GROUPWISE AGENTIC GRADING AND ADVANTAGE REDISTRIBUTION FOR CODE AGENT RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) for code agents often uses executable tests to provide binary rewards. With these rewards, Group Relative Policy Optimization (GRPO) assigns identical advantages to test-passing trajectories within each rollout group, overlooking differences in implementation quality and adherence to task requirements. This leaves the policy without a learning signal that favors clean, targeted implementations over those containing unnecessary or out-of-scope changes. We introduce GAGAR, a framework for quality-aware credit redistribution in code agent RL. Built on dynamic sampling that retains groups containing both passing and failing trajectories, GAGAR places all trajectories from each group in a shared workspace, where an SFT-trained agentic grader jointly inspects them and ranks the test-passing candidates. Based on this ranking, we downweight lower-ranked trajectories and proportionally rescale the advantages of all test-passing trajectories to restore their original sum. This sum-preserving redistribution retains the relative weights established by quality-based downweighting while shifting credit toward higher-quality implementations. We evaluate GAGAR at industrial scale using pre-RL SFT checkpoints of MiMo-V2.6-Flash (310B total parameters) and MiMo-V2.6-Pro (1.02T total parameters). Controlled code-only Flash experiments show improved code agent performance, reduced trajectory-length growth, and more stable training. We further apply GAGAR in large-scale mixed-task RL with both Flash and Pro. Our results support combining test-based verification with groupwise agentic grading to improve the quality and stability of code agent RL.

1 INTRODUCTION

Reinforcement learning (RL) with executable feedback provides a scalable approach to training code agents that inspect repositories, modify code, and validate their solutions over long interactions. In a common setup, each trajectory receives a binary reward according to whether its submitted implementation passes the tests.

With group-relative advantage estimation, as used in GRPO (Shao et al., 2024), test-passing trajectories within the same rollout group receive identical outcome advantages. This provides a clear signal for learning to solve a task, but leaves an important question unanswered: *among successful trajectories, which ones deserve stronger reinforcement?*

Passing the same tests does not imply equal implementation quality or equal practical value to developers. One solution may address the root cause with a focused change that follows repository conventions, while another introduces unnecessary complexity, weakens validation, or modifies code outside the requested scope. These differences shape the developer experience: they affect how much review and rework are needed before a patch can be accepted and merged, and how easily the resulting code can be maintained. Successful trajectories also differ in how effectively they gather evidence, identify a solution, and validate their changes. Binary feedback overlooks these distinctions when the test outcomes are identical. Our objective is to reinforce sound, efficient problem-solving strategies and precise, minimally invasive implementations that fully satisfy task requirements. In doing so, we aim to produce maintainable, merge-ready code and provide a reliable, low-friction user experience.

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.

Turning these distinctions into useful training signals requires both reliable assessment and appropriate credit allocation. Comparing successful trajectories from the same task can reveal unnecessary complexity or ineffective strategies that are difficult to recognize in isolation. We therefore adopt groupwise assessment rather than scoring trajectories independently. Reliable comparison also requires repository context and execution evidence that static, chat-based assessments may miss, motivating an agentic evaluator that can inspect code and run checks. Finally, simply downweighting lower-quality passes reduces the group's total positive advantage while leaving negative advantages unchanged. We therefore seek to redistribute credit among successful trajectories rather than merely weaken their overall positive training signal.

We introduce GAGAR (**Groupwise Agentic Grading for Advantage Redistribution**), a quality-aware framework for code agent RL. For each mixed-outcome rollout group, an agentic grader receives a shared workspace containing the task specification, repository, submitted patches, execution results, and trajectories. It can inspect code and run targeted checks before ranking the test-passing candidates, allowing ties when the evidence is inconclusive. The grader compares different implementations for the same task to identify unnecessary changes and potential problems. The resulting ranking reflects relative quality within the group and guides credit redistribution from lower-quality to higher-quality implementations.

The grader ranks passing solutions along five dimensions: suitability of the solution approach, implementation precision, minimality of changes, avoidance of unintended side effects, and consistency with codebase conventions. GAGAR uses these rankings to downweight lower-quality passing trajectories, then proportionally rescales the advantages of all passing trajectories. In the sum-preserving formulation, this restores their original total positive advantage while retaining the ranking-based relative weights and leaving failed-trajectory advantages unchanged.

Our primary controlled study evaluates GAGAR in code-only RL initialized from a pre-RL SFT checkpoint of MiMo-V2.6-Flash (310B total, 15B active parameters) on DeepSWE v1.1 (Huang et al., 2026) and SWE-bench Pro. Compared with binary-reward training, GAGAR improves later-stage DeepSWE pass rates and training stability while reducing interaction turns and token usage on both benchmarks. We further assess implementation quality and problem-solving behavior through a blinded rubric-based evaluation, reporting average win rates among test-passing solutions. We also examine how sum-preserving redistribution relates to credit balance and training stability. We further apply GAGAR in large-scale mixed-task RL with both Flash and MiMo-V2.6-Pro (1.02T total, 42B active parameters), starting from their respective pre-RL SFT checkpoints. After mixed-task RL, the DeepSWE v1.1 `avg@3` scores reach 67.9 and 71.9 for Flash and Pro, respectively (Section 4.5).

In summary, our contributions are:

- A grading framework that combines groupwise grading with agentic repository inspection and execution checks to identify quality differences that isolated, text-only assessments may miss.
- A quality-aware, zero-sum advantage redistribution formulation that shifts credit from lower- to higher-quality solutions while preserving total positive advantage and its balance with negative advantages.
- Industrial-scale validation from pre-RL SFT checkpoints of MiMo-V2.6-Flash and MiMo-V2.6-Pro, demonstrating improvements in task performance, efficiency, and training stability.

2 Related Work

Reinforcement Learning For LLM Agents. Reinforcement learning improves agents' multi-step decision-making through feedback from interactions with external environments. Agent Lightning decouples agent execution from RL training (Luo et al., 2025), while HiPER assigns credit across planning and execution (Peng et al., 2026). GRPO estimates group-relative advantages (Shao et al., 2024), and DAPO filters uniform-outcome groups through dynamic sampling (Yu et al., 2025). We build on this setting to distinguish implementation quality among test-passing coding trajectories.

