

GAPO: Robust Advantage Estimation for Real-World Code LLMs

Jianqing Zhang^{1,2}, Zhezheng Hao³, Wei Xia², Hande Dong², Hong Wang²,
Chenxing Wei⁴, Yuyan Zhou², Yubin Qi⁵, Qiang Lin², Jian Cao¹,

¹Shanghai Jiao Tong University, ²Tencent,

³Zhejiang University, ⁴Shenzhen University, ⁵Peking University

Correspondence: tsingz@sjtu.edu.cn, xwellxia@tencent.com, cao-jian@sjtu.edu.cn

Abstract

Reinforcement learning (RL) is widely used for post-training large language models (LLMs) in code editing, where group-relative methods, such as GRPO, are popular due to their critic-free and normalized advantage estimation. However, in real-world code-editing scenarios, reward distributions are often skewed with unpredictable noise, leading to distorted advantage computation and increased rollout outliers. To address this issue, we propose **Group Adaptive Policy Optimization (GAPO)**, which adaptively finds an *interval with the highest SNR (signal-to-noise Ratio)* per prompt and uses the median of that interval as an adaptive Q to replace the group mean in advantage calculation to reduce noise further. This adaptive Q robustly handles rollout noise while remaining plug-and-play and efficient. We evaluate **GAPO** on nine instruction-tuned LLMs (3B–14B) using a collected large dataset of 51,844 real-world, history-aware code-editing tasks spanning 10 programming languages. **GAPO** yields up to 4.35 in-domain (ID) and 5.30 out-of-domain (OOD) exact-match improvements over GRPO and its variant DAPO, while achieving lower clipping ratios and higher GPU throughput. Code: <https://github.com/TsingZ0/ver1-GAPO>.

1 Introduction

With the rapid advancement of large language models (LLMs), artificial intelligence (AI)-assisted coding has emerged as a prominent subfield and practical application (Chen et al., 2021; Li et al., 2022), demonstrating proven value in improving software engineering efficiency (Peng et al., 2023; Yetiştirten et al., 2023). Most code LLMs undergo a post-training stage, and reinforcement learning (RL) is a widely used method (Christiano et al., 2017; Wang et al., 2024; Hao et al., 2025).

Among RL methods, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and its variants are popular choices, known for their critic-free

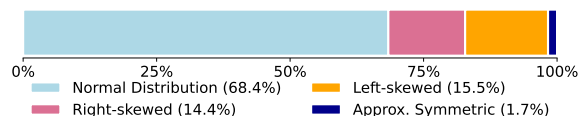


Figure 1: Reward distribution of rollouts before training using Qwen2.5-Coder-14B (Hui et al., 2024).

features (Liu et al., 2025b). The core characteristic of the GRPO family lies in its group-relative advantage computation, which samples a group of rollouts for each input prompt and calculates advantage values as normalized rewards relative to the mean reward within each group (Shao et al., 2024), as illustrated in Figure 2.

However, in real-world code-editing scenarios with complex contexts, inter-module invocation relationships, and diverse user intents, input prompts can inevitably induce noisy rollouts that contain outliers, which are unpredictable (Frauenknecht et al., 2025; Wu et al., 2022). In such practical cases, the expected normal or symmetric reward distributions often shift toward left- or right-skewed forms (Moore et al., 2009), a phenomenon frequently observed in practice (see Figure 1). Given a normally distributed reward within the range $[0, 1]$, when most rewards are greater than 0.5 (*i.e.*, the mean exceeds 0.5), the practical reward distribution becomes left-skewed, as unpredictable outliers appear anywhere within the range, but their impact is more pronounced in the lower-reward region (below 0.5). The reverse occurs for right-skewed cases. Such rollout noise, often model- and scenario-specific, hinders LLM generalization and is hard to handle consistently.

To reduce the impact of rollout noise and outliers, we propose **Group Adaptive Policy Optimization (GAPO)**, a robust group-adaptive advantage method that enhances GRPO and its variants (*e.g.*, DAPO (Yu et al., 2025)), as shown in Figure 2. Unlike the mean, which treats all rewards uniformly and

