CLIFF: LEARNING PROCESS REWARDS FROM THE FIRST MISTAKE

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) has emerged as a powerful paradigm for large language model (LLM) post-training, but its reliance on coarse outcome rewards leads to limited guidance on intermediate reasoning processes. Existing approaches such as process reward modeling and on-policy distillation introduce additional constraints, such as reliance on a specialized reward model or assuming identical reasoning patterns between teacher and student. Nevertheless, we observe that once a reasoning process first goes wrong, evaluating the subsequent reasoning provides limited additional information, as it is already conditioned on an invalid prefix. Therefore, we propose **Cliff**, a process-level credit assignment strategy that utilizes an off-the-shelf LLM as a teacher to identify the first mistake in each rollout. As a result, the rollout is naturally decomposed into two parts: a correct prefix and an incorrect suffix. Cliff then converts this signal into token-level advantages, assigning higher advantages for the correct prefix and lower scores afterward. Experiments across 12 different scenarios demonstrate that Cliff consistently improves reasoning performance, outperforming on-policy distillation by 15% and standard GRPO by 7%, even with teachers of modest capability. Furthermore, we analyse the role of "ground truth" in Cliff and investigate its training dynamics. These results establish Cliff as a simple, general and effective approach for improving RLVR with richer, fine-grained supervision.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has become the standard paradigm for post-training large language models (LLMs) on reasoning tasks. By leveraging automatically verifiable outcomes, RLVR enables scalable, unbiased reinforcement learning without human annotations and has demonstrated remarkable progress in various domains. Despite its success, the outcome reward used in RLVR is inherently coarse-grained as it only evaluates the final result of a reasoning trajectory. As a consequence, it cannot distinguish between reasoning processes of vastly different quality. For example, a nearly complete solution containing only a minor error is penalized identically to a completely incorrect attempt. Such sparse, result-oriented supervision provides limited guidance on where the model succeeds or fails during reasoning, reducing learning efficiency and making it difficult to assign credit to individual reasoning steps.

Prior work has explored several strategies to incorporate fine-grained supervision into reinforcement learning, where two prominent directions are Process Reward Models (PRMs) and On-Policy Distillation (OPD). However, these approaches have notable limitations as well. PRMs require training an additional reward model, making it less generalizable and susceptible to reward hacking (Zheng et al., 2024; Tiwari et al., 2026). OPD only achieves optimal performance when the teacher and student share similar reasoning patterns and are in the same family (Fu et al., 2026). These constraints limit their applicability in general RLVR settings.

To overcome these limitations, we ask a more fundamental question: *how fine-grained does a process signal really need to be?* Our key insight is that process supervision does not necessarily need to precisely evaluate every intermediate step, or every token. Once a reasoning process goes wrong, evaluating the subsequent reasoning may provide limited additional information, as it is already conditioned on an invalid prefix. This intuition is inspired by the notion of **Vacuous Implication** in formal logic: when the antecedent A is false, the implication "$A \rightarrow B$" is true regardless of B.

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.

While this is not a formal proof for our method, it motivates a simple design principle: *we only need to identify where the reasoning first goes wrong, rather than precisely evaluate everything that follows*.

Based on this principle, we propose **Cliff**, a process-level credit assignment method that divides each rollout exactly once at its first mistake, yielding only two segments—a correct prefix and an incorrect suffix—and uses this coarse-grained structure to provide process supervision without requiring token- or step-level rewards. Specifically, we utilize a teacher model to judge student rollouts and locate the first mistake in the student's reasoning. For a rollout that fails to reach the final answer, tokens before the first mistake receive relatively higher advantage scores, while tokens after the boundary receive negative feedback just like GRPO.

This decomposition offers several desirable properties. First, Cliff retains the simple formulation of RLVR while providing more informative process-level feedback: instead of assigning a single outcome-level reward to the entire rollout, Cliff distinguishes the valid reasoning from the point at which it first becomes invalid. Second, Cliff is both model-agnostic and task-agnostic. The supervision depends only on whether the reasoning remains correct up to a given point, rather than on the domain or the teacher's reasoning patterns. Thus, it can be applied across different models and tasks, provided that an evaluator is capable of identifying the first mistake. Finally, this design is naturally resistant to reward hacking. Since the reward is determined by the location of the first mistake, and postponing a mistake is always preferable to making the same mistake earlier, while a completely correct rollout remains optimal by definition.

Empirically, we first show LLM teachers can judge student answers and find out the first problematic reasoning accurately in Section 4, even when the model's own performance isn't perfect. We then show that Cliff outperforms on-policy distillation by 15% and GRPO by 7% across 12 different scenarios in Section 5. We also demonstrate that incorporating ground truth is useful for weaker judge models, and analyse the training dynamics of Cliff. Overall, Cliff provides a simple yet general framework for transforming coarse outcome-based supervision into informative learning signals, paving the way toward more scalable and reliable reinforcement learning for LLMs.

2 Related Work

Reinforcement Learning with Verifiable Rewards (RLVR). Due to its effectiveness and scalability, reinforcement learning (RL) has become a powerful paradigm for LLM post-training for reasoning tasks (Qian et al., 2026; Han et al., 2025; Yu et al., 2025), and have been widely used by state-of-the-art LLMs (Guo et al., 2025; Jaech et al., 2024; Team et al., 2025b). Common algorithms for RLVR includes PPO (Schulman et al., 2017), GRPO (Shao et al., 2024) and DAPO (Yu et al., 2026). Many recent work focuses on understanding Zeng et al. (2025b); Mroueh (2025); Wang et al. (2026); Zhai et al. (2026) and improving these methods, namely Dr.GRPO (Liu et al., 2025), Clip-Cov (Cui et al., 2025) and Self-aligned Reward (Han et al., 2026).

LLM Distillation and Reward Modeling. Distillation and reward modeling are both knowledge transfer approaches for LLMs. Knowledge distillation typically trains a student model to imitate a teacher's behavior (Kim & Rush, 2016; Jiao et al., 2020; Wang et al., 2020). On-policy distillation (OPD), as a more recent approach, trains the student on its own trajectories. OPD has shown strong performance (Agarwal et al., 2024; Lu & Lab, 2025; Zhao et al., 2026b; Yang et al., 2026a) and attracts increasing analysis on mechanisms (Li et al., 2026; Song & Zheng, 2026; Yang et al., 2026b), but its requirement for identical tokenizer and reasoning patterns limit its broader application (Li et al., 2026; Fu et al., 2026; Boizard et al., 2024).

Reward models (RMs) are a key component of LLM post-training and alignment. They can be trained to generate scalar scores (Ouyang et al., 2022; Rafailov et al., 2024; Li et al., 2024; Zhao et al., 2026c) as well as textual judgments (Zhao et al., 2026a; Chen et al., 2025). Although commonly applied to unverifiable tasks, RMs have also proven effective for reasoning tasks by introducing process reward (Setlur et al., 2025; Lightman et al., 2024; Wang et al., 2024; Song et al., 2025). However, some recent work suggests that process RMs require extensive data to ensure generalizability (Zheng et al., 2024; Zeng et al., 2025a), and often suffer from reward hacking (Cheng et al., 2026; Tiwari et al., 2026) or degeneration to outcome rewards (Zhang et al., 2025).

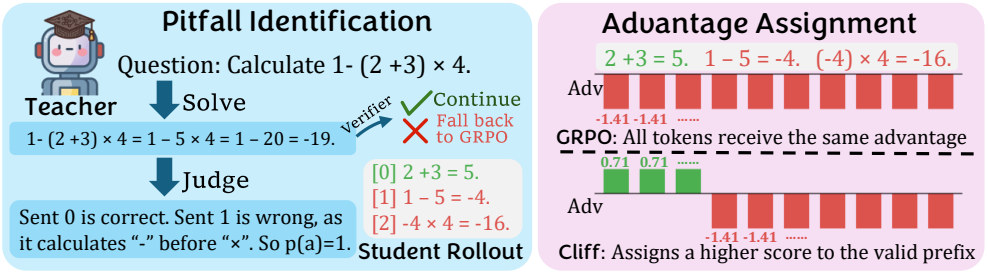

Figure 1: Overview of **Cliff**.