Reward Modeling And Shaping. Reward modeling and shaping can enrich training feedback beyond task success by assessing solution quality and problem-solving behavior. ReCode adds

candidate-wise process rewards for passing solutions (Fan et al., 2026), and TRIAGE supplies segment-level supervision (Xu et al., 2026), whereas we compare complete implementations within a task. Performance-based rewards also target runtime efficiency (Feng et al., 2026). CPO (Ye et al., 2025) and GRRM (Yang et al., 2026) use comparative evaluation for dialogue and translation, respectively. Agent-as-a-Judge evaluates task artifacts through agentic inspection (Zhuge et al., 2025). Groupwise Ranking Reward ranks verifier-passed multimodal reasoning with a single-pass, text-based grader (Jia et al., 2026). Our comparisons instead use interactive inspection of coding trajectories, patches, and repositories, including targeted execution checks.

Credit Assignment. Credit assignment determines how feedback is allocated across the decisions and trajectories that produce an outcome. Temporal approaches include RUDDER's return decomposition (Arjona-Medina et al., 2019) and FACTOR's trajectory-to-action and action-to-token allocation (Ma et al., 2026). GiGPO (Feng et al., 2025) and GraphGPO (Cheng et al., 2026) exploit shared intermediate states, which are difficult to align in code agent tasks, where trajectories often span over 100 turns and produce divergent repository and execution states. We instead redistribute credit across successful implementations without intermediate-state alignment.

At the optimization level, reward and advantage shaping adjust which trajectories or tokens receive stronger reinforcement. PAPO adds rubric-based advantages normalized within the passing subset, yielding a zero-sum additive correction (Tan et al., 2026). EDAS reshapes failed-trajectory advantages using error diversity (Liu et al., 2026), while GTPO/GRPO-S (Tan & Pan, 2025) and RL-ZVP (Le et al., 2025) use entropy-guided shaping, the latter on zero-variance groups. We instead reweight successful-trajectory advantages in mixed-outcome groups using groupwise implementation-quality ranks, preserving total positive credit and quality-induced weight ratios.

3 Approach

GAGAR augments reinforcement learning for code agents with groupwise quality supervision beyond binary task outcomes. Figure 1 shows how groupwise agentic grading is integrated into the RL training loop. For each task, the grader jointly examines successful and failed attempts using their full trajectories, submitted patches, repository context, and test results. It can inspect code and run targeted checks before ranking the valid passing implementations. These rankings determine relative weights on positive advantages; sum-preserving redistribution then shifts credit toward higher-quality solutions while preserving the total positive advantage and leaving failed-trajectory advantages unchanged. The resulting sequence-level advantages supervise the model-generated response tokens during the policy update. We present the core method in this section; Appendix A.2 explains how to integrate it with reward adjustments such as length penalties.

3.1 Training Setup

Let x denote a coding task specified by its requirements, initial repository state, and execution environment. For each task, the rollout policy generates multiple trajectories, each comprising the agent's responses, tool calls, and environment observations. The final patch produced by each trajectory is evaluated using executable tests, yielding a binary outcome reward.

After excluding trajectories that cannot be evaluated because of infrastructure failures, let $\mathcal{G}_x = \{\tau_i\}_{i=1}^{n}$ denote the group of n valid trajectories. Confirmed hacks receive zero reward and are treated as failures. We denote the resulting effective reward by $R_i \in \{0, 1\}$ and compute all group statistics over these n trajectories. Following Dr. GRPO (Liu et al., 2025), we use the mean-centered outcome advantage $A_i = R_i - \bar{R}$, where $\bar{R} = n^{-1} \sum_{j=1}^{n} R_j$, without normalizing by the group reward standard deviation.

We adopt dynamic sampling (Yu et al., 2025) to retain groups containing both successful and failed trajectories, so that $0 < \bar{R} < 1$. Let $\mathcal{P} = \{i : R_i = 1\}$ and $\mathcal{F} = \{i : R_i = 0\}$ denote the passing and failing subsets, respectively. All passing trajectories receive the same positive advantage $1 - \bar{R}$, leaving differences in implementation quality and problem-solving behavior undifferentiated. Our objective is to allocate credit according to these quality differences while preserving the total positive advantage assigned to the group.

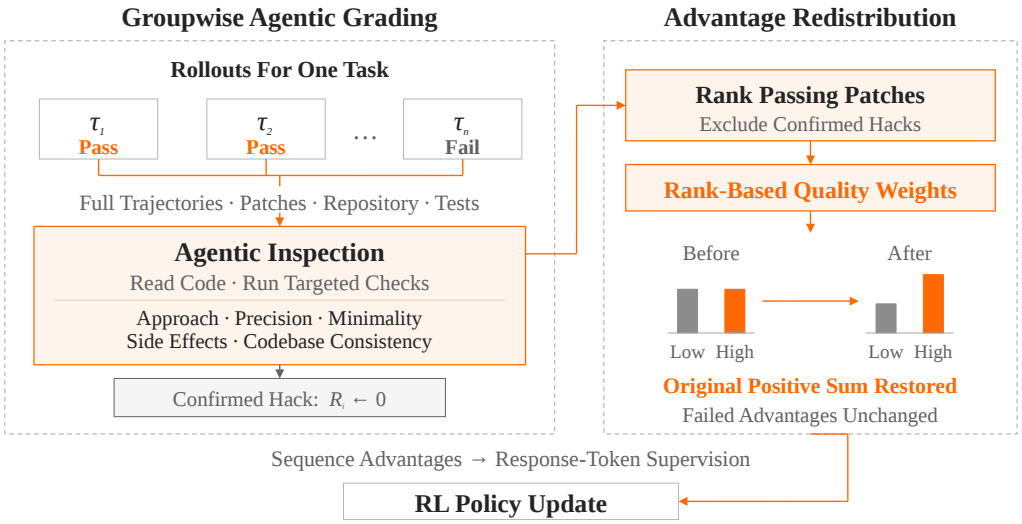

Figure 1: Overview of GAGAR. Groupwise agentic grading ranks passing implementations and guides sum-preserving advantage redistribution.