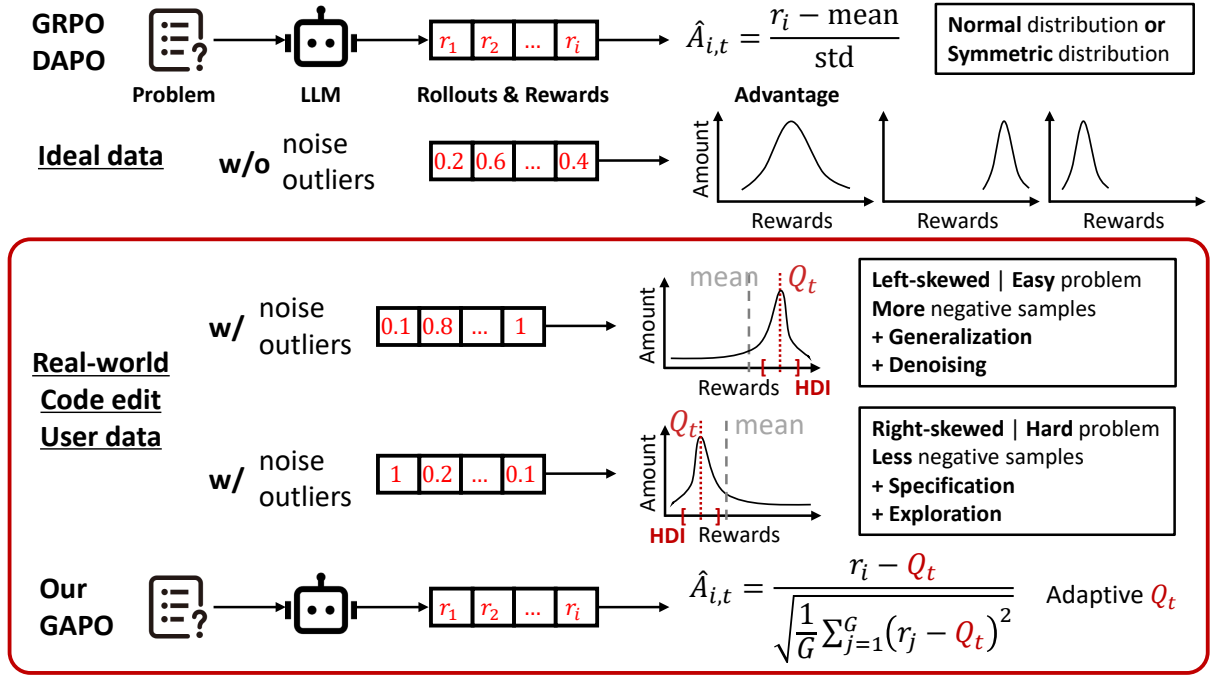


Figure 2: Illustration of GRPO/DAPO and our proposed **GAPO**, where r_i and $\hat{A}_{i,t}$ denote the i -th reward and the advantage within a group at step t , respectively. Q_t is defined as the median of an adaptive *Highest-Density Interval* (HDI) derived from the reward distribution of each prompt during the rollout process.

is sensitive to outliers, **GAPO** first identifies the sub-region with the highest signal-to-noise ratio (SNR) by reformulating the task as a classical *Highest-Density Interval* (HDI) problem (O’Neill, 2022), solved via an adapted sliding-window scan algorithm. We further enhance robustness by using the median of this region as the adaptive Q instead of the mean in group-relative advantage computation.

As shown in Figure 2, the adaptive Q provides additional benefits. For left-skewed distributions, **GAPO** generates more negative rollouts, improving generalization on easy problems (Mu et al., 2025; Zhu et al., 2025). For right-skewed distributions, it promotes specialized learning on hard cases. The LLM-specific rollout noise also carries useful information, reflecting corner-case behaviors and forming the model’s *blurry* ability edge. After updates, adaptive Q suppresses noise in easy cases and amplifies outlier advantages in hard cases, enhancing the LLM’s ability edge.

We evaluate nine diverse large language models (LLMs), both general-purpose and code-specialized, ranging from 3B to 14B parameters. Lacking public datasets for realistic, history-aware code edits, we collected 51,844 tasks across 10 languages—mainly Go (37.71%), Python (22.14%), and Java (21.03%), each with a prompt (context, history, edit range) and the ground-truth edited snippet

(see Table 3). Extensive experiments demonstrate the superiority of our **GAPO** over both GRPO and DAPO in both accuracy and GPU throughput. In summary, our contributions are:

- We are the first to observe that group-relative advantage is highly sensitive to outliers in real-world code editing, where reward distributions are often skewed by rollout noise.
- We introduce **GAPO**, which adaptively identifies the sub-region with the highest SNR for each input and scenario, enhancing the robustness of group-relative advantage calculation.
- We collect a large-scale real-world code-editing dataset and demonstrate **GAPO**’s ID and OOD superiority over GRPO and DAPO across nine LLMs (3B–14B), with lower clipping ratios and higher GPU throughput.

2 Preliminaries

2.1 Problem Formulation

In code editing, the model receives a prompt p with context, history, edit region, cursor, user instructions, and other details. The LLM (θ) generates a snippet $\hat{e}(p, \theta)$ to replace the edit region, or applies “no change” if indicated by special output tokens.

During training, we have access to the ground-truth edit e^* , the code modification the user intended for each prompt p . The objective is to maximize the expected reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{p \sim P(\mathcal{P})} [r(\hat{e}(p, \theta), e^*)], \quad (1)$$

where $r(\cdot, \cdot)$ is a reward function that quantifies the similarity or correctness of the predicted edit relative to the ground-truth edit.

2.2 Group Relative Policy Optimization

A key advantage of GRPO is that it eliminates the need for a separate value function approximation; instead, it calculates advantages using the average reward across multiple sampled responses (rollouts) generated from the same input prompt. When using GRPO from an RL view, we can rewrite Eq. (1) to be

$$\begin{aligned} \mathcal{J}_{GRPO}(\theta) &= \mathbb{E}_{p \sim P(\mathcal{P}), \{e_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|p)} \\ &\frac{1}{G} \sum_{i=1}^G \frac{1}{|e_i|} \sum_{i=1}^{|e_i|} [\min(\kappa_{i,t} \cdot \hat{A}_{i,t}, \rho_{i,t,\epsilon} \cdot \hat{A}_{i,t}) + C], \\ \text{s.t.}, \quad \kappa_{i,t} &= \frac{\pi_{\theta}(e_{i,t}|p, e_{i,<t})}{\pi_{\theta_{old}}(e_{i,t}|p, e_{i,<t})}, \\ \rho_{i,t,\epsilon} &= \text{clip}\left(\frac{\pi_{\theta}(e_{i,t}|p, e_{i,<t})}{\pi_{\theta_{old}}(e_{i,t}|p, e_{i,<t})}, 1 - \epsilon, 1 + \epsilon\right), \\ C &= -\beta \mathbb{D}_{KL}[\pi_{\theta} || \pi_{ref}], \end{aligned} \quad (2)$$

where π_{θ} , $\pi_{\theta_{old}}$, and π_{ref} are the updating policy model, old policy model, and reference policy model, respectively. For each prompt p , GRPO samples a group of G edits $\{e_1, e_2, \dots, e_G\}$ from the old policy $\pi_{\theta_{old}}$. Here, ϵ and β are preset hyperparameters that control the optimization behavior, and \mathbb{D}_{KL} denotes the KL divergence between the current training policy and a fixed reference policy (Shao et al., 2024).

