

## A Related Work

**Reinforcement Learning with Discrete Rewards.** Group Relative Policy Optimization (GRPO) [13] utilizes discrete rewards generated by a rule-based reward function to guide the policy model update. This reward function, known for its simplicity and unbiasedness, effectively mitigates reward hacking and has demonstrated strong performance. However, GRPO faces challenges related to slow training speed and unstable gradients during training. To address these issues, various methods have been proposed. DAPO [34] introduced a dynamic sampling strategy to improve gradient effectiveness by dynamically filtering invalid samples, thereby increasing sample efficiency, although this reduced training speed. CPPO [44] prunes completions with low absolute advantages, significantly reducing the number of gradient calculations and updates required, which enhances training efficiency but can lead to gradient estimation errors. GPG [45] directly optimizes the original reinforcement learning objective, eliminating the need for a proxy loss function and improving training efficiency. However, this simplification may result in a significant divergence between the actor and policy models. Dr.GRPO [35] improves token efficiency while maintaining inference performance. Despite these efforts, a critical challenge remains: these algorithms largely neglect the inherent difficulties introduced by discrete rewards during the optimization process. The oscillations caused by gradient vanishing and exploding are major contributors to the slow optimization speed. Our work specifically aims to overcome the challenges in gradient optimization that arise from using discrete rewards.

**Addressing Reward Design Challenges in LLM Reinforcement Learning.** Designing effective reward functions for identifying optimal strategies is a well-established area of research outside the context of Large Language Models (LLMs) [46, 47, 48]. However, a consensus on the optimal approach for LLM reinforcement learning has not yet been reached [49]. The RLHF framework proposed training a reward model to score LLM outputs [4]. A recurring challenge, as noted by numerous studies, is that low reward model accuracy can induce reward hacking [30, 31, 32]. Conversely, improving accuracy often reduces reward variance, which can slow down policy model convergence due to vanishing gradients [25]. Although the reward function presented in GRPO provides perfectly correct rewards, thereby avoiding reward hacking, it exacerbates gradient instability and hinders optimization speed [13, 35]. Recent theoretical findings indicate that a successful reward function requires a trade-off between variance and inaccuracy [37]. Motivated by this, our work seeks to design a reward function that effectively addresses the problem of reward hacking while simultaneously facilitating efficient optimization.

## B Theorems and proofs

### B.1 Proof of Proposition 1

The proof of Proposition 1 is expressed as follows:

*Proof.* By the policy gradient theorem, the gradient of the original objective (1) expands to:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{q \sim p_Q} \mathbb{E}_{o \sim \pi_{\theta}(\cdot|q)} [R(q, o) \nabla_{\theta} \log \pi_{\theta}(o|q)]. \quad (6)$$

For the noise-injected objective, its gradient becomes:

$$\nabla_{\theta} \tilde{J}(\pi_{\theta}) = \mathbb{E}_{q \sim p_Q} \mathbb{E}_{o \sim \pi_{\theta}(\cdot|q)} [\tilde{R}(q, o) \nabla_{\theta} \log \pi_{\theta}(o|q)]. \quad (7)$$

Substituting  $\tilde{R}(q, o) = R(q, o) + \epsilon$  and leveraging linearity of expectation:

$$\mathbb{E} [\nabla_{\theta} \tilde{J}] = \mathbb{E}_{q,o,\epsilon} [(R(q, o) + \epsilon) \nabla_{\theta} \log \pi_{\theta}(o|q)] \quad (8)$$

$$= \underbrace{\mathbb{E}_{q,o} [R(q, o) \nabla_{\theta} \log \pi_{\theta}(o|q)]}_{\mathbb{E}[\nabla_{\theta} J]} + \mathbb{E}_{\epsilon}[\epsilon] \cdot \mathbb{E}_{q,o} [\nabla_{\theta} \log \pi_{\theta}(o|q)]. \quad (9)$$

Zero-mean noise:  $\mathbb{E}_{\epsilon}[\epsilon] = 0$  by definition of  $\mathcal{N}(0, \sigma^2)$ . Thus, the cross-term vanishes:

$$\mathbb{E}[\nabla_{\theta} \tilde{J}] = \mathbb{E}[\nabla_{\theta} J] + 0 = \mathbb{E}[\nabla_{\theta} J]. \quad (10)$$