3.2 GROUPWISE AGENTIC GRADING

Groupwise Comparison. For each task, the grader jointly examines all rollouts in a shared workspace containing the task specification, repository, complete trajectories, submitted patches, and test outputs. Comparing different implementations of the same task helps the grader identify the strongest solutions and distinguish necessary changes from unnecessary complexity. This groupwise comparison also exposes ineffective problem-solving strategies that are difficult to recognize when evaluating candidates individually. Failed attempts provide additional context by revealing unsuccessful strategies, missed requirements, and failure modes, but only passing candidates receive quality rankings.

Both Flash and Pro experiments use the same pre-RL SFT checkpoint of MiMo-V2.6-Pro as the online grader (Section 4.1). Our initial implementation used Claude Opus 5, with an average end-to-end grading time of approximately 2,000 s per group. Since grading begins only after all rollouts in a group have completed, this added substantial latency to training. Our SFT-trained grader reduces average grading time to approximately 600 s while maintaining good accuracy. We further combine grading with partial-rollout scheduling so that grading completed groups can overlap with rollout generation for other tasks.

Agentic Evidence Gathering. Assessing a rollout group requires evidence scattered across long trajectories, patches, and repository files, which is difficult to review in a single model input. Our agentic grader instead gathers evidence iteratively. The grader first reviews a turn-by-turn summary of the rollouts to identify which parts require closer inspection. It then reads relevant portions of individual trajectories and cross-checks submitted patches against repository code and test logs. This allows it to trace how solutions were developed and investigate whether their changes are justified, rather than relying only on a fixed summary. It can also run targeted checks when inspection alone leaves a question unresolved. Negative assessments must cite supporting patch locations, trajectory events, or execution results.

Quality Criteria And Ranking. We assess five complementary aspects of code quality beyond test success. Approach suitability evaluates the solution strategy; precision and minimality assess whether its implementation is well-targeted and limited to necessary changes; side effects and codebase consistency assess its impact on existing behavior and maintainability. Together, these criteria guide learning toward sound strategies and well-targeted, merge-ready implementations, with the aim of reducing developer review and revision effort.

The grader first checks whether passing solutions rely on leaked or external answers. Confirmed cases receive zero reward and are excluded from quality ranking. For the remaining candidates, it scores these criteria and flags severe process issues. These assessments determine three quality tiers: $\mathcal{T}_1$ for strong implementations without severe process issues or unresolved regressions, $\mathcal{T}_2$ for intermediate candidates, and $\mathcal{T}_3$ for implementations with major quality defects. Within each tier, weighted criterion scores provide an initial ranking, which the grader can refine through evidence-backed comparisons; candidates remain tied when no distinction is justified. Each candidate's tier and within-tier rank are then mapped to a discount factor $f_i \in (0, 1]$ on its positive advantage: $\mathcal{T}_1$ candidates retain full or near-full weight, $\mathcal{T}_2$ candidates receive rank-dependent discounts, and $\mathcal{T}_3$ candidates receive the strongest fixed discount. Tied candidates receive identical factors. The grading stage thus produces a quality-based factor for each remaining passing trajectory, together with supporting evidence, as input to the advantage redistribution in Section 3.3. Detailed quality criteria, tiering, ranking, and factor-mapping rules are provided in Appendix A.1.

3.3 Sum-Preserving Advantage Redistribution

We use the quality factors f_i to redistribute credit among passing trajectories.

From Downweighting To Redistribution. Applying the factors alone would give $\widetilde{A}_i = f_i A_i$ for $i \in \mathcal{P}$ and leave failed-trajectory advantages unchanged. Since the original advantages sum to zero, this removes positive credit without a corresponding change on the negative side:

$$D = \sum_{i \in \mathcal{P}} (1 - f_i) A_i \geq 0, \qquad \sum_{i=1}^{n} \widetilde{A}_i = -D. \tag{1}$$

This deficit makes the total magnitude of negative advantages exceed the total positive advantage, weakening reinforcement of successful trajectories relative to the penalties on failed ones. In our experiments, quality-based downweighting without redistribution accompanies rapid growth in policy entropy and trajectory length, together with unstable evaluation performance (Section 4.4).

To favor higher-quality solutions while preserving the total positive advantage assigned to passing trajectories, we redistribute the removed credit among them. Let $S_+ = \sum_{i \in \mathcal{P}} A_i$ be this sum. We apply a common rescaling factor after quality-based downweighting:

$$\lambda = \frac{S_+}{\sum_{j \in \mathcal{P}} f_j A_j}, \qquad A_i^\star = \begin{cases} \lambda f_i A_i, & i \in \mathcal{P}, \\ A_i, & i \in \mathcal{F}. \end{cases} \tag{2}$$

The denominator is positive for every retained group. Rescaling applies to all passing trajectories; the removed credit is reallocated in proportion to their quality-weighted advantages.

Credit Conservation And Relative Preference. Equation 2 directly gives

$$\sum_{i \in \mathcal{P}} A_i^\star = S_+, \qquad A_i^\star = A_i \;\; (i \in \mathcal{F}), \qquad \sum_{i=1}^{n} (A_i^\star - A_i) = 0. \tag{3}$$

Thus, the change is zero-sum over the passing subset, and the group retains zero mean without modifying failed-trajectory advantages. Because all passes initially have the same advantage, the update has the equivalent form

$$A_i^\star = (1 - \bar{R}) \frac{f_i}{\bar{f}_{\mathcal{P}}}, \qquad \bar{f}_{\mathcal{P}} = \frac{1}{|\mathcal{P}|} \sum_{j \in \mathcal{P}} f_j, \quad i \in \mathcal{P}. \tag{4}$$

A candidate gains credit when its factor exceeds the passing-set average and gives up credit when its factor falls below that average. Because the same rescaling factor is applied to every passing trajectory, the advantage ratios established by downweighting remain unchanged: $A_i^\star / A_j^\star = \widetilde{A}_i / \widetilde{A}_j = f_i / f_j$ for $i, j \in \mathcal{P}$. Thus, restoring the total positive advantage preserves both the ordering and the relative strength of the quality preferences. If the group contains only one passing candidate, or all passing factors are equal, this formulation reduces to the original outcome advantages. Our method redistributes trajectory-level advantages without requiring matching intermediate states across rollouts.