The advantage $\hat{A}_{i,t}$ is computed based on the relative rewards of the edits within each group. Given any reward model, we score each edit to obtain a set of G rewards $\mathbf{r} = \{r_1, r_2, \dots, r_G\}$, which are then normalized to yield the advantage calculation:

$$\hat{A}_{i,t} = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}. \quad (3)$$

3 Method

3.1 Motivation

As shown in Figure 1, many real-world code-editing prompts produce skewed reward distribu-

tions with outliers, adding noise and negative impact to RL rollouts and training (Frauenknecht et al., 2025; Hollenstein et al., 2022) because Eq. (3) aggregates all rewards uniformly. Common methods use non-adaptive geometric-mean (Zhao et al., 2025) or quantile statistics to reduce outlier influence (John, 2015; Rousseeuw and Hubert, 2011), but as Figure 1 shows, reward distributions vary across prompts, so a single mean or quantile cannot adapt effectively. Moreover, *noise can be a form of LLM-specific useful information* that should not be discarded, as it reflects the model’s output and corner-case behaviors, forming its ‘blurry ability edge.’ To address these, we propose a group-adaptive advantage calculation that identifies the highest-density reward region, minimizing the impact of outliers, while also leveraging noise by amplifying the absolute advantage of outliers to make the LLM’s ability edge more pronounced.

3.2 Group Adaptive Policy Optimization

Our **GAPO** method does not alter the objective of existing group-relative RL approaches; instead, it only modifies the advantage computation in Eq. (3). This design makes it simple to implement and plug-and-play with various group-relative RL frameworks, such as ver1 (Sheng et al., 2025), requiring only a few lines of code. Specifically, we redefine the advantage as

$$\hat{A}_{i,t} = \frac{r_i - Q_t}{\sqrt{\frac{1}{G} \sum_{j=1}^G (r_j - Q_t)^2}}, \quad (4)$$

where the denominator represents a variant of the standard deviation, ensuring consistency with the replacement of the mean by the adaptive Q value.

The key challenge, then, is to obtain the adaptive Q value for each group corresponding to each input prompt while updating the policy models. Although the median can mitigate the effect of outliers, it is computed over the entire group and is less adaptive to varying user scenarios. Thus, we propose first to *identify the region free of outliers* and then *set Q as the median of this sub-region \mathcal{H}* .

To achieve this, we seek a sub-region with the highest SNR, which represents the majority of the rewards with fewer outliers. This naturally leads to a classical statistical problem of finding the HDI for a given probability mass, that is, the narrowest interval (O’Neill, 2022). Formally, for a group of reward values, this can be expressed as

Highest-Density Interval (HDI)

Given G real numbers (rewards) r_1, r_2, \dots, r_G and an integer k (here $k = \lceil G\tau \rceil$), find indices $i < j$ with $j - i + 1 \geq k$ that minimize the interval length

$$L(i, j) = r_j - r_i,$$

where $r_{(1)}, r_{(2)}, \dots, r_{(G)}$ are the sorted values.

Algorithm 1 Find the Highest-Density Interval

Require: \mathcal{G} : G reward values in $[0, 1]$, $\tau \in [0, 1]$.

Ensure: \mathcal{H} : the HDI covering at least τ of points.

```
1:  $\mathcal{G}' \leftarrow \text{sort}(\mathcal{G})$ 
2:  $\text{min\_length} \leftarrow \infty$ 
3:  $\text{best\_start} \leftarrow 0$ 
4:  $\text{best\_end} \leftarrow k - 1$ 
5: for  $\text{start} = 0$  to  $G - k$  do
6:    $\text{end} \leftarrow \text{start} + k - 1$ 
7:    $\text{current\_length} \leftarrow \mathcal{G}'[\text{end}] - \mathcal{G}'[\text{start}]$ 
8:   if  $\text{current\_length} < \text{min\_length}$  then
9:      $\text{min\_length} \leftarrow \text{current\_length}$ 
10:     $\text{best\_start} \leftarrow \text{start}$ 
11:     $\text{best\_end} \leftarrow \text{end}$ 
12:   end if
13: end for
14:  $\mathcal{H} \leftarrow \mathcal{G}'[\text{best\_start} : \text{best\_end} + 1]$ 
15: return  $\mathcal{H}$ 
```

Next, to solve this problem, we use the algorithm in Algorithm 1. The method first sorts the data to obtain order statistics, then applies a sliding window of fixed size k over the sorted array, computing $r_{(i+k-1)} - r_{(i)}$ for each window. The minimal such difference corresponds to the shortest interval containing k points.

Why this is optimal: any interval covering k points corresponds to a contiguous block in the sorted list; enlarging the interval beyond k points cannot make it strictly shorter, so it suffices to check only blocks of size exactly k . The computational complexity is dominated by the sorting step, which requires $\mathcal{O}(n \log n)$ time (n : number of roll-outs, usually small), while the sliding window scan costs only $\mathcal{O}(n)$. Therefore, the total complexity is $\mathcal{O}(n \log n)$ time with $\mathcal{O}(m)$ extra space.

4 Experiment

LLMs. For comprehensive evaluation, we consider *nine* large language models (LLMs) covering a diverse spectrum: **Mistral-v0.3** (Jiang

et al., 2023) (general-purpose, non-reasoning; 7B), **Qwen2.5** (Qwen Team, 2024) (general-purpose, non-reasoning; 3B, 7B), **Qwen3** (Yang et al., 2025) (general-purpose, reasoning; 4B, 8B), **Qwen2.5-Coder** (Hui et al., 2024) (code-specialized, non-reasoning; 3B, 7B, 14B), and **DeepSeek-Coder** (Guo et al., 2024a) (code-specialized, non-reasoning; 6.7B). All these models are instruction-tuned versions downloaded from Hugging Face (Hugging Face, 2025).

Training and Evaluation Data. Our goal is to work on code editing tasks in *real-world software engineering* with rich context and editing histories, but no open-sourced dataset is available. Therefore, we collect training data from internal company users and remove all data containing private and sensitive information. Specifically, we finally have a total of 51,844 code-editing data points across 10 programming languages (Go, Python, Java, C++, Kotlin, TypeScript, JavaScript, C, Rust, and Lua). As shown in Table 1, Go, Python, and Java are the majority, which are also among the most popular programming languages (TIOBE Software BV, 2025). Moreover, Table 1 shows a large variation in input prompt and output editing lengths across scenarios, *requiring adaptation*. For evaluation, we use the only available open-sourced zeta dataset (Zed Industries, 2025) (OOD), which contains hundreds of samples, together with our collected test set (ID) of 3,897 cases.