3 METHODOLOGY

3.1 GRPO

Cliff is an extension based on Group Relative Policy Optimization (GRPO) (Shao et al., 2024), a widely adopted algorithm in reinforcement learning. In the GRPO formulation, we denote the current policy as π_θ, the training set as D, and the outcome reward from an automatic verifier as R. For each step, we sample a query $q \sim D$, and several rollouts $a_{1:N} \sim \pi_{\theta_{\text{old}}}(q)$. The objective function can be written as[1]:

$$\mathcal{J}_{\text{GRPO}}(\pi_\theta) = \mathbb{E}\left[\frac{1}{\sum_{i=1}^{N}|a_i|}\sum_{i=1}^{N}\sum_{j=0}^{|a_i|-1}\left(\min\left\{r_{i,j}A_{i,j}, \text{clip}(r_{i,j}, 1-\epsilon, 1+\epsilon)A_{i,j}\right\} - \beta\,\mathbb{D}_{\text{KL}}[\pi_\theta\|\pi_{\text{ref}}]_{i,j}\right)\right]. \tag{1}$$

In the formula, $r_{i,j} = \frac{\pi_\theta(a_{i,j}|q, a_{i,<j})}{\pi_{\theta_{\text{old}}}(a_{i,j}|q, a_{i,<j})}$ is the importance sampling ratio and $A_{i,j}$ is the advantage:

$$A_{i,j} = \frac{R(q, a_i) - \text{mean}[R(q, a_1), ..., R(q, a_N)]}{\text{std}[R(q, a_1), ..., R(q, a_N))]}. \tag{2}$$

3.2 CLIFF

To provide universal, fine-grained process supervision for RLVR, we propose **Cliff**, which identifies the first mistake in a student rollout and decomposes it into a correct prefix and an incorrect suffix. An overview of Cliff is illustrated in Figure 1.

3.2.1 PITFALL STEP IDENTIFICATION

Cliff leverages a stronger teacher model to provide process-level feedback for each student rollout. The supervision is performed in two stages. In the first stage, the teacher independently generates a reference solution for each query. We then use the automatic verifier to evaluate the teacher's solution, and only keep groups where the teacher itself solves correctly, or $R(q, a_{\text{tea}}) = 1$. If the teacher's solution is incorrect, the corresponding group falls back to vanilla GRPO, as the teacher's guidance is no longer reliable.

In the second stage, the teacher judges each student rollout based on the question and its own verified solution. The teacher first determines whether the student's answer is correct and, if it is incorrect, identifies the first reasoning mistake. Note that the teacher model, instead of the automatic verifier, gives outcome rewards in the second stage—this enables the teacher to judge the correctness of the entire reasoning process, including cases where the final answer alone may not faithfully reflect the validity of the reasoning (Section 4 discusses the discrepancy between the teacher and the verifier).

We refer to the first sentence at which the reasoning becomes incorrect as the *Pitfall Step*[2]. denoted as $p_{\text{sent}}(a)$. The Pitfall Step serves as the boundary that separates each rollout into a correct prefix and an incorrect suffix. Thus, each incorrect rollout can be decomposed into a valid prefix

[1] In practice, we use DAPO's token-level loss aggregation (Yu et al., 2026).

[2] We instruct the teacher to not consider failed attempts corrected later as a mistake. See Section E.

before $p_{\text{sent}}(a)$ and a problematic suffix after it.[3] We carefully curate the judging prompt to ensure the teacher focuses on genuine reasoning errors rather than harmless typos or different but valid solution strategies (prompts are shown in Section E).

3.2.2 CLIFF ADVANTAGE ASSIGNMENT

After obtaining the Pitfall Step, we convert the teacher's judgment into token-level advantages for GRPO. The key idea is to refine the outcome-level advantage of GRPO by distinguishing the valid prefix from the erroneous suffix within an incorrect rollout.

We first calculate the GRPO-style advantage. The teacher's judgment on answer correctness (denoted as R_{tea}) is binary, so the GRPO-style advantage values within a group take only two possible values: A_{cor} shared by all tokens in correct rollouts and A_{inc} shared by all tokens in incorrect rollouts. Let the group size be N, and the group's reward statistics and advantages are therefore given by

$$\mu = \frac{\sum_{i\in 1:N} R_{\text{tea}}(q, a_i)}{N}, \qquad \sigma = \sqrt{\mu(1-\mu)}, \qquad A_{\text{cor}} = \frac{1-\mu}{\sigma}, \qquad A_{\text{inc}} = \frac{0-\mu}{\sigma}. \tag{3}$$

We then use the Pitfall Step to refine the advantage within each rollout. For an incorrect rollout, tokens before the Pitfall Step correspond to reasoning that remains valid, whereas tokens at and after the Pitfall Step belong to the reasoning that leads to the incorrect outcome. We therefore give a relatively higher advantage for the valid prefix while assigning the negative outcome-level advantage to the erroneous suffix. Let the first token in the Pitfall Step sentence be $p(a)$, the resulting token-level advantage in Cliff is:

$$A_{i,j} = \begin{cases} A_{\text{cor}} - b, & R_{\text{tea}}(q, a_i) = 1, \\ \lambda A_{\text{cor}} - b, & R_{\text{tea}}(q, a_i) = 0 \text{ and } j < p(a_i), \\ A_{\text{inc}} - b, & R_{\text{tea}}(q, a_i) = 0 \text{ and } j \geq p(a_i). \end{cases} \tag{4}$$

Here, R_{tea} refers to the answer correctness determined by the teacher model, and $\lambda \geq 0$ is a hyperparameter that controls the strength of positive reinforcement assigned to the *valid prefix* of an incorrect rollout. Empirically, we use $\lambda = 0$ as it avoids length hacking (see Section 6.2 and Section D). The offset term b is introduced to ensure that the token-level advantages have zero mean across the group. Specifically, the offset b is equal to the average token-level advantage before this recentering operation:

$$b = \frac{A_{\text{cor}} \sum_{i:\, R_{\text{tea}}(q,a_i)=1} |a_i| + \sum_{i:\, R_{\text{tea}}(q,a_i)=0} \left[\lambda A_{\text{cor}} p(a_i) + A_{\text{inc}}\big(|a_i| - p(a_i)\big)\right]}{\sum_{i=1}^{N} |a_i|}. \tag{5}$$

[4]

4 JUDGE QUALITY OF CLIFF

This section conducts a preliminary experiment to evaluate whether the teacher model can reliably identify the "Pitfall Step" in student reasoning trajectories. We compare the teacher's judgments with human annotations from two perspectives: its ability to determine whether a student solution is correct and its ability to accurately localize the first reasoning mistake.

4.1 SETTINGS

To facilitate a controlled evaluation, we construct a small, human-annotated dataset from DAPO-Math. We first sample a large number of student rollouts and then select 50 correct and 50 incorrect rollouts to form a balanced dataset. For each incorrect rollout, human experts are asked to identify the first reasoning step that leads to the incorrect conclusion (i.e., the Pitfall Step). We provide

[3] For overlength rollouts, we directly set $p_{\text{sent}}(a) = 0$, treating the entire sequence as problematic. This is both for efficiency and to avoid length hacking (see Section D).

[4] Token positions are indexed from 0, so the valid prefix $\{j < p(a_i)\}$ contains exactly $p(a_i)$ tokens.

Table 1: Teacher models' reference solution accuracy, judge consistency with the automatic verifier, and the similarity of Pitfall Step positions with human annotators.

Teacher Model	Ref. Solution	Ref. Acc.	Judge Cons.	# FP	# FN	Avg. p-dis	# p-dis ≤ 1
Qwen3-32B	Provided by the Teacher	65	88	1	11	3.00	68
Gemma3-27B		67	86	0	14	4.52	39
GPT-5.6 Sol[5]		93	91	0	9	1.23	82
Qwen3-32B	Ground Truth	100	91	0	9	3.30	57
Gemma3-27B			93	0	7	3.77	52
GPT-5.6 Sol			92	0	8	1.21	80
Qwen3-32B	An Incorrect Solution	0	60	4	36	4.70	42
Gemma3-27B			51	5	44	5.56	44
GPT-5.6 Sol			81	4	15	2.05	73

human annotators with the same instructions used by the LLM judge, ensuring that the two types of judgments are directly comparable.

4.2 Judge Performance

We evaluate the teacher models from three complementary perspectives. First, we measure each teacher's problem-solving accuracy by allowing it to solve the problems independently. Second, we measure the consistency between final answer correctness judged by the verifier and by the teacher LLM. Finally, for incorrect student solutions, we further compare the Pitfall Steps identified by the LLM judge and human experts. To quantify the alignment between human experts and the LLM judge, we define

$$\text{p-dis}(a) = |p_{\text{sent}}^{\text{human}}(a) - p_{\text{sent}}^{\text{LLM}}(a)|, \tag{6}$$

where $p_{\text{sent}}^{\text{human}}(a)$ and $p_{\text{sent}}^{\text{LLM}}(a)$ denote the sentence indices of the first incorrect reasoning step identified by the human annotator and the LLM, respectively. A smaller p-dis indicates closer agreement between the two, with a value of zero meaning that the LLM identifies exactly the same pitfall step as the human expert.