We use the advantage-space formulation because it directly expresses how quality weighting should change trajectory learning weights. Mean centering after reward shaping already guarantees zero-sum advantages, but does not by itself preserve the original total positive advantage, failed-trajectory advantages, or quality-induced weight ratios. Our rescaling enforces these additional constraints by redistributing removed positive credit among passing trajectories rather than discarding it. Under mean-centered estimation, we derive an exactly equivalent reward transformation from this target advantage allocation, rather than introducing a separate reward-shaping rule (Appendix A.3). The same conservation principle could be applied at the token level by redistributing positive credit among tokens without changing its total.

3.4 Online Training Integration

Grading runs asynchronously with rollout collection. Before using a grading result, we check its validity, for example by ensuring that no passing trajectory is missing from the ranking. Unusable results fall back to the original outcome advantages. Confirmed reliance on an external or leaked solution resets the affected reward to zero before group statistics are recomputed; this integrity correction is separate from quality-based redistribution. The resulting sequence advantage is broadcast to the model-generated response tokens throughout each trajectory. Appendix A.2 specifies the implementation safeguards and distinguishes them from the exact sum-preserving formulation above.

4 Experiments

We organize our experiments around two questions: (1) Can GAGAR improve code agent performance, implementation quality, and problem-solving behavior? (2) What role does sum-preserving redistribution play in training stability? We initialize our industrial-scale experiments from pre-RL SFT checkpoints of MiMo-V2.6-Flash and MiMo-V2.6-Pro. Our primary controlled study uses code-only RL with Flash, complemented by large-scale mixed-task RL with both Flash and Pro.

4.1 Experimental Setup

Models. We initialize RL from pre-RL SFT checkpoints of two industrial-scale mixture-of-experts models: MiMo-V2.6-Flash, with 310B total and 15B active parameters, and MiMo-V2.6-Pro, with 1.02T total and 42B active parameters. We use Flash and Pro as shorthand for the corresponding model families throughout the experiments. Both model families use the same pre-RL SFT checkpoint of MiMo-V2.6-Pro as the online grader.

Training Configuration. Our main experiments use code-only RL with Flash, a training batch size of 128, 16 rollouts per prompt, and token-mean loss aggregation. We use the mean-centered advantage estimator and mixed-outcome group filtering described in Section 3.1, with grading performed asynchronously during rollout collection. Separately, we integrate GAGAR into industrial-scale mixed-task RL runs of both Flash and Pro. Each run uses 1,568 prompts per update, 16 rollouts per prompt. These runs combine coding with other domains, as detailed in Section 4.5.

Benchmarks And Evaluation Protocol. Our evaluation covers two public benchmarks. DeepSWE v1.1 (Huang et al., 2026) evaluates long-horizon software development. SWE-bench Pro (Deng et al., 2025) measures challenging repository-level issue resolution. On both benchmarks, we evaluate the baseline and GAGAR under identical experimental settings and compare shared training steps, reporting pass rates, mean interaction turns, and mean total token length. Each evaluation uses three samples per task, with mean pass rate reported as `avg@3`.

Baselines. The main Flash comparison contrasts a binary-outcome baseline without online quality grading against GAGAR, which adds groupwise agentic grading and sum-preserving advantage redistribution. To examine credit balance, we also compare against quality-based downweighting without redistribution or subsequent group centering in Section 4.4. We include Kimi K3, GPT-5.6 Sol, and Claude Opus 5 as external baseline models for the industrial-scale mixed-task RL experiments in Section 4.5.

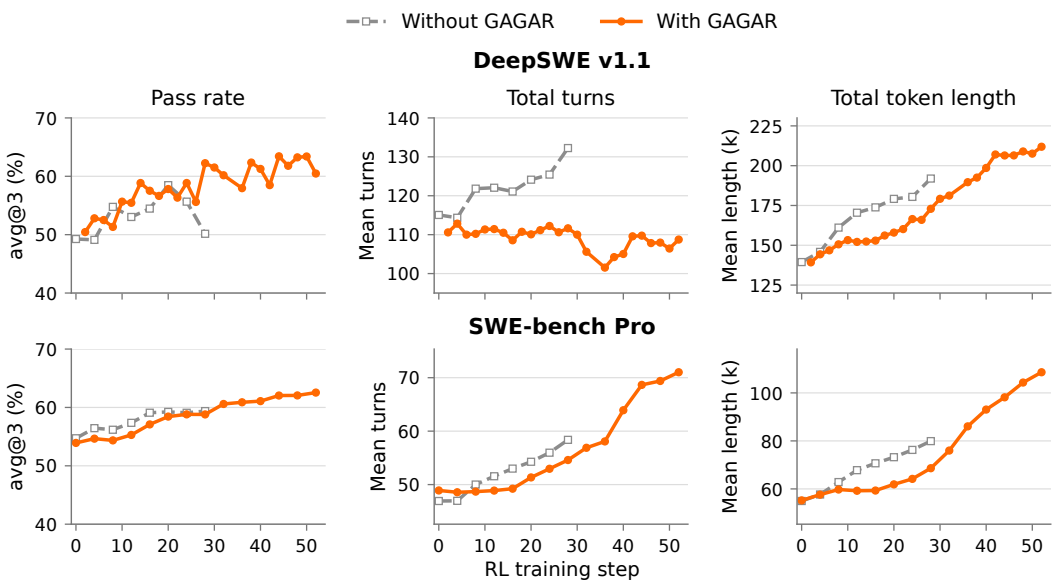

Figure 2: Training dynamics of Flash with and without GAGAR. Columns show pass rate `avg@3`, mean main-agent turns, and mean main-agent token length.

4.2 MAIN EXPERIMENTS WITH FLASH

Figure 2 compares task performance and interaction efficiency throughout code-only RL with and without GAGAR.

Task Performance. We stop the binary-reward baseline at step 28 in response to rapid performance degradation: its DeepSWE pass rate falls from 58.5% at step 20 to 50.2%. At step 28, GAGAR achieves 62.2% on DeepSWE v1.1, exceeding the baseline by 12.1 percentage points. With continued training, GAGAR reaches a peak DeepSWE pass rate of 63.4% at step 44. On SWE-bench Pro, the baseline plateaus at approximately 59% from step 16 through step 28, whereas GAGAR continues to improve with further training, reaching 62.5% at step 52.