Table 1: Statistics of the training data for real-world code editing tasks. “Input” and “Output” denote the input prompt and output ground-truth *text length* ranges, respectively.

	Input	Output	Count	Percent
Go	1,925 – 24,883	36 – 717	19,549	37.71%
Python	1,984 – 24,651	36 – 788	11,477	22.14%
Java	2,021 – 23,181	38 – 833	10,905	21.03%
C++	2,008 – 22,463	39 – 736	3,245	6.26%
Kotlin	2,146 – 16,353	38 – 706	2,302	4.44%
TypeScript	2,033 – 21,603	41 – 557	1,946	3.75%
JavaScript	2,033 – 18,230	43 – 568	1,239	2.39%
C	2,134 – 14,106	44 – 628	896	1.73%
Rust	2,608 – 11,204	44 – 507	221	0.43%
Lua	2,991 – 10,348	44 – 611	64	0.12%
Total	1,925 – 24,883	36 – 833	51,844	100.00%

Input-Output Structure. Each data consists of two fields: <prompt> and <edit>. The <prompt> field contains the code context, a sequence of edit histories, the code edit range (with the cursor’s position), and user-provided hints. The

Table 2: Exact match accuracy on the test set for nine LLMs. The asterisk (*) denotes reasoning LLMs. Step Δ : **GAPO** requires fewer training steps than the baselines to reach their best accuracy, showing better training efficiency.

Name	Mistral-v0.3	Qwen2.5		Qwen3*		Qwen2.5-Coder			DeepSeek-Coder
	7B	3B	7B	4B	8B	3B	7B	14B	
GMPO	12.71	38.94	39.38	39.20	36.95	39.74	40.12	42.58	23.07
KRPO	13.02	38.73	39.14	39.53	37.35	39.80	40.31	42.58	23.07
QAE	16.52	32.99	39.47	38.25	39.76	38.34	41.89	—	40.58
GRPO	12.93	38.80	39.05	39.47	36.80	39.69	40.05	42.64	23.32
GAPO (G)	13.58	39.96	41.36	40.09	39.62	42.70	44.40	46.25	23.85
Improve.	+0.65	+1.16	+2.31	+0.62	+2.82	+3.01	+4.35	+3.61	+0.53
Step Δ	-87	-109	-100	-55	-42	-139	-169	-121	-58
DAPO	16.59	32.80	39.46	37.99	39.80	38.74	41.64	—	41.09
GAPO (D)	17.20	33.87	41.13	39.62	40.64	39.98	43.96	—	43.03
Improve.	+0.61	+1.07	+1.67	+1.37	+0.84	+1.24	+2.32	—	+1.94
Step Δ	-58	-80	-33	-19	-48	-93	-65	—	-45

<edit> field is the ground truth output. Details are summarized in Table 3.

Field	Components
<prompt>	<system prompt> <current code> <sequence of edit histories> <code edit range> & <cursor> <user prompt>
<edit>	<ground truth>

Table 3: Components of each data in the training and test datasets.

Baselines. (1) Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is a widely influential RL algorithm for LLMs (Kumar et al., 2025). (2) Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) (Yu et al., 2025) is a popular successor to GRPO. (3) GMPO (Zhao et al., 2025) replaces the arithmetic mean with a non-adaptive geometric mean. (4) KRPO (Wang et al., 2025b) employs a Kalman filter to estimate latent reward uncertainty. (5) QAE (Wu et al., 2025b) introduces a group-wise K -quantile reward baseline for entropy-safe reasoning based on DAPO. GMPO, KRPO, and QAE are designed for discrete-reward tasks and *lack adaptability to diverse real-world noisy rollout distributions*.

Other Settings. We use the popular ver1 framework (Sheng et al., 2025) for RL training, which is widely adopted in industry due to its scalability. We follow most of the default set-

tings for GRPO and DAPO in ver1 with adaptive modifications for our code edit tasks. Our **GAPO** method is straightforward to implement, requiring only a few lines of code by modifying the `compute_grpo_outcome_advantage` function in the GRPO and DAPO implementations (see details in our code), showing its high compatibility. We use the exact match (em) as the evaluation metric following (Deng et al., 2025). All results are reported as the average over five trials. See more details and results in the Appendix.

We use a continuous-reward function that combines the exact match (em) metric (Dibia et al., 2023) and a normalized edit distance (ed) metric:

$$r(\hat{e}, e^*) = \frac{1}{2} \left[em(\hat{e}, e^*) + \left(1 - \frac{ed(\hat{e}, e^*)}{\max\{l(\hat{e}), l(e^*)\}} \right) \right], \quad (5)$$

where \hat{e} and e^* represent LLM output and ground truth, respectively. $ed(\hat{e}, e^*)$ is computed using the classical dynamic programming algorithm (Lcvenshtcin, 1966), and $l(\cdot)$ returns the code length.

4.1 In-Domain Performance

We begin by presenting the improvements of our **GAPO** over both GRPO and DAPO on *nine* LLMs, covering general-purpose and code-specific models, as well as reasoning and non-reasoning types. The original DAPO implementation encounters out-of-memory (OOM) issues with 14B models in our long-context setting, leading to missing results. **GAPO (G)** and **GAPO (D)** denote the application of our adaptive Q to GRPO and DAPO, respectively.

As shown in Table 2, our **GAPO** yields greater

benefits for the Qwen series compared to Mistral and DeepSeek. Within the Qwen family, **GAPO** demonstrates the strongest effect on Qwen2.5-Coder, achieving up to a 4.35-point improvement in exact match accuracy over GRPO/DAPO. This indicates that **GAPO** is particularly effective for initially strong, code-specific LLMs. In contrast, its effectiveness is limited for weaker models with low baseline performance, such as Mistral-v0.3. Overall, the *adaptive advantage computation* at the core of **GAPO** generalizes seamlessly to both GRPO and DAPO. However, GMPO, KRPO, and QAE perform similarly to their baselines (GRPO or DAPO), showing unstable improvements on real-world tasks with diverse noisy reward distributions.