From the upper part of Table 1, we can draw the following conclusions:

• **Strong problem-solving ability leads to reliable process judgment.** GPT-5.6 Sol achieves the highest problem-solving accuracy and also produces the most reliable judgments, with its identified Pitfall Steps showing strong agreement with human annotations. This suggests that highly capable models can not only solve mathematical problems effectively, but also reliably localize the first reasoning error in incorrect student solutions.

• **Judging student rollouts is easier than solving problems.** Although Qwen3-32B and Gemma3-27B obtain noticeably lower problem-solving accuracy than GPT-5.6 Sol, they can still reliably identify Pitfall Steps and maintain good agreement with human experts. This observation is encouraging for Cliff, as it suggests identifying the mistake might be substantially easier than solving the problem from scratch.

• **Most discrepancies come from false negatives.** The judge rarely incorrectly rejects a correct student solution (i.e., false positives), while false negatives occur in approximately 10% of cases. Importantly, some of these false negatives may not indicate genuinely unreliable judgment, as the automatic verifier may falsely accept solutions that arrive at the correct final answer through guessed answers or flawed reasoning.

4.3 Impact of Reference Solution Quality

This section analyzes how the quality of the reference solution affects the teacher's judgment. Specifically, we investigate two variants: in the first variant, the teacher is provided with a correct reference solution, while in the second variant, it is intentionally provided with an incorrect reference solution. As shown in Table 1, the quality of the reference solution has a clear impact on

judgment performance. Providing a correct reference solution leads to higher judgment accuracy and more accurate localization of the Pitfall Step, whereas an incorrect reference solution degrades both metrics and results in a larger p-dis.

This finding also motivates the reference-solution filtering procedure in Cliff. As described in Section 3, we use the automatic verifier to filter the teacher's reference solution and only proceed when the reference solution is verified as correct. **Therefore, the "Ground Truth" setting provides a more faithful estimate of the judge's performance in actual Cliff training.** Under this setting, the judgment accuracy exceeds 90% and the average p-dis is only around 3 sentences, indicating that the teacher can reliably identify the first reasoning error given a reliable reference solution.

4.4 Case Studies

To better understand the behavior of Cliff, we present several representative examples in Section F using Qwen3-32B as the teacher model. These examples provide qualitative evidence for the effectiveness of the proposed judging framework. We find that the teacher can analyze the student's reasoning step by step and accurately identify the earliest reasoning error in most cases, rather than simply checking whether the final answer is correct. In particular, the teacher is able to detect problematic reasoning even when a student arrives at the correct final answer, demonstrating that its judgment captures the validity of the reasoning process beyond the outcome alone. While the teacher's $p(a)$ does not always exactly match that of human annotators, which reflects the inherent ambiguity in localizing the first mistake, its decisions are generally reasonable. Overall, these examples suggest that Cliff can provide meaningful process-level supervision.

5 Experiments

5.1 Settings

Models. We utilize two student models: Qwen3-4B-Base (Yang et al., 2025) and Phi-4-mini-Instruct Abdin et al. (2024). We apply supervised finetuning (SFT) to Qwen3-4B-Base on OpenThoughts (Guha et al., 2025) before RL to provide instruction-following ability. We choose three teacher models: a state-of-the-art LLM: GPT-5.6 Sol (OpenAI, 2026), and two open-source models: Qwen3-32B, Gemma3-27B (Team et al., 2025a).

Datasets. We test Cliff on two domains: math reasoning and coding. For math reasoning, the models are trained on DAPO-math-17k-processed (Yu et al., 2026); for coding, the models are trained on Deepcoder (Luo et al., 2025). Details of all training and evaluation datasets are listed in Section A. Notably, we use a binary reward for coding, only giving a score of 1 when the code passes all test cases.

Baselines. We compare Cliff against the following baselines:

- GRPO: The prevalent RL algorithm, using an automatic verifier to provide sequence-level signals.
- GRPO with Teacher: Use the teacher to judge whether the student rollout is correct, but the same advantage is applied to the entire rollout.
- Distillation (Kim & Rush, 2016): Sample responses from the teacher, and train student models in a supervised fine-tuning style.
- On-policy Distillation (OPD): Use the teacher's distribution on student rollouts to provide token-level signals. Only applied for open-source teachers[6].

Training Details. We show training details in Section B.

5.2 Main Results

The main experimental results are reported in Table 2. We make the following observations:

[6]We use the k1 estimator (Lu & Lab, 2025) for same-family OPD and the method from Niu et al. (2026) for cross-tokenizer OPD.

Table 2: **Cliff**'s performance on math and coding benchmarks.

Setting	Teacher	Math Reasoning					Algorithmic Coding			
		GSM8k	MATH-500	DAPO	AIME	Avg.	CodeContests	LiveCode	DeepCoder	Avg.
Qwen3-4B	N/A	84.53	71.00	29.40	17.69	50.66	14.25	20.29	10.60	15.05
+ GRPO		92.80	79.00	42.90	32.01	61.68	23.75	27.66	21.20	24.20
+ Distill	GPT-5.6	90.978	74.40	38.40	30.54	58.58	21.25	25.37	15.20	20.61
+ GRPO		92.04	80.00	44.00	33.44	62.37	23.25	28.64	22.60	24.83
+ **Cliff**		**93.17**	**83.20**	**49.30**	**36.98**	**65.66**	26.25	27.82	23.80	**25.96**
+ Distill	Qwen	92.722	75.4	38.2	27.44	58.44	22.00	26.35	20.60	22.98
+ OPD		91.21	77.20	37.90	26.37	58.17	19.25	25.70	16.20	20.38
+ GRPO		92.12	78.40	44.90	29.37	61.20	25.50	27.66	23.40	25.52
+ **Cliff**		92.12	**80.80**	**48.90**	**36.65**	**64.62**	26.50	28.15	23.40	**26.02**
+ Distill	Gemma	92.722	72.00	33.10	25.83	55.91	25.00	27.82	20.40	24.41
+ OPD		91.13	74.00	33.20	22.62	55.24	18.50	26.64	18.80	21.31
+ GRPO		91.73	78.40	45.50	32.44	62.02	24.00	29.13	21.40	24.84
+ **Cliff**		92.72	**80.60**	**47.50**	**33.97**	**63.70**	27.25	27.50	22.00	**25.58**
Phi-4-Mini	N/A	86.58	49.40	17.00	7.93	40.23	18.25	19.80	16.80	18.28
+ GRPO		88.93	66.40	29.00	14.79	49.78	20.00	23.57	21.60	21.72
+ Distill	GPT-5.6	88.779	59.00	23.40	15.43	46.65	20.25	20.13	19.60	19.99
+ GRPO		**89.39**	66.60	28.00	15.43	49.86	24.50	25.53	24.80	**24.94**
+ **Cliff**		89.08	**69.00**	**31.80**	**17.04**	**51.73**	23.75	26.84	24.20	24.93
+ Distill	Qwen	91.13	70.20	24.90	14.89	50.28	17.75	24.39	18.60	20.25
+ OPD		89.39	65.60	24.70	12.76	48.11	19.25	22.13	19.80	20.39
+ GRPO		87.14	68.40	**29.30**	16.29	50.18	23.25	24.55	20.20	22.67
+ **Cliff**		**90.06**	**69.40**	29.10	**17.36**	**51.48**	24.25	24.88	21.60	**23.58**
+ Distill	Gemma	90.296	64.40	22.00	14.58	47.82	20.00	23.08	18.60	20.56
+ OPD		87.26	52.00	24.90	13.43	44.40	18.75	21.77	19.40	19.97
+ GRPO		**90.42**	65.60	28.00	15.51	49.88	23.50	21.73	20.80	22.01
+ **Cliff**		89.23	**68.60**	**29.60**	**16.18**	**50.90**	24.25	23.90	21.60	**23.25**

• **GRPO provides a stable baseline.** GRPO achieves decent performance across all experimental settings, improving over the SFT-style distillation and OPD baselines in most cases.

• **Cliff consistently outperforms all other methods.** Cliff achieves the best performance under every evaluated setting. The improvements are observed across different student models and benchmark domains, demonstrating the effectiveness and generality of the proposed framework.

• **Cliff is not overly dependent on a strong teacher.** As expected, employing a stronger teacher generally leads to better student performance, with the GPT-5.6 Sol teacher consistently outperforming their open-source counterparts. However, the performance gap is relatively modest, suggesting that Cliff does not require a near-perfect teacher. Even moderately capable teacher models can identify reasoning pitfalls and provide useful training signals, making Cliff practical and scalable in settings where access to frontier LLMs is limited.