□

## 946 B.2 Proof of Proposition 2

947 *Proof.* Consider the perturbed objective function with noise-augmented reward  $R(q, o) + \epsilon$ . The  
 948 estimated value of the gradient of the noise enhancement objective function using  $n$  samples is:

$$\nabla \hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^n [\nabla \log \pi_{\theta}(o_i | q_i) \cdot (R(q_i, o_i) + \epsilon_i)], \quad (11)$$

949 where  $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$  is the Gaussian noise. The original reward gradient is:

$$\nabla \hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^n [\nabla \log \pi_{\theta}(o_i | q_i) \cdot (R(q_i, o_i))]. \quad (12)$$

950 Under this condition, the Eq. (11) simplifies to:

$$\nabla \hat{J}(\theta) = \underbrace{\nabla \hat{J}(\theta)}_{\text{origin gradient}} + \underbrace{\frac{1}{n} \sum_{i=1}^n [\nabla \log \pi_{\theta}(o_i | q_i) \cdot \epsilon_i]}_{\text{noise gradient}}. \quad (13)$$

951 While the expectation  $\mathbb{E}_{\epsilon}[\epsilon] = 0$  implies the noise contribution's mean is zero, the variance of the  
 952 gradient term persists. To compute this variance, we use the definition:  $\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$ .  
 953 Applying this to the noise-induced component  $\epsilon \cdot \nabla_{\theta} \log \pi_{\theta}(o|q)$ , we get:

$$\text{Var}(\epsilon \cdot \nabla_{\theta} \log \pi_{\theta}(o|q)) = \mathbb{E}[\epsilon^2 \cdot \|\nabla_{\theta} \log \pi_{\theta}(o|q)\|^2] - (\mathbb{E}[\epsilon \cdot \nabla_{\theta} \log \pi_{\theta}(o|q)])^2. \quad (14)$$

954 Since  $\mathbb{E}[\epsilon] = 0$ , the second term vanishes. For the first term, note that:

$$\mathbb{E}[\epsilon^2] = \text{Var}(\epsilon) + (\mathbb{E}[\epsilon])^2 = \sigma^2 + 0 = \sigma^2. \quad (15)$$

955 This allows us to simplify the variance expression to:

$$\text{Var}(\text{noise gradient}) = \text{Var}(\epsilon \cdot \nabla_{\theta} \log \pi_{\theta}) = \sigma^2 \cdot \mathbb{E}[\|\nabla_{\theta} \log \pi_{\theta}(o|q)\|^2] > 0, \quad (16)$$

956 provided  $\nabla_{\theta} \log \pi_{\theta}$  is not identically zero (a reasonable assumption for non-degenerate policies).

957  $\square$

## 958 B.3 Proof of Proposition 3

959 From Definition 1 in [37], The accuracy of the reward function is as follows:

960 **Definition 1.** Given a prompt  $x \in \mathcal{X}$ , the accuracy of a reward model  $r_{RM} : \mathcal{X} \times \mathcal{Y} \rightarrow [-1, 1]$  with  
 961 respect to a distribution  $\mathcal{D}$  over unordered output pairs is defined by:

$$\text{acc}_{x, \mathcal{D}}(r_{RM}) := \mathbb{E}_{\{y, y'\} \sim \mathcal{D}} [\mathbb{1}[\text{sign}(r_{RM}(x, y) - r_{RM}(x, y')) = \text{sign}(r_G(x, y) - r_G(x, y'))]], \quad (17)$$

962 where  $r_G$  is the ground truth reward,  $\mathbb{1}[\cdot]$  is an indicator function, and  $\text{sign} : \mathbb{R} \rightarrow \{-1, 0, 1\}$  is the  
 963 sign function.<sup>1</sup>

964 From Definition 2 in [37], The variance of the reward function is as follows:

965 **Definition 2.** Given a policy  $\pi_{\theta}$ , prompt  $x \in \mathcal{X}$ , and reward model  $r_{RM} : \mathcal{X} \times \mathcal{Y} \rightarrow [-1, 1]$ , the  
 966 reward variance induced by  $r_{RM}$  for  $\pi_{\theta}$  and  $x$  is defined by:

$$\text{Var}_{y \sim \pi_{\theta}(\cdot | x)}[r_{RM}(x, y)] := \mathbb{E}_{y \sim \pi_{\theta}(\cdot | x)} \left[ (r_{RM}(x, y) - \mathbb{E}_{y' \sim \pi_{\theta}(\cdot | x)}[r_{RM}(x, y')])^2 \right]. \quad (18)$$