Interaction Efficiency And Training Dynamics. Without GAGAR, DeepSWE trajectory lengths grow rapidly and more trajectories are truncated at the length limit, accompanying the sharp decline in pass rate. With GAGAR, DeepSWE turn counts remain roughly stable through step 52 and token length grows more gradually, supporting continued training without the baseline's abrupt deterioration. GAGAR uses fewer turns and tokens at every shared DeepSWE checkpoint, with similar reductions on SWE-bench Pro. At step 28, it reduces DeepSWE's mean turn count from 132.3 to 111.6 and mean token length from 191.9k to 172.9k, reductions of 15.6% and 9.9%, respectively. On SWE-bench Pro, mean turns decrease from 58.4 to 54.6 and mean token length from 79.9k to 68.6k, reductions of 6.5% and 14.1%. The larger pass-rate gains and turn savings on DeepSWE highlight the benefit on difficult, long-horizon tasks requiring many interactions. SWE-bench Pro trajectories are shorter and retain more headroom for growth during continued training.

4.3 IMPLEMENTATION QUALITY AND PROBLEM-SOLVING BEHAVIOR

We evaluate solution quality beyond test success on a fixed random sample of 30 DeepSWE tasks at shared evaluation checkpoints through the end of baseline training. Each task-level group pools up to three rollouts per method, including failures, with anonymized identifiers and randomized order. Claude Opus 5, distinct from the online grader, jointly reviews recorded trajectories, submitted patches, and test outcomes using the five criteria in Section 3.2, scoring and ranking all candidates.

We compute the rubric-weighted quality score defined in Appendix A.1. We then derive average win rates from the joint groupwise rankings. Finally, we measure how often each method ranks first in its group, splitting cross-method ties equally.

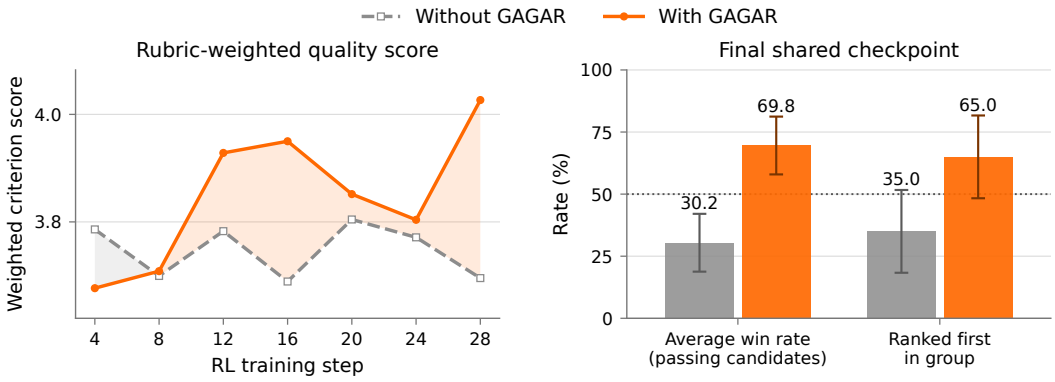

Figure 3: Implementation quality on DeepSWE v1.1. Left: mean rubric-weighted quality score. Right: average win rate among passing candidates and group-first share at the final shared checkpoint.

Figure 3 shows that, at the final shared checkpoint, GAGAR achieves a mean quality score of 4.03 versus 3.70 for the baseline, with an average win rate of 69.8% among passing candidates. It ranks first in 65.0% of groups after splitting ties. Across the audited checkpoints, improvements are most apparent in implementation precision, minimality of changes, and avoidance of unintended side effects; the fraction of candidates assigned to the highest-quality tier, $\mathcal{T}_1$, increases from 25.4% to 34.3%. These assessments support our goal of producing precise, task-scoped, merge-ready implementations that reduce developer review and revision effort.

4.4 Ablation Of Sum-Preserving Redistribution

We compare code-only Flash training with the full redistribution method against a downweighting-only run, which applies quality factors to passing trajectories without restoring the removed positive credit or re-centering the resulting advantages. Figure 4 shows training and evaluation dynamics over the first 30 steps.

Downweighting alone removes positive credit while leaving negative advantages unchanged, shifting their balance toward negative-advantage updates, as described by equation 1. The downweighting-only run has a consistently higher policy-gradient loss, averaging 0.0304 over steps 1–30, compared with 0.0021 for the full method. This is consistent with the signed nature of the objective: near unit importance ratios, the policy-gradient loss is approximately the negative weighted mean advantage, so an excess of negative advantage can raise its value.

Without redistribution, policy entropy rises from 0.359 at step 1 to 0.905 at step 30, while mean training-rollout length grows from 47.1k to 114.1k tokens. With redistribution, entropy increases more gradually, from 0.358 to 0.513, and mean length grows from 46.5k to 69.9k tokens. The sharper increases without redistribution accompany unstable downstream performance rather than sustained evaluation gains.

On DeepSWE, the downweighting-only pass rate drops from 56.5% at step 18 to 48.8% at step 20, then partially recovers to 56.2% at step 28, compared with 62.2% for the full method. Its mean interaction count and token length peak at 158.3 turns and 246.7k tokens at step 22. At step 28, these remain elevated at 143.5 turns and 236.0k tokens, versus 111.6 turns and 172.9k tokens with redistribution. Together, these observations support restoring positive credit rather than discarding it when introducing quality preferences.