• **The gain comes from Cliff's credit assignment rather than merely introducing a teacher.** We use "GRPO with teacher" as an ablation setting to isolate the benefit of using a teacher from that of Cliff's two-fold credit assignment. This setting yields only marginal improvements over the original GRPO objective, while Cliff produces substantially larger gains. This comparison highlights that the key contribution of Cliff is not merely using an LLM as a reward model, but explicitly identifying the Pitfall Step and assigning credit separately to the reasoning before and after the first mistake. Such fine-grained credit assignment provides substantially more informative supervision than treating the entire reasoning trajectory as a single unit.

6 Analysis

6.1 Cliff without Ground Truth Filter

In the main experiments, Cliff employs a filtering procedure in which the teacher's self-generated reference solution is verified by the automatic verifier. Cliff is applied only to questions for which the

teacher produces a verified correct solution, and vanilla GRPO with feedback from the auto verifier is used otherwise. However, such ground-truth verification is not always available in real-world settings. A natural question is whether Cliff can improve reasoning even in scenarios without ground truth, so in this section, we conduct an experiment without the filtering step. In this scenario, all process-level feedback is derived solely from the teacher's own reference solution and its judgments.

From Table 3, we can find that using GPT-5.6 Sol as the teacher achieves almost the same performance in settings with and without the ground-truth verifier, indicating that its self-generated reference solutions are already sufficiently accurate and reliable. In contrast, weaker teacher models such as Qwen3-32B and Gemma3-27B experience a performance drop of around 2% when the ground-truth filter is removed. For these teacher models, removing the ground-truth filter means some inaccurate reference solutions will be used to judge student rollouts, disturbing the student's training signals. This suggests that **the primary function of the ground-truth filter is to compensate for the reference solution quality of less capable teachers**, while a strong teacher can provide reliable guidance on its own in most cases.

Table 3: Comparison of Cliff with and without ground truth.

Student	Teacher	Use GT	GSM8k	MATH-500	DAPO	AIME	Avg.
Qwen3-4B	GRPO	Yes	92.80	79.00	42.90	32.01	61.68
	GPT-5.6	Yes	93.17	83.20	49.30	36.98	65.66
		No	93.25	81.20	48.50	36.65	64.90
	Qwen	Yes	92.12	80.80	48.90	36.65	64.62
		No	90.11	78.80	47.70	34.84	62.69
	Gemma	Yes	92.72	80.60	47.50	33.97	63.70
		No	89.94	78.60	46.60	34.27	62.35
Phi-4-Mini	GRPO	Yes	88.93	66.40	29.00	14.79	49.78
	GPT-5.6	Yes	89.08	69.00	31.80	17.04	51.73
		No	89.91	71.20	27.90	16.82	51.46
	Qwen	Yes	90.06	69.40	29.10	17.36	51.48
		No	89.39	67.20	27.40	14.88	49.72
	Gemma	Yes	89.23	68.60	29.60	16.18	50.90
		No	87.80	64.10	26.90	14.41	48.30

Notably, even open-source teacher models without ground-truth filtering still often outperform vanilla GRPO. This is consistent with the finding in Section 4 that judging student rollouts is easier than solving problems from scratch, and further demonstrates that **Cliff can provide useful process-level supervision even when reliable ground-truth solutions are unavailable.**

6.2 The Choice of λ

In Cliff, the hyperparameter λ controls the amount of positive advantage assigned to the correct prefix of an incorrect reasoning trajectory, which is specified in Equation (4). To investigate the effect of λ, we conduct Cliff training with three different values.

Table 4: Cliff with different λ values.

Setting	GSM8K	MATH-500	DAPO-math	AIME	Avg. Acc.	Avg. Len.
GRPO	92.80	79.00	42.90	32.01	61.68	1279
$\lambda = 0.0$	**93.17**	**83.20**	**49.30**	**36.98**	65.66	1506
$\lambda = 0.5$	92.873	82.8	47.2	35.798	64.67	1481
$\lambda = 1.0$	93.252	81.2	47.8	33.655	63.98	1959

From the results in Table 4, we can observe that $\lambda = 0$ provides the best performance with reasonable generation length, and $\lambda = 0.5$ achieves comparable performance. Increasing λ to 1.0, however, leads to substantially longer responses and a noticeable degradation in reasoning performance. We attribute this behavior to the potential incentive for length hacking introduced by a large λ. When the valid prefix of an incorrect rollout receives a positive advantage, a larger λ provides a stronger incentive to preserve and extend the already-correct portion of the reasoning before reaching more difficult or error-prone steps. In the extreme case of $\lambda = 1.0$, the model may benefit from producing longer correct prefixes even when the additional reasoning is uninformative, thereby increasing response length without improving the final outcome. This lengthening effect can also make the model less effective at progressing toward the actual solution.

In conclusion, $\lambda = 0$ **leads to the most preferable results and is therefore used in our main experiments.** Specifically, $\lambda = 0$ means the correct prefix of an incorrect rollout receives an advantage of $A = -b$, without additional positive reinforcement. The theoretical analysis in Section D provides a formal characterization of this potential length-hacking behavior with similar conclusions,

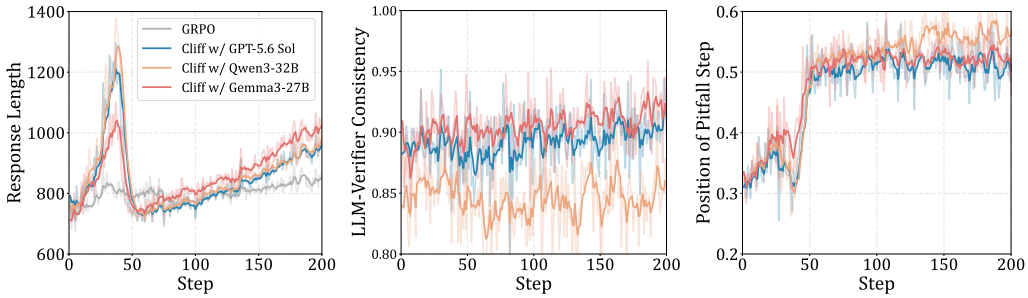

Figure 2: Training plots of GRPO and Cliff[7]. The student model is Phi-4-Mini-Instruct trained on the math domain.

showing that $\lambda = 0$ is free from length hacking, while a sufficiently large λ value may promote longer but meaningless rollouts.

6.3 Training Dynamics

In this section, we analyze the evolution of different metrics throughout the training process to better understand the training dynamics of Cliff. The metrics are illustrated in Figure 2.

First, we examine the trend of response length during training. All settings gradually develop longer reasoning trajectories, indicating that the models learn to conduct deeper reasoning to math problems. Cliff generally produces longer responses than vanilla GRPO, with a noticeable spike in response length during the early stage of training. This suggests that Cliff initially encourages the model to explore longer reasoning trajectories before the generation length gradually stabilizes.

Second, we examine the consistency between teacher judgments and the automatic verifier. While the consistency is pretty high (85% to 90%), there are a non-negligible fraction of responses judged negatively by the teacher while being accepted by the verifier. This discrepancy is expected because the verifier primarily evaluates the final answer, whereas the teacher also considers the validity of the reasoning process. We also observe lower consistency for the Gemma teacher, indicating its relatively limited capability in reliably judging student reasoning.

Finally, we track the relative position of the Pitfall Step in the whole answer for incorrect rollouts. We observe that this ratio drastically increases in around 50 steps to 0.5, and then gradually stabilizes. This suggests that, early in training, the model learns to complete early steps of reasoning even for difficult problems, leading to relatively longer valid prefixes. As training progresses, the metric reaches a stable level, indicating that the model has established a more consistent reasoning process instead of hacking the teacher model. These dynamics suggest that Cliff encourages improvements in intermediate reasoning rather than merely optimizing the final answer.

7 Conclusion

In this work, we introduce **Cliff**, a simple yet effective reward shaping algorithm that identifies the first mistake (Pitfall Step) in a rollout and assigns different advantages to tokens before and after it. Unlike approaches that rely on additional reward models or restrictive assumptions about reasoning trajectories, Cliff provides a task-agnostic and model-agnostic learning signal that works with teachers of different capabilities, bringing 7% performance gain over vanilla GRPO on average. We also analyse the impact of ground truth and the training dynamics of Cliff. Overall, this work proposes a direction to improve RLVR of LLMs with better ways of interpreting and utilizing the simple signals, instead of incorporating complex supervision mechanisms. In the future, we plan to extend the method to agentic settings and explore rule-based Pitfall Step detectors.