967 The proof of Proposition 3 is expressed as follows:

968 *Proof.* Suppose the original reward model is  $r_{RM}(\mathbf{x}, \mathbf{y})$ . We add a zero-mean Gaussian random  
 969 variable  $\epsilon$  with variance  $\sigma^2$  to it, resulting in the new reward signal  $\tilde{r}_{RM}(\mathbf{x}, \mathbf{y})$ :

$$\tilde{r}_{RM}(\mathbf{x}, \mathbf{y}) = r_{RM}(\mathbf{x}, \mathbf{y}) + \epsilon, \quad (19)$$

970 where  $\epsilon \sim \mathcal{N}(0, \sigma^2)$ , and we assume that  $\epsilon$  is independent of  $\mathbf{y}$  (conditioned on  $\mathbf{x}$ ).

<sup>1</sup>For a set of prompts, accuracy refers to the mean accuracy over the set.

971 **1. Change in Reward Variance (Based on Definition 2).** Definition 2 defines the reward variance  
 972 induced by the reward model  $r_{RM}$  under policy  $\pi_\theta$  and prompt  $\mathbf{x}$  as:

$$\text{Var}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}[r_{RM}(\mathbf{x}, \mathbf{y})] := \mathbb{E}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})} \left[ \left( r_{RM}(\mathbf{x}, \mathbf{y}) - \mathbb{E}_{\mathbf{y}' \sim \pi_\theta(\cdot|\mathbf{x})}[r_{RM}(\mathbf{x}, \mathbf{y}')] \right)^2 \right]. \quad (20)$$

973 This is the standard definition of variance: the expected squared deviation from the mean.

974 Now, we compute the variance of the perturbed reward signal  $\tilde{r}_{RM}$ :

$$\text{Var}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}[\tilde{r}_{RM}(\mathbf{x}, \mathbf{y})] = \text{Var}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}[r_{RM}(\mathbf{x}, \mathbf{y}) + \epsilon]. \quad (21)$$

975 Using the property of variance for two random variables  $A$  and  $B$ :

$$\text{Var}(A + B) = \text{Var}(A) + \text{Var}(B) + 2\text{Cov}(A, B). \quad (22)$$

976 Here,  $A = r_{RM}(\mathbf{x}, \mathbf{y})$  (a random variable depending on  $\mathbf{y}$ ), and  $B = \epsilon$  (independent of  $\mathbf{y}$ ). Since  $\epsilon$   
 977 is independent of  $\mathbf{y}$ ,  $A$  and  $B$  are independent, so  $\text{Cov}(A, B) = 0$ . Therefore:

$$\text{Var}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}[r_{RM}(\mathbf{x}, \mathbf{y}) + \epsilon] = \text{Var}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}[r_{RM}(\mathbf{x}, \mathbf{y})] + \text{Var}(\epsilon). \quad (23)$$

978 Since  $\text{Var}(\epsilon) = \sigma^2$ , the perturbed reward variance becomes:

$$\text{Var}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}[\tilde{r}_{RM}(\mathbf{x}, \mathbf{y})] = \text{Var}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}[r_{RM}(\mathbf{x}, \mathbf{y})] + \sigma^2. \quad (24)$$

979 **Conclusion:** If  $\sigma^2 > 0$ , adding zero-mean Gaussian noise to the reward model increases the reward  
 980 variance by exactly  $\sigma^2$ . This aligns with our intuitive understanding based on Theorem 1.

981 **2. Change in Reward Model Accuracy (Based on Definition 1).** Definition 1 defines the accuracy  
 982 of a reward model  $r_{RM}$  at a given prompt  $\mathbf{x}$  and distribution  $\mathcal{D}$  over unordered output pairs as:

$$\text{acc}_{\mathbf{x}, \mathcal{D}}(r_{RM}) := \mathbb{E}_{(\mathbf{y}, \mathbf{y}') \sim \mathcal{D}} [\mathbb{1}[\text{sign}(r_{RM}(\mathbf{x}, \mathbf{y}) - r_{RM}(\mathbf{x}, \mathbf{y}')) = \text{sign}(r_G(\mathbf{x}, \mathbf{y}) - r_G(\mathbf{x}, \mathbf{y}'))]], \quad (25)$$

983 where  $r_G$  is the ground truth reward function. This measures the probability that the reward model  
 984 correctly ranks a pair  $(\mathbf{y}, \mathbf{y}')$  relative to the ground truth.