Beyond accuracy, **GAPO** achieves higher training efficiency (Step Δ in Table 2), requiring fewer steps than GRPO and DAPO to reach the same accuracy, with especially large gains over GRPO. This stems from improved generalization on easy problems and enhanced specialization on hard ones, aligned with the optimization objective. Using adaptive Q encourages exploration on hard cases by amplifying the absolute advantage of outliers (with infrequent high rewards), making the gains more pronounced for GRPO, which lacks inherent exploration (Yu et al., 2025).

4.2 Out-of-Domain Performance

Table 4: Exact match accuracy on the OOD zeta set for Qwen2.5-Coder and Qwen3.

Model Size	Qwen3*		Qwen2.5-Coder		
	4B	8B	3B	7B	14B
GMPO	7.58	14.39	17.42	21.21	21.97
KRPO	8.33	15.15	18.18	22.73	23.49
QAE	13.64	18.94	14.39	16.67	—
GRPO	8.33	13.64	17.42	21.97	22.73
GAPO (G)	10.61	16.67	20.45	24.24	25.76
Improve.	+2.27	+3.03	+3.03	+2.27	+3.03
Imp. (%)	+27.27	+22.22	+17.39	+10.34	+13.33
DAPO	12.88	19.70	13.64	16.67	—
GAPO (D)	16.67	22.73	18.94	20.45	—
Improve.	+3.79	+3.03	+5.30	+3.79	—
Imp. (%)	+29.41	+15.38	+38.89	+22.73	—

We further evaluate **GAPO** on the open-sourced zeta dataset (Zed Industries, 2025), which, to our knowledge, is the only code-editing benchmark currently available and is out-of-domain (OOD) relative to our collected training set. Because the zeta set is small, the accuracy exhibits a stepwise

pattern. As shown in Table 4, **GAPO** consistently outperforms the original GRPO/DAPO across model scales on this OOD benchmark, achieving gains of up to 5.30 absolute points and 38.88% relative improvement. These substantial OOD improvements stem from **GAPO**'s robust feature and enhanced generalization capability.

4.3 Performance Curves

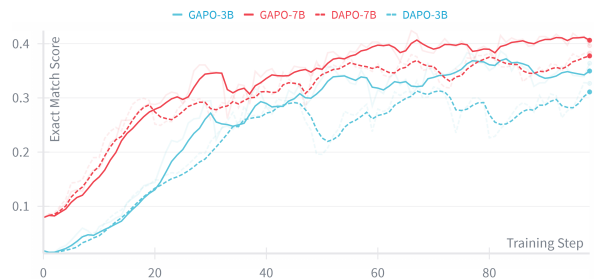


Figure 3: Performance curves on the test set for DAPO with Qwen2.5-Coder-3B/7B, as logged by W&B. The dotted curves correspond to LLMs post-trained with the original DAPO.

We further present the W&B-logged ID performance curves in Figure 3. Due to space limitations, only the results for DAPO are shown. The curves demonstrate that **GAPO** consistently improves DAPO, with clear performance gains that persist during training. Notably, **GAPO** complements DAPO's dynamic sampling, which filters zero-variance prompts and increases the relative proportion of outlier prompts.

4.4 Policy Training Curves

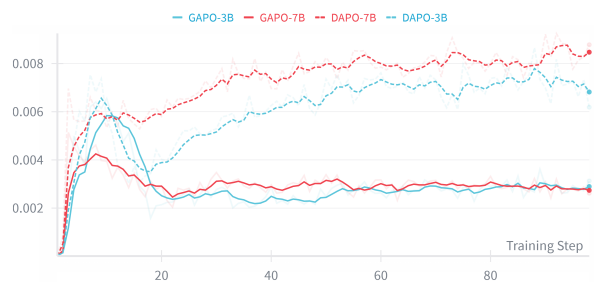


Figure 4: Clip fraction curves on the training set for DAPO with Qwen2.5-Coder-3B/7B, logged by W&B.

The clip fraction curves in Figure 4 illustrate the proportion of actions whose probability ratios were clipped during gradient updates. A low `pg_clipfrac` indicates that few updates reach the clipping threshold, implying gentler policy changes and better alignment with the model's current capabilities, allowing fuller utilization of the reward in-

formation. For both the peak and converged values, our **GAPO** consistently exhibits lower `pg_clipfrac` than DAPO, especially in later training steps.

4.5 GPU Throughput Curves

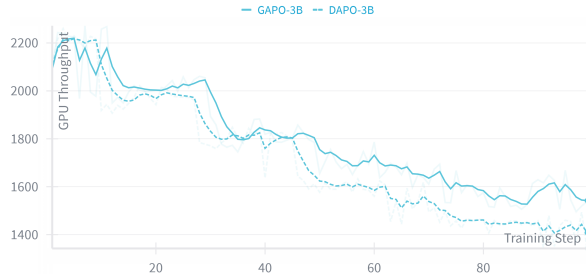


Figure 5: GPU throughput curves on the training set for DAPO with Qwen2.5-Coder-3B, as logged by W&B.

GPU throughput is closely related to training efficiency, typically measured as the number of tasks processed per unit time. Due to magnitude differences, we present results only for the 3B models. By reducing noise and enhancing training stability, **GAPO** indirectly improves GPU throughput, as shown in Figure 5. Quantitatively, **GAPO** improves the average throughput of DAPO by 4.96% compared to the baseline. The improvement gap becomes more pronounced as training progresses.