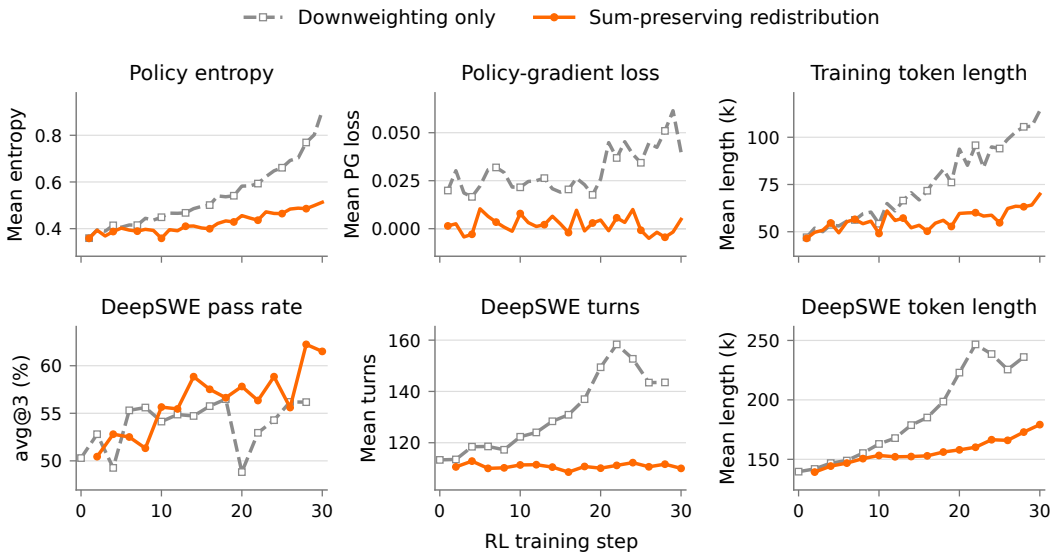

Figure 4: Credit balance and training dynamics in Flash RL.

4.5 Industrial-Scale Mixed-Task RL

To assess applicability in industrial-scale training, we integrate GAGAR into mixed-task RL across multiple domains with both Flash and Pro, spanning 310B and 1.02T total parameters, respectively. Each update uses 1,568 prompts with 16 rollouts per prompt. Grading and redistribution operate on eligible coding-task groups within this heterogeneous workload.

Table 1 compares the industrial-scale Flash and Pro checkpoints with K3, GPT-5.6 Sol, and Claude Opus 5. On DeepSWE v1.1, Pro outperforms Kimi K3 despite having substantially fewer total parameters: 1.02T versus 2.8T (Team, 2026). It also surpasses GPT-5.6 Sol on SWE-bench Pro, scoring 62.7% versus 60.5%. On DeepSWE v1.1, its performance approaches GPT-5.6 Sol and Claude Opus 5, which score 73.0% and 74.0%, respectively. These comparisons demonstrate competitive code agent performance relative to frontier models.

Table 1: Coding performance (%) of industrial-scale mixed-RL models and external baselines.

Model	**DeepSWE**	**SWE-bench Pro**
K3	69.0	63.3
GPT-5.6 Sol	73.0	60.5
Claude Opus 5	74.0	79.9
Flash (mixed RL)	67.9	60.9
Pro (mixed RL)	71.9	62.7

5 Conclusion

We presented GAGAR, which combines groupwise agentic grading with sum-preserving advantage redistribution to reinforce higher-quality test-passing coding trajectories. Experiments with the 310B-parameter Flash model show improved coding performance with fewer interaction turns and lower token usage, while ablations highlight the importance of credit balance for training stability. Industrial-scale mixed-task RL with Flash and the 1.02T-parameter Pro model demonstrates the applicability of the approach at both model scales.

AI USE STATEMENT

In this work, we used generative AI tools to edit the manuscript to improve its clarity and readability. For figures, the authors supplied the data and specified the designs, while AI assistance was limited to improving their visual presentation. Additionally, we used these tools to suggest a title and keywords for the paper. We have manually reviewed and verified all AI-assisted content. We take responsibility for the final content of this work, including all AI-assisted content.

REPRODUCIBILITY STATEMENT

We provide an open-source implementation of our method in the supplementary materials. Section 3 describes the grading and advantage-redistribution procedure, Appendix A specifies the ranking rules and implementation safeguards, and Section 4.1 describes the experimental setting. The pre-RL SFT checkpoints used for policy initialization and online grading have not been publicly released; the industrial training data are also not fully public. Exact reproduction of the industrial-scale results therefore requires access to these resources and the corresponding training infrastructure.

REFERENCES

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: return decomposition for delayed rewards. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 13544–13555, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/16105fb9cc614fc29e1bda00dab60d41-Abstract.html.

Xin Cheng, Shuo He, Lang Feng, Haiyang Xu, Ming Yan, Lei Feng, and Bo An. Beyond trajectory-level attribution: Graph-based credit assignment for agentic reinforcement learning. *CoRR*, abs/2605.26684, 2026. doi: 10.48550/ARXIV.2605.26684. URL https://doi.org/10.48550/arXiv.2605.26684.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can AI agents solve long-horizon software engineering tasks? *CoRR*, abs/2509.16941, 2025. doi: 10.48550/ARXIV.2509.16941. URL https://doi.org/10.48550/arXiv.2509.16941.

Lishui Fan, Yu Zhang, Mouxiang Chen, and Zhongxin Liu. Recode: Reinforcing code generation with reasoning-process rewards. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 43896–43914. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.2031. URL https://doi.org/10.18653/v1/2026.acl-long.2031.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Yunlong Feng, Yang Xu, Xiao Xu, Binyuan Hui, and Junyang Lin. Towards better correctness and efficiency in code generation. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial*

Intelligence, AAAI 2026, Singapore, January 20-27, 2026, pp. 30708–30716. AAAI Press, 2026. doi: 10.1609/AAAI.V40I36.40327. URL https://doi.org/10.1609/aaai.v40i36.40327.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. Deepswe: Measuring frontier coding agents on original, long-horizon engineering tasks. *CoRR*, abs/2607.07946, 2026. doi: 10.48550/ARXIV.2607.07946. URL https://doi.org/10.48550/arXiv.2607.07946.

Mengzhao Jia, Zhihan Zhang, and Meng Jiang. Prioritizing the best: Incentivizing reliable multimodal reasoning by rewarding beyond answer correctness. *CoRR*, abs/2604.18892, 2026. doi: 10.48550/ARXIV.2604.18892. URL https://doi.org/10.48550/arXiv.2604.18892.

Thanh-Long V. Le, Myeongho Jeon, Kim Vu, Viet Dac Lai, and Eunho Yang. No prompt left behind: Exploiting zero-variance prompts in LLM reinforcement learning via entropy-guided advantage shaping. *CoRR*, abs/2509.21880, 2025. doi: 10.48550/ARXIV.2509.21880. URL https://doi.org/10.48550/arXiv.2509.21880.

