making in uncertainty.
5. Human judgment and ethics.
6. Physical and emotional labor.
7. Accountability and responsibility.
8. Customization and personalization.
9. Human touch in service industries.
10. Job creation and economic benefits.

Critique for Candidate 2. Clear and relevant overall, but there is redundancy (emotional intelligence vs human touch in service). The "physical and emotional labor" point drifts toward misuse rather than explaining irreplaceable human qualities. Some items could be tightened to stay directly on the question.

Candidate Response 3 (intra-sample).

1. Lack of empathy and human connection.
2. Contextual understanding and common sense.
3. Creativity and innovation.
4. Decision-making and accountability.
5. Flexibility and adaptability.
6. Emotional intelligence and moral judgment.
7. Learning and development.
8. Social and cultural understanding.
9. Critical thinking and problem-solving.
10. Human rights and dignity.

Critique for Candidate 3. Well written and coherent, but it misses a major real-world concern: socioeconomic impact of automation (job displacement, inequality). Some points overlap (accountability vs moral judgment). Broaden domain coverage while keeping reasons distinct.

Candidate Response 4 (intra-sample).

1. Loss of creativity and originality.
2. Emotional intelligence and empathy.
3. Critical thinking and problem-solving.
4. sj Job fulfillment and purpose.
5. Accountability and responsibility.
6. Physical and mental health.
7. Social interactions and human connection.
8. Continuous learning and adaptation.
9. Local and contextual knowledge.
10. Value-laden decision-making.

Critique for Candidate 4. Good breadth and structure, but contains a visible typo ("sj") that hurts professionalism. Several items are conceptually close (critical thinking, learning, value-laden decisions). Tighten wording and ensure each reason covers a distinct dimension.

Group-Level Aggregated Feedback (from 4 candidates + critiques).

- Fix surface errors and unclear references (e.g., remove "sj", avoid undefined terms like "hallway corridor").
- Reduce redundancy by separating closely related themes (empathy vs social connection; ethics vs accountability; learning vs adaptability).
- Improve coverage by adding a core socioeconomic dimension (job displacement, inequality, societal stability) that is missing in some candidates.
- Keep each reason distinct and directly tied to why replacement is undesirable; tighten overly long explanations.

Refinement (synthesized using group-level NL feedback).

1. **Empathy and care:** Many roles require genuine empathy, comfort, and trust building that AI cannot authentically provide.
2. **Accountability:** High-stakes decisions demand accountable agents who can explain choices and bear responsibility for harm.
3. **Human values and ethics:** Real decisions involve value trade-offs (fairness, rights, dignity) that require human moral judgment and societal legitimacy.

4. **Context and common sense:** Humans reliably interpret implied meaning, social cues, and situational context that remains brittle for AI.
5. **Creativity and originality:** Human creativity produces novel perspectives and breakthroughs beyond pattern recombination.
6. **Adaptation to novelty:** Humans can rapidly reframe goals and operate under shifting constraints without retraining on massive datasets.
7. **Social connection:** Human-to-human relationships, mentorship, and community support are foundational and not substitutable by automation.
8. **Economic and societal stability:** Large-scale replacement risks job displacement, widening inequality, and downstream social instability.
9. **Cultural and local understanding:** Humans navigate cultural norms and local realities that matter in education, law, and public services.
10. **Meaning and purpose:** Work and contribution are central to human well-being; broad replacement can erode purpose and participation.

O LIMITATIONS

Jointly training refinement introduces extra per-step compute for refinement generation and reward evaluation (1.39× per step; Table 24); in our experiments this overhead is compensated by the faster convergence of GOLF (Section 6.1). GOLF relies on the quality of the natural language feedback: for non-verifiable tasks it depends on a capable generative reward model (Appendix J), and for mathematical reasoning our main experiments use the ground-truth answer as a simplified critique (Appendix F). Tables 1 and 2 report single training runs; the key comparisons are repeated with three random seeds in Appendix M.1. Finally, our policies range from 4B to 14B parameters (Appendix M.2); benefits at substantially larger scales remain to be tested.