4.6 Hyperparameter Study

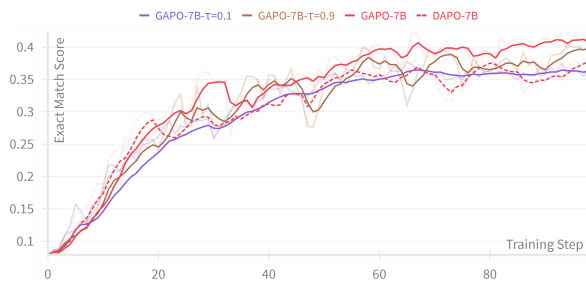


Figure 6: Performance curves on the test set for DAPO with Qwen2.5-Coder-7B using τ values of 0.1, 0.5 (default), and 0.9, as logged by W&B.

The only hyperparameter of our **GAPO** method is τ , which defines the percentage range of the dense region. We study its effects on Qwen2.5-Coder-7B, with the results shown in Figure 3 and Figure 4. Across these figures, we observe that the default value of $\tau = 0.5$ achieves the best overall performance. While $\tau = 0.9$ produces suboptimal learning curves, it exhibits higher instability, as indicated by larger `pg_clipfrac` values in DAPO frameworks. Conversely, $\tau = 0.1$ yields lower

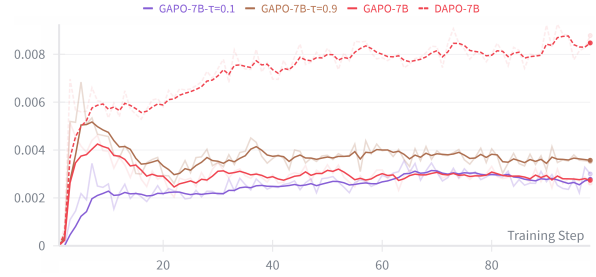


Figure 7: Clip fraction curves on the training set for DAPO with Qwen2.5-Coder-7B using τ values of 0.1, 0.5 (default), and 0.9, as logged by W&B.

accuracy but demonstrates the best stability, evidenced by smoother learning curves and smaller `pg_clipfrac` values. Therefore, τ can be tuned within the range 0.1–0.5 to balance accuracy and stability according to different requirements.

4.7 Ablation Study

Table 5: Exact match accuracy of **GAPO**’s variants on the test set using Qwen2.5-Coder (3B, 7B, 14B).

Size	3B	7B	14B
GRPO	39.69	40.05	42.64
GAPO (G) (median, div)	42.70	44.40	46.25
GAPO (G) (median, std)	39.42	40.23	44.43
Δ to GRPO	-0.27	+0.18	+1.79
Δ to GAPO (G)	-3.28	-4.17	-1.82
GAPO (G) (mean, div)	41.72	42.54	45.10
Δ to GRPO	+2.03	+2.49	+2.46
Δ to GAPO (G)	-0.98	-1.86	-1.15
DAPO	38.74	41.64	—
GAPO (D) (median, div)	39.98	43.96	—
GAPO (D) (median, std)	37.32	40.65	—
Δ to DAPO	-1.62	-0.99	—
Δ to GAPO (D)	-2.86	-3.31	—
GAPO (D) (mean, div)	38.96	41.11	—
Δ to DAPO	+0.22	-0.53	—
Δ to GAPO (D)	-1.02	-2.85	—

We have demonstrated the superiority of **GAPO** over QAE, which uses a non-adaptive quantile similar to our median; therefore, we do not perform an ablation of the adaptive HDI. In addition to using the median within the adaptive dense region as our adaptive Q , denoted **GAPO** (median, div), which modifies both the numerator and denominator in the original advantage computation of GRPO (Eq. (3)), we consider two variants:

1. **GAPO** (median, std): replaces only the numerator with the median while keeping the denominator unchanged (*i.e.*, still based on the standard deviation);
2. **GAPO** (mean, div): uses the mean instead of the median within the adaptive dense region to compute Q .

As shown in Table 5, both variants perform worse than the default **GAPO**, and in most cases even underperform the original GRPO/DAPO, particularly **GAPO** (median, std). In **GAPO** (median, std), only the numerator of the advantage computation differs from that of GRPO/DAPO, which introduces a shift in the advantage values. This shift cannot consistently eliminate the negative impact of outliers in the real-world code editing scenarios, like GMPO (Zhao et al., 2025); instead, it introduces noise and ultimately degrades performance. Larger models (*e.g.*, 14B) appear more robust to this advantage shift.

In contrast, replacing the median with the mean in both the numerator and denominator, *i.e.*, **GAPO** (mean, div), results in less performance degradation than **GAPO** (median, std). This is because **GAPO** (mean, div) leverages statistics from the dense and highest-SNR sub-region to represent the entire (potentially outlier-contaminated) action distribution, reducing the influence of outliers. The fact that **GAPO** (median, div) outperforms **GAPO** (mean, div) demonstrates that the median better suppresses the impact of outliers than the mean, yielding a more robust advantage calculation.

5 Related Work

5.1 Stabilized Variants of GRPO

To address the instability and limited exploration of GRPO, recent studies have proposed improvements from multiple perspectives. For instance, (Wei et al., 2025) highlights GRPO’s gradient instability and mitigates it by dithering discrete reward signals with noise. Dr. GRPO (Liu et al., 2025a) corrects training bias by jointly considering response length and question difficulty. GSPO (Zheng et al., 2025) mitigates high variance in Mixture-of-Experts models via sequence-level importance sampling. GRPO-MA (Wang et al., 2025a) reduces variance by averaging multi-answer rewards without calibration of reward baseline.

Another line of work enhances GRPO stability through reward baseline calibration but fails

to adapt to skewed reward distributions. GMPO (Zhao et al., 2025) simply replaces the arithmetic mean with a geometric mean to mitigate outlier sensitivity, without adaptation to varying reward distributions. KRPO (Wang et al., 2025b) relies on the Gaussian distribution assumption on rewards and hyperparameter tuning of the Kalman filter to estimate latent reward baselines and uncertainty, failing to address signal bias under skewed distributions. Moreover, QAE (Wu et al., 2025b) simply introduces a group-wise K -quantile baseline for entropy-safe reasoning, but it cannot identify noise-free regions or perform adaptive denoising.