Wenpu Liu, Yuqi Xu, Weichu Xie, Yongfu Zhu, Shuai Dong, Ziyue Wang, Wenqi Shao, Xiaoying Zhang, Tong Yang, Nan Duan, and Jiaqi Wang. Leveraging error diversity in group rollouts for reinforcement learning. *CoRR*, abs/2605.17333, 2026. doi: 10.48550/ARXIV.2605.17333. URL https://doi.org/10.48550/arXiv.2605.17333.

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. *CoRR*, abs/2503.20783, 2025. doi: 10.48550/ARXIV.2503.20783. URL https://doi.org/10.48550/arXiv.2503.20783.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent lightning: Train ANY AI agents with reinforcement learning. *CoRR*, abs/2508.03680, 2025. doi: 10.48550/ARXIV.2508.03680. URL https://doi.org/10.48550/arXiv.2508.03680.

Lichao Ma, Yang Sun, Shuaitao Zhao, Yangyi Fang, Cong Qin, Xiaoliang Fu, Yuhang Tian, Yuchen Wei, Junbo Zhu, Yang Wei, Lu Pan, and Jiaye Lin. How much, then where: Credit-conserving action-to-token allocation for multi-turn agent reinforcement learning. *CoRR*, abs/2608.07118, 2026. doi: 10.48550/ARXIV.2608.07118. URL https://doi.org/10.48550/arXiv.2608.07118.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo García, and Mingyi Hong. Hiper: Hierarchical reinforcement learning with explicit credit assignment for large language model agents. *CoRR*, abs/2602.16165, 2026. doi: 10.48550/ARXIV.2602.16165. URL https://doi.org/10.48550/arXiv.2602.16165.

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

Hongze Tan and Jianfei Pan. GTPO and GRPO-S: token and sequence-level reward shaping with policy entropy. *CoRR*, abs/2508.04349, 2025. doi: 10.48550/ARXIV.2508.04349. URL https://doi.org/10.48550/arXiv.2508.04349.

Zelin Tan, Zhouliang Yu, Bohan Lin, Zijie Geng, Hejia Geng, Yudong Zhang, Mulei Zhang, Yang Chen, Shuyue Hu, Zhenfei Yin, Chen Zhang, and Lei Bai. Stabilizing rubric integration training via decoupled advantage normalization. *CoRR*, abs/2603.26535, 2026. doi: 10.48550/ARXIV.2603.26535. URL https://doi.org/10.48550/arXiv.2603.26535.

Kimi Team. Kimi K3: open frontier intelligence. *CoRR*, abs/2607.24653, 2026. doi: 10.48550/ARXIV.2607.24653. URL https://doi.org/10.48550/arXiv.2607.24653.

Yuanda Xu, Zhengze Zhou, Hejian Sang, Xiaomin Li, Jiaxin Zhang, Xinchen Du, Zhipeng Wang, and Alborz Geramifard. TRIAGE: role-typed credit assignment for agentic reinforcement learning. *CoRR*, abs/2606.32017, 2026. doi: 10.48550/ARXIV.2606.32017. URL https://doi.org/10.48550/arXiv.2606.32017.

Sen Yang, Shanbo Cheng, Lu Xu, Jianbing Zhang, and Shujian Huang. GRRM: group relative reward modeling for machine translation. *CoRR*, abs/2602.14028, 2026. doi: 10.48550/ARXIV.2602.14028. URL https://doi.org/10.48550/arXiv.2602.14028.

Jing Ye, Rui Wang, Yuchuan Wu, Victor Ma, Feiteng Fang, Fei Huang, and Yongbin Li. CPO: addressing reward ambiguity in role-playing dialogue via comparative policy optimization. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 297–323. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-EMNLP.18. URL https://doi.org/10.18653/v1/2025.findings-emnlp.18.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/zhuge25a.html.

A Method Implementation Details

A.1 Quality Criteria And Rank-To-Weight Mapping

Table 2: Five dimensions of groupwise implementation-quality assessment, with representative higher- and lower-quality signals.

Dimension	Higher-Quality Signals	Lower-Quality Signals
Approach Suitability	Root-cause resolution Appropriate solution strategy	Symptom-level workarounds Fundamentally flawed strategy
Implementation Precision	Correct change locations Complete behavior coverage	Scattered special cases Redundant fallback logic
Minimality Of Changes	Necessary implementation edits Task-scoped changes	Unrelated refactoring Redundant helpers or branches
Unintended Side Effects	Preserved surrounding behavior Stable public interfaces	Unrelated behavior changes New risks in core code paths
Codebase Consistency	Reuse of existing abstractions Consistent code conventions	Ad hoc mechanisms Inconsistent style or comments

The grader assigns integer scores from 1 to 5 to the five criteria in Table 2. Denote these scores by s_i^{app}, s_i^{prec}, s_i^{min}, s_i^{side}, and s_i^{style}, respectively. The initial ranking uses

$$W_i = 0.30s_i^{\text{app}} + 0.25s_i^{\text{prec}} + 0.20s_i^{\text{min}} + 0.15s_i^{\text{side}} + 0.10s_i^{\text{style}}. \tag{5}$$

These scores are produced after joint inspection of the group. The grader can revise the initial ordering, but must explain each revision with task-specific evidence. Rankings must respect quality tiers, and ties are allowed within each tier.

Quality Tiers. Tiers are recomputed from the criterion scores and flagged issues in a fixed order. We first assign candidates to $\mathcal{T}_3$ if they exhibit a confirmed unrequested rewrite or test-specific workaround, receive the lowest approach-suitability score, or score at most 2 on both minimality and side-effect avoidance. Among the remaining candidates, those scoring at least 4 on every criterion with no severe process issue or unresolved regression belong to $\mathcal{T}_1$; all others belong to $\mathcal{T}_2$. Confirmed reliance on leaked or external solutions is handled through reward correction before tier assignment.