[7]Consistency refers to the fraction of rollouts' outcome rewards judged the same by the teacher model and automatic verifier. The position of Pitfall Step means the average fraction of tokens in the valid prefix (before the first mistake) of an incorrect rollout.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, design or provide feedback on research methodology or experiments, interpret results, and assist with translation. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, clean and reformat datasets, generate synthetic datasets, or support qualitative and thematic data analysis. Additionally, we used generative AI tools to identify relevant literature and edit the research paper to improve readability. We have humanly reviewed all AI-assisted work to ensure its correctness, including code, paper content, and relevant work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections A, B and 5.2 describe details to reproduce this paper. We will release the code via GitHub.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Nicolas Boizard, Kevin El Haddad, Céline Hudelot, and Pierre Colombo. Towards cross-tokenizer distillation: the universal logit distillation loss for llms. *arXiv preprint arXiv:2402.12030*, 2024.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, et al. Rm-r1: Reward modeling as reasoning. *arXiv preprint arXiv:2505.02387*, 2025.

Jie Cheng, Gang Xiong, Ruixi Qiao, Lijun Li, Chao Guo, Junle Wang, Yisheng Lv, and Fei-Yue Wang. Stop summation: Min-form credit assignment is all process reward model needs for reasoning. *Advances in Neural Information Processing Systems*, 38:131646–131671, 2026.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Yuqian Fu, Haohuan Huang, Kaiwen Jiang, Jiacai Liu, Zhuo Jiang, Yuanheng Zhu, and Dongbin Zhao. Revisiting on-policy distillation: Empirical failure modes and simple fixes. *arXiv preprint arXiv:2603.25562*, 2026.

Etash Kumar Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Rea Sprague, et al. Openthoughts: Data recipes for reasoning models. In *First Workshop on Foundations of Reasoning in Language Models*, 2025.

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

Peixuan Han, Zijia Liu, and Jiaxuan You. Tomap: Training opponent-aware llm persuaders with theory of mind. *arXiv preprint arXiv:2505.22961*, 2025.

Peixuan Han, Adit Krishnan, Gerald Friedland, Jiaxuan You, and Luyang Kong. Self-aligned reward: Towards effective and efficient reasoners. In *The International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10011225.

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

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of the association for computational linguistics: EMNLP 2020*, pp. 4163–4174, 2020.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Dongheng Li, Yongchang Hao, and Lili Mou. Llmr: Knowledge distillation with a large language model-induced reward. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 10657–10664, 2024.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

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. *arXiv preprint arXiv:2503.20783*, 2025.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025. Notion Blog.

Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification. *arXiv preprint arXiv:2503.06639*, 2025.

Yifan Niu, Han Xiao, Dongyi Liu, Zelong Wang, Dihong Gong, Yasheng Wang, and Jia Li. Breaking the tokenizer barrier: On-policy distillation across model families. *arXiv preprint arXiv:2606.09456*, 2026.

OpenAI. Gpt-5.6, 2026. URL https://openai.com/index/gpt-5-6/.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL https://arxiv.org/abs/2305.18290.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 60808–60838, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models. *arXiv preprint arXiv:2604.00626*, 2026.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. Prmbench: A fine-grained and challenging benchmark for process-level reward models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25299–25346, 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, et al. Gemma 3 technical report, 2025a. URL https://arxiv.org/abs/2503.19786.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025b.

Rishabh Tiwari, Aditya Tomar, Udbhav Bamba, Monishwaran Maheswaran, Heng Yang, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Reward under attack: Analyzing the robustness and hackability of process reward models. *arXiv preprint arXiv:2603.06621*, 2026.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *Advances in Neural Information Processing Systems*, 38:115452–115486, 2026.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shenzhi Yang, Guangcheng Zhu, Bowen Song, Haobo Wang, Mingxuan Xia, Xing Zheng, Yingfan Ma, Zhongqi Chen, Weiqiang Wang, Junbo Zhao, et al. Oprd: On-policy representation distillation. *arXiv preprint arXiv:2606.06021*, 2026a.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation. *arXiv preprint arXiv:2602.12125*, 2026b.

Haofei Yu, Zhengyang Qi, Yining Zhao, Kolby Nottingham, Keyang Xuan, Bodhisattwa Prasad Majumder, Hao Zhu, Paul Pu Liang, and Jiaxuan You. Sotopia-rl: Reward design for social intelligence. *arXiv preprint arXiv:2508.03905*, 2025.

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

Thomas Zeng, Shuibai Zhang, Shutong Wu, Christian Classen, Daewon Chae, Ethan Ewer, Minjae Lee, Heeju Kim, Wonjun Kang, Jackson Kunde, et al. Versaprm: Multi-domain process reward model via synthetic reasoning data. *arXiv preprint arXiv:2502.06737*, 2025a.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025b.

Zepeng Zhai, Meilin Chen, Jiaxuan Zhao, Junlang Qian, Lei Shen, and Yuan Lu. Rewards as labels: Revisiting rlvr from a classification perspective, 2026. URL https://arxiv.org/abs/2602.05630.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025.

Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, et al. Genprm: Scaling test-time compute of process reward models via generative reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34932–34940, 2026a.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models, 2026b. URL https://arxiv.org/abs/2601.18734.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards. In *International Conference on Learning Representations*, volume 2026, pp. 2548–2581, 2026c.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

A DATASETS

For math datasets, we use no system prompt and only the raw question as the user prompt. During evaluation, we parse the last integer, fraction number, or content wrapped in *boxed*{...} in the model's output and compare it with the ground truth.

For coding datasets, we also use no system prompt. We combine the raw question, an input-output example, and coding instructions in the user prompt. Due to resource limitations, we consider the single-turn coding task, where the model can't run its code in advance. During evaluation, we extract the code block in the model's answer and use the sandbox from VolcEngine[8] to test the code. We use 10 cases per question, set a time limit of 5 seconds, and only give a score of 1 when the model passes all test cases.

Table 5 shows the statistics of all datasets used. We show an example of each domain below.

Table 5: Statistics of datasets.

Dataset	GSM8k	MATH-500	DAPO	AIME	CodeContests	LiveCodeBench	DeepCoder	OpenThoughts
License	MIT	MIT	Apache-2.0	Apache-2.0	CC BY 4.0	CC	MIT	Apache-2.0
# Train	7473	-	13,116	-	-	-	13.485	113,957
# Eval	1319	500	1000	933	400	611	500	-

An Example from DAPO-Math

The vertices of a regular nonagon (9-sided polygon) are to be labeled with the digits 1 through 9 in such a way that the sum of the numbers on every three consecutive vertices is a multiple of 3. Two acceptable arrangements are considered to be indistinguishable if one can be obtained from the other by rotating the nonagon in the plane. Find the number of distinguishable acceptable arrangements.

An Example from DeepCoder

Given a permutation $P_1, \ldots, P_N$ of $1, \ldots, N$, find the number of integers i $(1 \leq i \leq N)$ that satisfy the following condition: for every integer j $(1 \leq j \leq i)$, $P_i \leq P_j$.

Constraints
- $1 \leq N \leq 2 \times 10^5$
- $P_1, \ldots, P_N$ is a permutation of $1, \ldots, N$.
- All values in the input are integers.

Input
The input is given from standard input in the following format:
N
P1 P2 ... PN

Output
Print the number of integers i that satisfy the condition.

Sample Input
5
4 2 5 1 3

Sample Output
3

Read the input from standard input (stdin) and write the answer to standard output (stdout). First reason step by step about your approach, then put your final solution in a single python code block. Your code should finish in 5s under the data size specified in the question.

[8] https://hub.docker.com/r/volcengine/sandbox-fusion

Table 6: Training configurations.

(a) Distill

Hyperparameter	Value
Actor learning rate	$1e-5$
Batch Size	384
Training step	400
Max response length	4096

(b) OPD

Hyperparameter	Value
Learning rate	$1e-6$
Batch Size	160
Training step	1000
Max response length	4096
Rollout temp (τ)	1.0
Advantage clip (ϵ)	0.2
Log Prob Clamp	-10
KL Estimator	k1

(c) GRPO & Cliff

Hyperparameter	Value
Actor learning rate	$1e-6$
Batch Size	64
Num of rollouts	12
Training step	200
Max response length	4096
Rollout temp (τ)	1.0
KL penalty (β)	0
Advantage clip (ϵ)	0.2

B Training Details

We develop our code based on veRL (Sheng et al., 2025). We provide key hyperparameters of all training algorithms including distill with teacher rollouts, on-policy distillation, and GRPO in Table 6. For the warm-up SFT for Qwen3, we use the same setting as Distill, but train 2 epoch on the Openthought dataset. We follow standard practices in veRL and ensure different settings have comparable data budgets. Models are trained on 4 NVIDIA H100 80GB GPUs or B200 192GB GPUs[9].

For Cliff-specific parameters, we use $\lambda = 0$ and a teacher rollout temperature of 0.6. To save compute, we skip teacher judging for prompt groups that the automatic verifier scores unanimously — all rollouts correct or all incorrect. Such groups carry no reward variance and thus receive zero advantage under the GRPO-style loss. Skipping them only causes signal loss when the judge disagrees with the verifier, which is very rare on these groups where the question is too simple or too hard. During inference, we use greedy decoding and a maximum response length of 8192.