Despite these efforts, stability in LLM RL for code tasks remains underexplored. Besides, existing methods rely on discrete validation rewards from mathematical reasoning, unlike the continuous rewards in the code-editing tasks studied here.

5.2 LLM for Code Edit

Code editing (Li et al., 2023; Guo et al., 2024b; Nam et al., 2025) is more challenging than basic code refinement, as it depends on the complex context and cursor condition. To address this, Self-Edit (Zhang et al., 2023) fine-tunes a fault-aware neural editor in a generate-and-edit manner to improve code quality and accuracy. EDITLORD (Li et al., 2025) avoids direct prompting or fine-tuning by using one LLM to extract transformation rules and a second to apply them to code pairs. IterPref (Wu et al., 2025a) leverages offline preference learning for iterative debugging, enabling context-aware code editing based on user feedback. LEDEX (Jiang et al., 2024) automatically collects refinement trajectories and enhances LLMs’ self-debugging ability via supervised fine-tuning and RL. Finally, RLEF (Gehring et al., 2024) performs code editing with real-time execution feedback, optimizing refinement through end-to-end RL.

6 Conclusion

We propose **GAPO**, a robust RL method that replaces the global mean with an adaptive group-wise (Q) to handle skewed, outlier-prone distributions in real-world code editing. Tested on 10 languages and nine LLMs (3B–14B), **GAPO** consistently outperforms alternatives with minimal overhead, improving generalization on left-skewed tasks and specialization on right-skewed ones, providing a plug-and-play solution for stable RL-based code LLM post-training.

7 Limitations

We focus on adaptively identifying the highest SNR region to mitigate noise when estimating the true reward distribution. However, identifying a single continuous high-SNR region may overlook multi-peak reward distributions, leading to inaccurate reward estimates. Additionally, the noise distribution is currently unknown and unpredictable, but it presents an avenue for future exploration.

Acknowledgments

This research is supported by Voicecomm. It is also supported by the Smart Medical Special Research Fund of the Shanghai Municipal Health Commission (Grant No. 2025ZHYL003) and the Interdisciplinary Program of Shanghai Jiao Tong University (project number YG2024QNB05). We also thank Tencent CodeBuddy (<https://www.codebuddy.ai/>) and WeChat Game team for their valuable support.

References

- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*.
- Le Deng, Xiaoxue Ren, Chao Ni, Ming Liang, David Lo, and Zhongxin Liu. 2025. Enhancing project-specific code completion by inferring internal api information. *IEEE Transactions on Software Engineering*.
- Victor Dibia, Adam Fourney, Gagan Bansal, Forough Poursabzi-Sangdeh, Han Liu, and Saleema Amershi. 2023. Aligning offline metrics and human judgments of value for code generation models. In *Findings of the Association for Computational Linguistics: ACL 2023*.
- Bernd Frauenknecht, Devdutt Subhasish, Friedrich Solowjow, and Sebastian Trimpe. 2025. On rollouts in model-based reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.
- Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. 2024. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024a. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.
- Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, and 1 others. 2024b. Codeedit-bench: Evaluating code editing capability of large language models. *arXiv preprint arXiv:2404.03543*.
- Zhezhen Hao, Hong Wang, Haoyang Liu, Jian Luo, Jiarui Yu, Hande Dong, Qiang Lin, Can Wang, and Jiawei Chen. 2025. Rethinking entropy interventions in rlvr: An entropy change perspective. *arXiv preprint arXiv:2510.10150*.
- Jakob Hollenstein, Sayantan Auddy, Matteo Saveriano, Erwan Renaudo, and Justus Piater. 2022. Action noise in off-policy deep reinforcement learning: Impact on exploration and performance. *arXiv preprint arXiv:2206.03787*.
- Hugging Face. 2025. Hugging face — the ai community building the future. <https://huggingface.co/>. Accessed: 2025-10-08.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth e Lacroix, and William El Sayed. 2023. *Mistral 7b*. *Preprint*, arXiv:2310.06825.
- Nan Jiang, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya B Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, and Anoop Deoras. 2024. Ledex: Training llms to better self-debug and explain code. *Advances in Neural Information Processing Systems*.
- Onyedikachi O John. 2015. Robustness of quantile regression to outliers. *American Journal of Applied Mathematics and Statistics*, 3(2):86–88.
- Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip HS Torr, Fahad Shahbaz Khan, and Salman Khan. 2025. Llm post-training: A deep dive into reasoning large language models. *arXiv preprint arXiv:2502.21321*.
- VI Lcvenshtcin. 1966. Binary coors capable or ‘correcting deletions, insertions, and reversals. In *Soviet physics-doklady*.