Let k_i denote the zero-based rank of a candidate's tied group within its tier, and let K_2 be the number of tied groups in $\mathcal{T}_2$. The factor map is

$$f_i = \begin{cases} 1, & i \in \mathcal{T}_1,\ k_i = 0, \\ f_{\text{runner}}, & i \in \mathcal{T}_1,\ k_i > 0, \\ f_{\max} - (f_{\max} - f_{\min})\dfrac{k_i}{K_2 - 1}, & i \in \mathcal{T}_2,\ K_2 > 1, \\ f_{\max}, & i \in \mathcal{T}_2,\ K_2 = 1, \\ f_{\text{low}}, & i \in \mathcal{T}_3. \end{cases} \tag{6}$$

The Flash configuration uses $f_{\text{runner}} = 0.9$, $f_{\min} = 0.4$, $f_{\max} = 0.85$, and $f_{\text{low}} = 0.2$. All candidates in a tied group receive the same factor. Factors are determined by tied-group ranks rather than by the number of candidates preceding a trajectory.

A.2 Implementation Safeguards

The training implementation bounds the common rescaling factor in equation 2. Writing $\lambda_{\text{bnd}} = \min(\lambda, \lambda_{\max})$, it computes

$$B_i = \begin{cases} \lambda_{\text{bnd}} f_i A_i, & i \in \mathcal{P}, \\ A_i, & i \in \mathcal{F}, \end{cases} \qquad A_i^{\text{train}} = B_i - \frac{1}{n}\sum_{j=1}^{n} B_j. \tag{7}$$

The Flash configuration sets $\lambda_{\max} = 1.5$. When the bound is inactive, the centering term is zero and the update equals $A_i^\star$. When it is active, centering restores zero group mean and preserves the ordering of passing advantages, but the original positive-advantage sum and failed-trajectory advantages need not be preserved. The exact conservation identities in Section 3.3 apply to the sum-preserving formulation, not to this bounded branch.

Existing Reward Postprocessing. The training system also supports length-based reward adjustments, separate from grading. If such postprocessing produces nonbinary scalar rewards r_i, the implementation uses $a_i = r_i - \bar{r}$, where $\bar{r}$ is their valid-group mean. Passing candidates are weighted using $a_i^+ = \max(a_i, 0)$, and the common rescaling factor targets $\sum_{i\in\mathcal{P}} a_i^+$ rather than a binary-reward sum:

$$\lambda_{\text{bnd}} = \min\left(\frac{\sum_{j\in\mathcal{P}} a_j^+}{\sum_{j\in\mathcal{P}} f_j a_j^+}, \lambda_{\max}\right), \qquad B_i = \begin{cases} \lambda_{\text{bnd}} f_i a_i^+, & i \in \mathcal{P}, \\ a_i, & i \in \mathcal{F}. \end{cases} \tag{8}$$

Rescaling is skipped when the denominator is zero, and the final valid-group mean is subtracted as in equation 7. Unlike the binary case, passing candidates can have unequal initial advantages; their final ordering then depends on both those advantages and the quality factors. When $r_i = R_i$, this rule reduces to the binary formulation with the same bound.

Infrastructure-invalid trajectories are masked before group statistics are computed. Groups confirmed to have broken task evaluation are excluded from training. When a reference patch is available, the grader may use it to understand the task, but similarity to that patch is not a quality criterion. The reference is not a sampled trajectory and receives no training advantage. Invalid or unavailable grading results leave the original learning signal in place.

A.3 Equivalent Reward-Space Formulation

Under the mean-centered advantage estimator in Section 3.1, the sum-preserving update admits an exactly equivalent reward transformation. We keep the same valid rollout group and the passing and failing subsets determined by outcome verification.

Equivalent Rewards. We construct the equivalent rewards from the target advantages by setting $R'_i = A_i^\star + \bar{R}$, where $\bar{R}$ is the original pass rate. Substituting equation 4 for passes and $A_i^\star = -\bar{R}$ for failures gives

$$R'_i = \begin{cases} \bar{R} + (1-\bar{R})\dfrac{f_i}{\bar{f}_\mathcal{P}}, & i \in \mathcal{P}, \\ 0, & i \in \mathcal{F}. \end{cases} \tag{9}$$

Since $\sum_{i\in\mathcal{P}} f_i / \bar{f}_\mathcal{P} = |\mathcal{P}|$, the transformed rewards satisfy $\sum_{i=1}^n R'_i = |\mathcal{P}|$ and therefore $\overline{R'} = \bar{R}$. Subtracting this unchanged group mean gives

$$R'_i - \overline{R'} = \begin{cases} (1-\bar{R})\dfrac{f_i}{\bar{f}_\mathcal{P}}, & i \in \mathcal{P}, \\ -\bar{R}, & i \in \mathcal{F}, \end{cases} = A_i^\star. \tag{10}$$

For fixed rollouts and quality factors, this produces the same policy loss and gradient when all other loss terms, masks, and update settings remain unchanged. The transformed rewards depend on the group and may exceed 1; they are optimization signals, not replacements for the binary outcome labels used to identify passes and select groups. Additional reward clipping or reward-standard-deviation normalization generally breaks the equivalence.

Why Direct Reward Discounting Differs. Simply setting $\widehat{R}_i = f_i$ for passing trajectories and $\widehat{R}_i = 0$ for failures changes the group mean to $\bar{R}\bar{f}_\mathcal{P}$. The resulting advantages are

$$\widehat{A}_i = \begin{cases} f_i - \bar{R}\bar{f}_\mathcal{P}, & i \in \mathcal{P}, \\ -\bar{R}\bar{f}_\mathcal{P}, & i \in \mathcal{F}. \end{cases} \tag{11}$$

This changes failed-trajectory advantages, generally alters the advantage ratios among passes, and can assign negative advantages to passing trajectories when $f_i < \bar{R}\bar{f}_\mathcal{P}$. It is therefore not equivalent to our redistribution. Unlike advantage downweighting without subsequent centering, however,

direct reward discounting followed by mean subtraction still yields zero-sum advantages. The instability observed in our downweighting-only ablation should not be attributed to reward shaping in general.

Bounded And Postprocessed Updates. The implementation in Appendix A.2 also admits a reward-space realization. Let B_i be the intermediate value in equation 7 or equation 8, and let $\bar{r}$ be the original group mean reward. Setting $r'_i = B_i + \bar{r}$ and then centering gives $r'_i - \overline{r'} = B_i - \bar{B} = A_i^{\text{train}}$, where $\bar{B} = n^{-1} \sum_i B_i$. This reproduces the bounded implementation, including its deviations from exact conservation when the rescaling cap is active.