C Training-Time Overhead of Cliff

Cliff introduces additional teacher inference compared with standard GRPO, including teacher-generated reference solutions and judgments of student rollouts. In our current implementation, the average training time per step is approximately $1.8\times$ that of standard GRPO under the same experimental setting. However, we note that teacher inference is naturally amenable to asynchronous execution: teacher reference generation and rollout judgments can be pipelined with rollout generation and policy optimization, potentially reducing the wall-clock overhead without changing the learning objective. We leave the implementation and systematic evaluation of such asynchronous training to future work.

D Length Dynamics of Cliff

This section provides a theoretical analysis on length dynamics of Cliff. To summarize:

- GRPO encourages lengthening when and only when correct rollouts are longer than incorrect ones.
- Cliff is more "lenient" on lengthening than GRPO, and a larger λ makes lengthening easier.
- $\lambda = 0$ eliminates the possibility to hack the reward by lengthening alone. $\lambda < \frac{L_{cor}}{L_{max}}$ is a sufficient in most conditions as it ensures the advantage of an incorrect rollout never exceed a correct one.
- Hard capping $p(a) = 0$ for overlength rollouts (considering the whole rollout as wrong) is necessary to stable training.

[9]We find the vLLM inference engine may behave differently for Phi-4-mini on different GPUs, so Phi-4-mini experiments are conducted on H100s. Qwen-4-3B experiments are conducted on B200s.

D.1 NOTATIONS

We follow the notation in Section 3 and add the quantities of Table 7.

Table 7: Additional notations for the math analysis.

Symbol	Meaning
$L_{\max}$	The maximum response length
$T = \sum_{i=1}^{N} \lvert a_i \rvert$	The total number of response tokens in a group
$s_i = p(a_i)/\lvert a_i \rvert$	The fraction of the correct prefix in an incorrect rollout
N_{cor}, N_{inc}	The number of correct / incorrect rollouts in a group
L_{cor}, L_{inc}	The average of $\lvert a_i \rvert$ over the correct / incorrect rollouts
$\rho = L_{\text{cor}}/L_{\text{inc}}$	The ratio of correct and incorrect rollout lengths in a group

D.2 LENGTH MASS (Φ) AND LENGTH BIAS OF GRPO

The length of an autoregressive LLM policy's samples is controlled by the EOS probability at each position. Raising a non-EOS token at a position where EOS was a live alternative lowers the relative mass of EOS; raising the EOS token itself shortens. **In other words, each untruncated rollout a_i contributes $|a_i| - 1$ non-EOS tokens and exactly one EOS token**. Given that GRPO assigns the same advantage to every token in a rollout, a group's net lengthening pressure (*length mass*) can be defined as

$$\Phi \;=\; \sum_{i=1}^{N}\Big[\sum_{j=0}^{|a_i|-2} A_{ij} - A_{i,|a_i|-1}\Big] = = \sum_{i=1}^{N} A_i|a_i| \;-\; 2\sum_{i=1}^{N} A_i \;=\; \sum_{i=1}^{N} A_i|a_i|. \tag{7}$$

The final equality holds because the mean-centering feature in GRPO () gives $\sum_{i=1}^{N} A_i = 0$.

By decomposing A_{ij} to the correct tokens and incorrect tokens, we have:

$$\begin{aligned}\Phi \;&=\; \sum_{i=1}^{N} A_i|a_i| \;=\; A_{\text{cor}} \sum_{i|R(q,a_i)=1} |a_i| \;+\; A_{\text{inc}} \sum_{i|R(q,a_i)=0} |a_i| \;=\; N\Big[\mu A_{\text{cor}} L_{\text{cor}} + (1-\mu) A_{\text{inc}} L_{\text{inc}}\Big] \\ &= N\Big[\mu \frac{1-\mu}{\sigma} L_{\text{cor}} - (1-\mu)\frac{\mu}{\sigma} L_{\text{inc}}\Big] \;=\; \frac{N\mu(1-\mu)}{\sigma}\left(L_{\text{cor}} - L_{\text{inc}}\right) \;=\; N\,\sigma\left(L_{\text{cor}} - L_{\text{inc}}\right).\end{aligned} \tag{8}$$

As a conclusion, GRPO's length mass Φ has the same sign of $(L_{\text{cor}} - L_{\text{inc}})$, or $\Phi > 0 \iff \rho > 1$. Therefore, **GRPO is a faithful amplifier of whatever correctness-length correlation the policy's own samples happen to carry.** There is therefore no length channel to exploit – lengthening cannot increase the objective unless longer responses are genuinely more often correct – so reward hacking on length does not occur.

D.3 LENGTH MASS OF CLIFF

Since Cliff recenters token-level advantage to zero, we have $\sum_i \sum_j A_{ij} = 0$. Therefore, the length mass of Cliff can be written as:

$$\Phi \;=\; \sum_{i=1}^{N}\Big[\sum_{j=0}^{|a_i|-2} A_{ij} - A_{i,|a_i|-1}\Big] = \sum_{i=1}^{N}\Big[\sum_{j=0}^{|a_i|-1} A_{ij} - 2A_{i,|a_i|-1}\Big] = -2\sum_{i=1}^{N} A_{i,|a_i|-1} = 2Nb. \tag{9}$$

The final equality holds because the advantage of the last token in a rollout is always $A_{\text{cor}} - b$ for correct rollouts and $A_{\text{inc}} - b$ for incorrect ones, since $p(a) \le |a_i| - 1$. Given that the outcome-level GRPO advantages are normalized within each group, $N_{\text{cor}} A_{\text{cor}} + N_{\text{inc}} A_{\text{inc}} = 0$.

Using Equation (5), we can further derive:

$$\begin{aligned}\Phi = 2Nb &= \frac{2N}{T}\Big[A_{cor}(N_{cor}L_{cor} + N_{inc}L_{inc}\lambda\bar{s}) + A_{inc}N_{inc}L_{inc}(1-\bar{s})\Big] \\ &= \frac{2N^2(1-\mu)L_{inc}}{\sigma T}\Big[\mu\rho + (1-\mu)\lambda\bar{s} - \mu(1-\bar{s})\Big]\end{aligned} \tag{10}$$

Therefore:

$$\Phi > 0 \Longleftrightarrow \mu\rho + (1-\mu)\lambda\bar{s} - \mu(1-\bar{s}) > 0 \Longleftrightarrow \rho > 1 - \bar{s}(1 + \lambda\frac{1-\mu}{\mu}). \tag{11}$$

From this formulation, we can find that:

• If $\bar{s} = 0$, then Cliff degenerates to GRPO, where the lengthening condition is $\rho > 1$.

• When $\bar{s} > 0$, we denote $f(\lambda) = 1 + \lambda\frac{1-\mu}{\mu}$, then we have $f(\lambda) > 0$ and $df(\lambda)/d\lambda > 0$. This means **Cliff encourages lengthening more than GRPO does (the lengthening condition is easier to achieve), and a larger λ makes the lengthening trend stronger.**

The importance of truncation. Previous analysis is built upon an assumption that the EOS token appears in each rollout. This assumption might not hold when some rollouts reach L_{max} and are truncated. In GRPO, overlength rollouts almost always receive a zero reward and negative advantage, suppressing the lengthing trend. In Cliff, however, the teacher may give a long valid prefix for reasoning that is not wrong but just did not finish. That's why we set a hard cap $p(a) = 0$ for overlength rollouts in Cliff to ensure overlength rollouts are penalized. This setting is thus not merely a saving of judge calls but a necessary guard.

D.4 Cliff's Robustness to Reward Hacking

We consider the possibility of length reward hacking when $b > 0$, or the overall trend of a group is lengthening. Length hacking is excluded if two conditions hold: **(C1)** a correct rollout always outranks an incorrect one in total advantage, and **(C2)** an incorrect rollout cannot raise its total advantage by lengthening its valid prefix.

For **(C1)**:

$$\begin{aligned}\text{(C1)} \quad &\Longleftrightarrow \quad L_{max}\left(\lambda A_{\text{cor}} - b\right) < L_{\text{cor}}\left(A_{\text{cor}} - b\right) \quad \Longleftrightarrow \quad L_{max} < L_{cor}\frac{A_{cor} - b}{\lambda A_{cor} - b} \\ &\Longleftarrow \quad L_{max} < L_{cor}\frac{A_{cor} - b}{\lambda(A_{cor} - b)} \quad \Longleftrightarrow \quad \lambda < \frac{L_{cor}}{L_{max}}\end{aligned} \tag{12}$$

Therefore, we obtain a sufficient condition for (C1). Given the empirical value $L_{cor} \approx 0.3L_{max}$, $\lambda < 0.3$ is safe for condition (C1).