- Jia Li, Ge Li, Zhuo Li, Zhi Jin, Xing Hu, Kechi Zhang, and Zhiyi Fu. 2023. Codeeditor: Learning to edit source code with pre-trained models. *ACM Transactions on Software Engineering and Methodology*, 32(6):1–22.
- Weichen Li, Albert Jan, Baishakhi Ray, Junfeng Yang, Chengzhi Mao, and Kexin Pei. 2025. Editlord: Learning code transformation rules for code editing. *arXiv preprint arXiv:2504.15284*.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.
- Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025a. Understanding rl-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.
- Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, and 1 others. 2025b. Part i: Tricks or traps? a deep dive into rl for llm reasoning. *arXiv preprint arXiv:2508.08221*.
- David S Moore, George P McCabe, and Bruce A Craig. 2009. *Introduction to the Practice of Statistics*, volume 4. WH Freeman New York.
- Yongyu Mu, Jiali Zeng, Bei Li, Xinyan Guan, Fandong Meng, Jie Zhou, Tong Xiao, and Jingbo Zhu. 2025. Dissecting long reasoning models: An empirical study. *arXiv preprint arXiv:2506.04913*.
- Daye Nam, Ahmed Omran, Ambar Murillo, Saksham Thakur, Abner Araujo, Marcel Blistein, Alexander Frömmgen, Vincent Hellendoorn, and Satish Chandra. 2025. Prompting llms for code editing: Struggles and remedies. *arXiv preprint arXiv:2504.20196*.
- Ben O’Neill. 2022. Smallest covering regions and highest density regions for discrete distributions. *Computational Statistics*, 37(3):1229–1254.
- Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirel. 2023. The impact of ai on developer productivity: Evidence from github copilot. *arXiv preprint arXiv:2302.06590*.
- Qwen Team. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.
- Peter J Rousseeuw and Mia Hubert. 2011. Robust statistics for outlier detection. *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, 1(1):73–79.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*.
- TIOBE Software BV. 2025. *Tiobe index*. <https://www.tiobe.com/tiobe-index/>. Accessed: 2025-12-31.
- Hongcheng Wang, Yinuo Huang, Sukai Wang, Guanghui Ren, and Hao Dong. 2025a. Grpo-ma: Multi-answer generation in grpo for stable and efficient chain-of-thought training. *arXiv preprint arXiv:2509.24494*.
- Hu Wang, Congbo Ma, Ian Reid, and Mohammad Yaqub. 2025b. Kalman filter enhanced grpo for reinforcement learning-based language model reasoning. *arXiv preprint arXiv:2505.07527*.
- Junqiao Wang, Zeng Zhang, Yangfan He, Zihao Zhang, Xinyuan Song, Yuyang Song, Tianyu Shi, Yuchen Li, Hengyuan Xu, Kunyu Wu, and 1 others. 2024. Enhancing code llms with reinforcement learning in code generation: A survey. *arXiv preprint arXiv:2412.20367*.
- Chenxing Wei, Jiarui Yu, Ying Tiffany He, Hande Dong, Yao Shu, and Fei Yu. 2025. Redit: Reward dithering for improved llm policy optimization. *arXiv preprint arXiv:2506.18631*.
- Jie Wu, Haoling Li, Xin Zhang, Jianwen Luo, Yangyu Huang, Ruihang Chu, Yujiu Yang, and Scarlett Li. 2025a. Iterpref: Focal preference learning for code generation via iterative debugging. *arXiv preprint arXiv:2503.02783*.
- Junkang Wu, Kexin Huang, Jiancan Wu, An Zhang, Xiang Wang, and Xiangnan He. 2025b. Quantile advantage estimation for entropy-safe reasoning. *arXiv preprint arXiv:2509.22611*.
- Zifan Wu, Chao Yu, Chen Chen, Jianye Hao, and Hankz Hankui Zhuo. 2022. Plan to predict: Learning an uncertainty-foreseeing model for model-based reinforcement learning. *Advances in Neural Information Processing Systems*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Burak Yetiştirgen, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. *arXiv preprint arXiv:2304.10778*.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo:

An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Zed Industries. 2025. Zeta: Repository-level code edit prediction dataset. <https://huggingface.co/datasets/zed-industries/zeta>. Dataset, Hugging Face; Apache-2.0 license.

Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. 2023. Self-edit: Fault-aware code editor for code generation. *arXiv preprint arXiv:2305.04087*.

Yuzhong Zhao, Yue Liu, Junpeng Liu, Jingye Chen, Xun Wu, Yaru Hao, Tengchao Lv, Shaohan Huang, Lei Cui, Qixiang Ye, and 1 others. 2025. Geometric-mean policy optimization. *arXiv preprint arXiv:2507.20673*.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, and 1 others. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. 2025. The surprising effectiveness of negative reinforcement in llm reasoning. *arXiv preprint arXiv:2506.01347*.

A Additional Experiments

A.1 Additional Settings

We follow most of the default settings for GRPO and DAPO in ver1 with adaptive modifications for our code edit tasks, such as an input prompt length of 4096, an output length of 1024, a rollout batch size of 512, a training batch size of 32, 8 rollouts per iteration, and a total of 10 epochs. We use the default best hyperparameter settings for baseline methods. For zeta (Zed Industries, 2025), we evaluate on its “dpo” subset, since the “eval” subset contains only 33 examples.

A.2 Reward Distribution Analysis

Furthermore, we visualize representative examples sampled from input prompts that exhibit four types of reward distributions in Figure 8, providing a closer look at the skewed cases. Mean–median gaps are evident in left- and right-skewed distributions, where outliers mainly occur. By further analyzing the relationship between the mean–median difference and the sign of the advantages computed in Eq. (3) and Eq. (4), we find that our **GAPO** method generates more negative rollouts than the original GRPO/DAPO in left-skewed distributions. This occurs because Q is larger than the mean in such cases, causing most rewards to fall below Q and thus produce negative advantages. This behavior

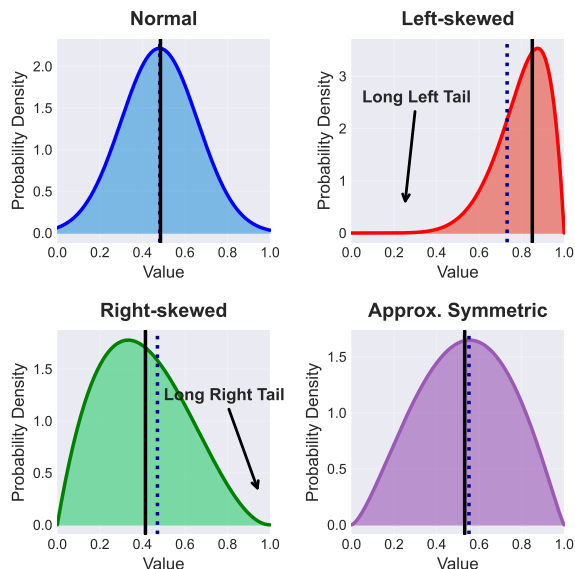


Figure 8: Examples of four types of reward distribution of rollouts before training using Qwen2.5-Coder-14B.

is reasonable, as left-skewed distributions typically correspond to *relatively easy problems* with generally higher rewards, where more negative samples promote better generalization during training (Mu et al., 2025; Zhu et al., 2025). Conversely, for right-skewed distributions (*relatively hard problems*), **GAPO** encourages more specialized learning, leading to improved accuracy on these challenging cases. These trends align with our post-training goals.