985 Now consider the accuracy of the perturbed reward model  $\tilde{r}_{RM}$ :

$$\text{acc}_{\mathbf{x}, \mathcal{D}}(\tilde{r}_{RM}) = \mathbb{E}_{(\mathbf{y}, \mathbf{y}') \sim \mathcal{D}} [\mathbb{1}[\text{sign}(\tilde{r}_{RM}(\mathbf{x}, \mathbf{y}) - \tilde{r}_{RM}(\mathbf{x}, \mathbf{y}')) = \text{sign}(r_G(\mathbf{x}, \mathbf{y}) - r_G(\mathbf{x}, \mathbf{y}'))]]. \quad (26)$$

986 We analyze the difference:

$$\tilde{r}_{RM}(\mathbf{x}, \mathbf{y}) - \tilde{r}_{RM}(\mathbf{x}, \mathbf{y}') = (r_{RM}(\mathbf{x}, \mathbf{y}) + \epsilon_1) - (r_{RM}(\mathbf{x}, \mathbf{y}') + \epsilon_2), \quad (27)$$

987 where  $\epsilon_1, \epsilon_2 \sim \mathcal{N}(0, \sigma^2)$  are independently sampled. Let:

$$\Delta r_{RM} = r_{RM}(\mathbf{x}, \mathbf{y}) - r_{RM}(\mathbf{x}, \mathbf{y}'), \quad \Delta r_G = r_G(\mathbf{x}, \mathbf{y}) - r_G(\mathbf{x}, \mathbf{y}'), \quad (28)$$

988 and define  $\eta = \epsilon_1 - \epsilon_2 \sim \mathcal{N}(0, 2\sigma^2)$ . Then:

$$\tilde{r}_{RM}(\mathbf{x}, \mathbf{y}) - \tilde{r}_{RM}(\mathbf{x}, \mathbf{y}') = \Delta r_{RM} + \eta. \quad (29)$$

989 The condition for accuracy becomes:

$$\text{sign}(\Delta r_{RM} + \eta) = \text{sign}(\Delta r_G), \quad (30)$$

990 compared to the original condition:

$$\text{sign}(\Delta r_{RM}) = \text{sign}(\Delta r_G). \quad (31)$$

991 We now analyze how the addition of noise affects this condition:

992 - If the original model is accurate, i.e.,  $\text{sign}(\Delta r_{RM}) = \text{sign}(\Delta r_G)$ , then adding noise  $\eta$  may cause  
 993  $\text{sign}(\Delta r_{RM} + \eta) \neq \text{sign}(\Delta r_G)$ , especially when  $|\eta|$  is large enough to flip the sign of  $\Delta r_{RM}$ .

994 - If the original model is inaccurate, i.e.,  $\text{sign}(\Delta r_{RM}) \neq \text{sign}(\Delta r_G)$ , there is a small chance that  
 995  $\eta$  flips the sign back to match  $\text{sign}(\Delta r_G)$ , but this is not systematic and depends on the values of  
 996  $\Delta r_{RM}$  and  $\Delta r_G$ .

997 In general, since  $\eta$  is independent of both  $\Delta r_{RM}$  and  $\Delta r_G$ , it is more likely to disrupt the correct sign  
 998 relationship rather than correct it. Thus, adding random noise tends to make the ranking decisions  
 999 more random.

1000 **Conclusion:** Adding zero-mean Gaussian noise to the reward model makes its relative ranking of  
 1001 output pairs more random, thereby **reducing** the reward model's accuracy (acc).

1002 **3. Upper and lower bounds of training time).** According to Theorem 1, the lower bound on the  
 1003 training time  $t_\gamma$  is influenced by the variance of the reward function. Specifically, using Equation (24),  
 1004 we derive that the lower bound on  $t_\gamma$  satisfies:

$$\Omega \left( \left( \mathbb{E}_{q \sim p_Q} [\text{Var}_{o \sim \pi_\theta(\cdot|q)} R(q, o)] + \sigma^2 \right)^{-\frac{1}{3}} \right) \leq t_\gamma. \quad (32)$$