For **(C2)**:

$$\text{(C2)} \quad \Longleftrightarrow \quad \lambda A_{\text{cor}} - b \leq 0 \quad \Longleftrightarrow \quad \lambda \leq \frac{b}{A_{\text{cor}}} \tag{13}$$

Given the empirical value $b \approx 0.05$, (C2) requires λ to be a very small value, which is why we use $\lambda = 0$ in the main experiments. It's also worth noting that (C2) isn't required for robust training, as extending the valid prefix is hacking only if the added tokens are padding or repetition; if they are genuine reasoning that carries the rollout further before it errs, rewarding them is exactly what Cliff is for. What separates the two cases is the judge's verdict rather than the shape of the advantage.

E PROMPTS

This section shows the prompts for the judge model. In Stage 1, the judge simply solves the problem by itself, so we don't explicitly set prompts, following the format in Section A. In Stage 2, the system prompt for identifying the student's mistake is shown below. We make the following special considerations when designing the prompt:

• If the student's reasoning process is stuck or repeating, assign the pitfall step at that position.

• Avoid over-penalizing for different but correct approaches, failed attempts corrected later, or minor mistakes that don't impact the answer.

The student's rollout is presented to the teacher as a sequence of sentences, which serve as the basic units for identifying the Pitfall Step. For plain text, a "sentence" is split by separators, including ".", ",", "?", new lines, etc. For structured formats such as LaTeX or code blocks, each line is treated as a sentence.

System Prompt for Cliff Stage 2, Math Domain

You are a math expert. You will be given a math problem, a REFERENCE solution written by an expert (with its final answer), and a student's solution split into numbered sentences, one per line, in the form [id] sentence. Compare the student's solution against the reference and decide whether it is correct; only if it is genuinely incorrect, identify the [id] of the FIRST mistake.

Rules for finding the mistake:
- DECISION RULE: if the student's FINAL ANSWER matches the reference's, the solution is almost always correct – mark it correct unless a step contains a concrete, undeniable error that a correct solution could not contain.
- Flag ONLY genuine, provable errors:
  - propagating calculation errors
  - misinterpretation of the problem
  - unjustified leaps a correct proof could not make
  - repetitive, gibberish, or incoherent degeneration
- NEVER flag the following:
  - a different-but-valid approach or notation
  - an algebraically equivalent expression or restatement (e.g. "k ¡ 22" vs "k ¡= 21" over integers)
  - a step that is merely unnecessary, verbose, or informal
  - a failed attempt corrected later
  - a minor slip that changes nothing later
- When unsure, treat the step as correct and give the student the benefit of the doubt.
- Report only the FIRST genuine mistake.

Answer format:
Your output should be formatted in three parts, each beginning with "**Part X:**" where X is 1, 2, or 3. The three parts are:
- Part 1: Briefly restate the reference's final answer.
- Part 2: First compare the student's final answer to the reference. If they match, search only for an undeniable answer-invalidating error and, finding none, mark the solution correct. Otherwise walk the sentences by [id] and stop at the FIRST sentence you are confident is genuinely wrong.
- Finally (Part 3), output a JSON object with the following fields:
  - "correctness": a boolean indicating whether the solution is correct.
  - "first_mistake_id": the integer [id] (shown in brackets before each sentence) of the sentence where the first mistake occurs, or null if the solution is correct.
  - "first_mistake_sentence": the text of that sentence copied from the list, or null if the solution is correct.
  - "explanation": a brief explanation of the first mistake and what it should be, or null if the solution is correct.

System Prompt for Cliff Stage 2, Coding Domain

You are an expert programmer. You will be given a programming problem, a REFERENCE solution written by an expert (a correct approach with its code), and a student's solution split into numbered lines, one per line, in the form [id] line. Compare the student's solution against the reference and decide whether it is correct; only if it is genuinely incorrect, identify the [id] of the FIRST mistake.

Rules for finding the mistake:
- DECISION RULE: if the student's code implements a correct approach that would produce the right outputs, the solution is correct – mark it correct unless a line contains a concrete, undeniable bug that a correct solution could not contain.
- Flag ONLY genuine, provable errors:
  - an incorrect algorithm or logic that produces wrong output
  - off-by-one, wrong-index, or boundary errors that change the result on common/typical inputs
  - code that crashes or loops forever (never terminates), regardless of how slow a terminating solution is
  - repetitive, gibberish, or incoherent degeneration
- NEVER flag the following:
  - a different-but-valid algorithm, data structure, or coding style than the reference
  - a line that is merely unnecessary, verbose, or informal (if not excessive repetitive)
  - ANY failed attempt that the student later corrects.
  - a bug that changes the output ONLY on certain boundary values or a rare input while the solution is correct on all other cases.
  - a solution that is correct but its time/space complexity might exceed the limits implied by the statement's constraints.
- When unsure, treat the line as correct and give the student the benefit of the doubt.
- Report only the FIRST genuine mistake.

Answer format:
Your output should be formatted in three parts, each beginning with "**Part X:**" where X is 1, 2, or 3. The three parts are:
- Part 1: Briefly restate the reference's code design.
- Part 2: First identify the student's final solution, and check whether that final solution implements a correct approach that would produce the right outputs. If so, search only for an undeniable output-invalidating bug and, finding none, mark the solution correct. Otherwise walk the lines by [id] and stop at the FIRST line in the REASONING or CODE BLOCK that you believe is genuinely incorrect. A buggy code line is often the faithful implementation of a decision the student already stated in the reasoning above it; when that is the case, the earlier reasoning line is the mistake, not the code line. Before settling on a code line, check whether the reasoning above it already states the same flawed decision – if it does, report that reasoning line instead.
- Part 3: Finally, output a JSON object with the following fields:
  - "correctness": a boolean indicating whether the solution is correct.
  - "first_mistake_id": the integer [id] (shown in brackets before each sentence) of the sentence where the first mistake occurs, or null if the solution is correct.
  - "first_mistake_sentence": the text of that sentence copied from the list, or null if the solution is correct.
  - "explanation": a brief explanation of the first mistake and what it should be, or null if the solution is correct.

An Example of User Prompt for Cliff Stage 2

Problem:
.....
Reference Solution (by an expert):
.....
Student's Solution (numbered sentences):
[0] .....
[1] .....
.....

F CASE STUDIES

F.1 CASE 1: THE TEACHER IDENTIFIES THE STUDENT'S MISTAKE

Problem:
How many positive integers less than or equal to 100 have a prime factor that is greater than 4?

Student's Solution:
[0] To determine how many positive integers less than or equal to 100 have a prime factor greater than 4, we use the principle of inclusion-exclusion.
[1] 1.
[2] **Total numbers**: There are 100 positive integers from 1 to 100.
[3] 2.
[4] **Numbers with prime factors ≤ 4**: These are numbers composed of the primes 2, 3, and 5.
[5] We calculate the count of such numbers using inclusion-exclusion:
[6] - **Multiples of 2**: 50
[7] - **Multiples of 3**: 33
[8] - **Multiples of 5**: 20
[9] - **Multiples of 6 (2×3)**: 16
[10] - **Multiples of 10 (2×5)**: 10
[11] - **Multiples of 15 (3×5)**: 6
[12] - **Multiples of 30 ($2\times3\times5$)**: 3
[13] Applying inclusion-exclusion:
[14] $50 + 33 + 20 - 16 - 10 - 6 + 3 = 74$
[15] 3.
[16] **Numbers with prime factors > 4**: Subtract the count of numbers with prime factors ≤ 4 from the total:
[17] $100 - 74 = 26$
[18] **Answer**: $\boxed{26}$

Teacher's Judgment:
Part 1: The reference answer is 80.

Part 2: The student's final answer is 26, which does not match the reference answer of 80. Let me trace through the student's solution to find the first mistake.

The student's approach in [4] is already wrong. The problem asks for numbers with a prime factor *greater than 4*. The primes greater than 4 are 5, 7, 11, 13, ... The complementary set (numbers with NO prime factor greater than 4) would be numbers whose only prime factors are 2 and 3 (the primes ≤ 4). However, the student states in [4] that "Numbers with prime factors ≤ 4: These are numbers composed of the primes 2, 3, and 5." Including 5 is incorrect because $5 > 4$, so 5 is NOT a prime ≤ 4.

Furthermore, even if we accept the student's interpretation, their approach using inclusion-exclusion to count multiples of 2, 3, and 5 is fundamentally flawed because being a multiple of 2 doesn't mean your only prime factors are ≤ 4 (e.g., 10 is a multiple of 2 but also has 5 as a factor). The inclusion-exclusion as applied counts numbers divisible by at least one of $\{2, 3, 5\}$, not numbers whose prime factors are all in $\{2, 3, 5\}$.