1005 According to Theorem 2, in some cases (particularly when a perfectly accurate model would lead to  
 1006 arbitrarily slow training), an inaccurate model could instead lead to faster increases in the true reward.  
 1007 We derive that the lower bound on  $t_\gamma$  satisfies:

$$t_\gamma \leq \mathcal{O}(\pi_{\theta(0)}(\mathbf{y}^\gamma | \mathbf{x})^{-1}) \quad (33)$$

1008 □

## 1009 C Training Dynamic

1010 In this section, we show more Training Dynamic information.

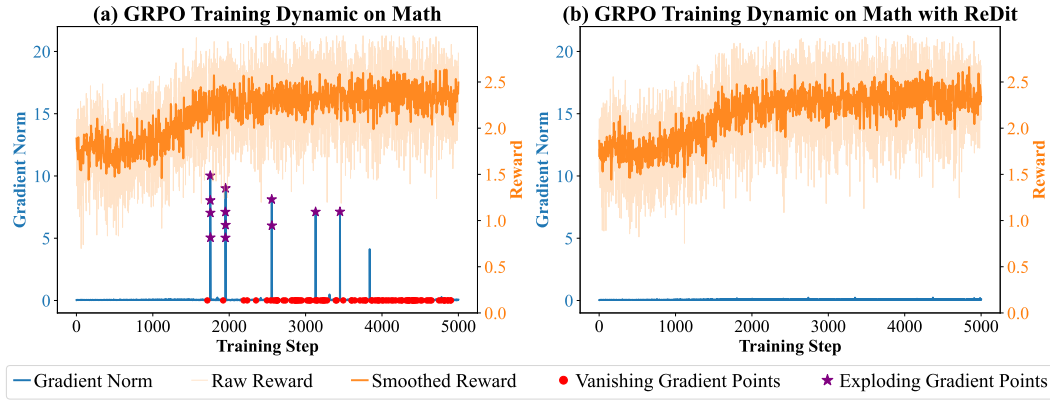


Figure 9: Training Dynamics of Gradient Norm and Reward on Math Dataset.

1011 Figure 9 shows the training dynamics of using and not using ReDit on the Math dataset, indicating  
 1012 that using ReDit can solve the problems of gradient oscillation and gradient vanishing, and improve  
 1013 training stability

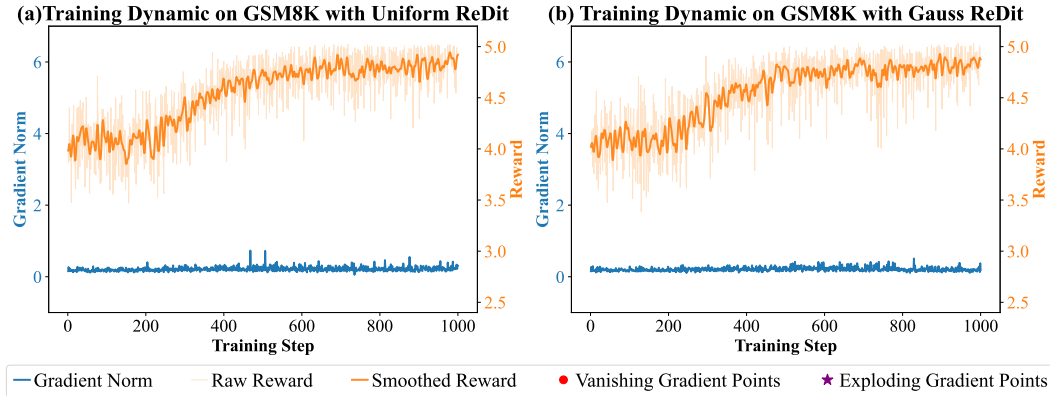


Figure 10: Training dynamics of gradient norm and reward on the GSM8K dataset, showing the impact of perturbations of different distributions.

1014 Fig 10 and Fig 11 Training dynamics using uniform and Gaussian perturbations. For both uniform  
 1015 and Gaussian perturbations, ReDit shows amazing gradient stability and training stability.