The first clear, undeniable error is in sentence [4], where the student incorrectly includes 5 as a prime ≤ 4.

Part 3:
```json {
"correctness": false,
"first_mistake_id": 4,
"first_mistake_sentence": "**Numbers with prime factors $\leq 4$**: These are numbers composed of the primes 2, 3, and 5.",
"explanation": "The student incorrectly includes 5 as a prime $\leq 4$. The primes less than or equal to 4 are only 2 and 3. The prime 5 is greater than 4, so it should not be included in this set."
} ```
```

F.2 Case 2: The Teacher Catches an Incorrect Solution that Passes the Automatic Verifier

Problem:
How many three-digit positive integers have an odd number of even digits? Provide your answer as an integer.
Student's Solution:
[0] To determine how many three-digit positive integers have an odd number of even digits, we analyze the possible combinations of even and odd digits in each place (hundreds, tens, units).
[1] A three-digit number has an odd number of even digits if it has 1 or 3 even digits.
[2] **Case 1: Exactly 1 even digit**
[3] - **Hundreds place even, tens and units odd**:
[4] $5 \times 5 \times 5 = 125$
[5] - **Tens place even, hundreds and units odd**:
[6] $4 \times 5 \times 5 = 100$
[7] - **Units place even, hundreds and tens odd**:
[8] $4 \times 5 \times 5 = 100$
[9] **Total for Case 1**: $125 + 100 + 100 = 325$
[10] **Case 2: Exactly 3 even digits**
[11] - All three digits even:
[12] $5 \times 5 \times 5 = 125$
[13] **Total for Case 2**: 125
[14] **Total numbers with odd even digits**:
[15] $325 + 125 = 450$
[16] **Verification via Complementary Counting**
[17] Total three-digit numbers: 900
[18] Numbers with 0 or 2 even digits:
[19] - **0 even digits**: $5 \times 5 \times 5 = 125$
[20] - **2 even digits**:
[21] - Hundreds and tens even, units odd: $5 \times 5 \times 5 = 125$
[22] - Hundreds and units even, tens odd: $5 \times 5 \times 5 = 125$
[23] - Tens and units even, hundreds odd: $5 \times 5 \times 5 = 125$
[24] **Total for 0 or 2 even digits**: $125 + 125 + 125 + 125 = 500$
[25] **Numbers with odd even digits**: $900 - 500 = 400$
[26] **Final Answer**:
[27] $\boxed{450}$

Teacher's Judgment:

Part 1: The reference solution's final answer is 450.

Part 2: The student's final answer is 450, which matches the reference. Let me check for any undeniable errors.

Looking at the student's work:

[4] For "Hundreds place even, tens and units odd": The student writes 5 × 5 × 5 = 125. But even digits for the hundreds place are {2, 4, 6, 8} = 4 choices, not 5. This is incorrect.

[6] For "Tens place even, hundreds and units odd": The student writes 4 × 5 × 5 = 100. But odd digits for the hundreds place are {1, 3, 5, 7, 9} = 5 choices, and even digits for tens are {0, 2, 4, 6, 8} = 5 choices, and odd digits for units are {1, 3, 5, 7, 9} = 5 choices. This should be 5 × 5 × 5 = 125, not 100.

So sentence [4] is the first genuine mistake - the hundreds place has only 4 even digits (2,4,6,8), not 5, so this should be 4 × 5 × 5 = 100, not 125.

Then in [6], the student has the hundreds place odd (should be 5 choices) × tens even (5 choices) × units odd (5 choices) = 5 × 5 × 5 = 125, but writes 4 × 5 × 5 = 100.

The student appears to have swapped the counts for sub-cases 1a and 1b/1c. The errors cancel out in the total (giving 325 instead of the correct 350), and then in Case 2, the student writes 5 × 5 × 5 =

125 for all three digits even, but this should be 4 × 5 × 5 = 100 (since hundreds place even has only 4 options). This error also compensates, giving a final answer of 450 which happens to be correct.

Despite the correct final answer, sentence [4] contains a concrete, undeniable error: the hundreds digit being even gives 4 choices (2,4,6,8), not 5, so the calculation should be 4 × 5 × 5 = 100, not 5 × 5 × 5 = 125.

However, per the rules: ”if the student’s FINAL ANSWER matches the reference’s, the solution is almost always correct – mark it correct unless a step contains a concrete, undeniable error that a correct solution could not contain.”

Sentence [4] states that when the hundreds place is even and tens/units are odd, the count is 5 × 5 × 5 = 125. This is undeniably wrong since even hundreds digits are {2,4,6,8} = 4 choices. A correct solution could not contain this step.

Part 3:
```json {
”correctness”: false,
”first_mistake_id”: 4,
”first_mistake_sentence”: ”$5 \times 5 \times 5 = 125$”,
”explanation”: ”When the hundreds place is even, there are only 4 choices (2, 4, 6, 8), not 5. The correct calculation should be 4 × 5 × 5 = 100.”
} ```

### F.3 Case 3: A False Case Where the Teacher’s Pitfall Step is Inconsistent with Human Experts

**Problem:**
One writes 268 numbers around a circle, such that the sum of 20 consecutive numbers is always equal to 75. The numbers 3, 4, and 9 are written in positions 17, 83, and 144, respectively. Find the number in position 210.
**Student’s Solution:**
[0] Given 268 numbers arranged around a circle such that the sum of any 20 consecutive numbers is always 75, we need to find the number in position 210.
[1] The numbers 3, 4, and 9 are located at positions 17, 83, and 144, respectively.
[2] 1.
[3] **Periodicity Analysis**:
[4] - The sum of any 20 consecutive numbers is 75, implying the sequence is periodic with period 20.
[5] - Therefore, the number in position $n$ is the same as the number in position $n \mod 20$.
[6] 2.
[7] **Position 210**:
[8] - Calculate $210 \mod 20$:
[9] $210 \div 20 = 10$ remainder $10 \implies 210 \equiv 10 \mod 20$
[10] - Thus, the number in position 210 is the same as the number in position 10.
[11] 3.
[12] **Using Given Positions**:
[13] - The numbers 3, 4, and 9 are at positions 17, 83, and 144, respectively.
[14] - Convert these positions to their equivalent positions modulo 20:
[15] $17 \mod 20 = 17, \quad 83 \mod 20 = 3, \quad 144 \mod 20 = 4$
[16] - Therefore, the numbers 3, 4, and 9 are at positions 17, 3, and 4 modulo 20.
[17] 4.
[18] **Sum of Consecutive Blocks**:
[19] - The sum of any 20 consecutive numbers is 75.
[20] For example, the sum of positions 17 to 36 (20 numbers) is 75.
[21] - The sum of positions 17 to 36 includes the numbers 3, 4, and 9.
[22] The remaining 17 numbers must sum to $75 - (3 + 4 + 9) = 69$.
[23] 5.
[24] **Conclusion**:
[25] - Since the sequence is periodic with period 20, the number in position 10 (and thus position 210) must be the same as the number in position 3 modulo 20, which is 4.
[26] Thus, the number in position 210 is $\boxed{4}$.
```

Teacher's Judgment:

Part 1: The reference solution's final answer is -1.

Part 2: The student's final answer is 4, which does not match the reference answer of -1. Let me trace through the student's solution to find the first genuine mistake.

Sentences [0]-[4] are correct: the sum condition does imply periodicity with period 20 (i.e., $a_{i+20} = a_i$ for all i in a linear sequence).

Sentence [5]: "Therefore, the number in position n is the same as the number in position $n \mod 20$." This is where the first genuine error occurs. Since the numbers are arranged in a circle of 268 positions, the periodicity $a_{i+20} = a_i$ combined with the circular condition $a_i = a_{i+268}$ means the actual period divides $\gcd(20, 268) = 4$. The sequence has period 4, not period 20. The student incorrectly assumes that period 20 works directly without checking compatibility with the circular arrangement of 268 numbers (since 20 does not divide 268).

Part 3:
```json {
"correctness": false,
"first_mistake_id": 5,
"first_mistake_sentence": "Therefore, the number in position $n$ is the same as the number in position $n \mod 20$.",
"explanation": "Since the 268 numbers are arranged in a circle, we have both a_{i+20} = a_i and a_{i+268} = a_i. This means the true period divides gcd(20, 268) = 4, so the sequence has period 4, not period 20. The student incorrectly assumed period 20 works on a circle of 268 numbers."
}```

**Explanation by the Human Annotator:**

**The teacher identifies sentence [5] as the pitfall step, but the claim "20 is a period for the ring" is completely correct.** While the claim in [5] is insufficient to solve the problem (to solve the problem, one must find "4 is a period"), it shouldn't be considered a reasoning mistake.

The human expert assigns the pitfall step to **sentence [25]**; that's where the actual mistake happens.
```