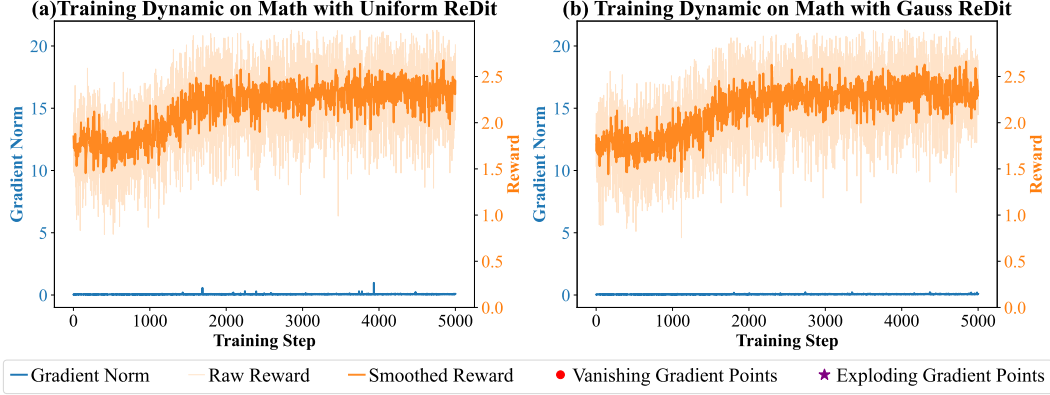


Figure 11: Training dynamics of gradient norm and reward on the Math dataset, showing the impact of perturbations of different distributions.

## D Experimental setting

### D.1 Dataset

In this section, we introduce the statistics of the dataset and the additional processing performed on the dataset. The statistics of the dataset are shown in Table 3.

Table 3: Number of samples in the train, validation, and test datasets for various datasets.

Number of samples	train dataset	validation dataset	test dataset
GSM8K	7473	-	1319
MATH	7506	-	5003
Geometry3K	2100	300	601

In addition, We added new templates to the original dataset to ensure the model could complete the required tasks and output formats. It is important to note that the added templates did not alter the original dataset, and special processing was performed for different LLMs. The specific examples are as follows:

#### Dataset Format of GSM8K

```
dataset: GSM8K
"prompt": [
  {"role": "system", "content": "Respond in the following format:
  <reasoning> ... </reasoning> <answer> ...</answer>"},
  {"role": "user", "content": "What is the largest single-digit prime number?"},
  {"role": "assistant", "content": "<reasoning> 9 is divisible by 3 and 8
  is divisible by 2, but 7 is prime. </reasoning>
  <answer>7</answer>"},
  {"role": "user", "content": "{question}}
],
"answer": {answer}
```

### Dataset Format of MATH

```
dataset: MATH
  "prompt": [
    {"role": "system", "content": "Respond in the following format:
    <reasoning> ... </reasoning> <answer> ...</answer>"},
    {"role": "user", "content": "{question}
    Let's think step by step and output the final answer within \\boxed{ }."
  },
  "answer": {answer}
```

1025

### Dataset Format of Geometry3K

```
dataset: Geometry3K
  "prompt": [
    {"role": "user", "content": [{
      "type": "image",
      "image": {image},
    },
    {
      "type": "text",
      "text": {question} + ".
      You FIRST think about the reasoning process as an internal monologue and
      then provide the final answer. The reasoning process MUST BE enclosed
      within <think> </think> tags. The final answer MUST BE put in \\boxed{ }."
    }
  ],
  "answer": {answer}
```

1026

## 1027 D.2 Reward function

1028 We design five reward functions for the GSM8K dataset and show how to implement ReDit:

### GSM8K Accuracy Reward Function

```
1 def correctness_reward_func_with_noise(prompts, completions, answer
2   , **kwargs) -> list[float]:
3   def extract_number(s: str) -> str:
4     match = re.search(r'\d+', s)
5     return match.group(0) if match else ''
6   responses = [completion[0]['content'] for completion in
7     completions]
8   q = prompts[0][-1]['content']
9   extracted_responses = [extract_xml_answer(r) for r in responses
10   ]
11   original_rewards = [2.0 if extract_number(r) == extract_number(
12     a) else 0.0 for r, a in zip(extracted_responses, answer)]
13
14   # ReDit add
15   noisy_rewards = [r + random.uniform(-m * 2.0, m * 2.0) for r in
16     original_rewards]
17   #noisy_rewards = [r + random.gauss(0, 2.0 * m / (3 ** 0.5)) for
18     r in original_rewards]
19   return noisy_rewards
```

1029

#### GSM8K Int Reward Function

```
1 def int_reward_func_with_noise(completions, **kwargs) -> list[float]:
2     responses = [completion[0]['content'] for completion in completions]
3     extracted_responses = [extract_xml_answer(r) for r in responses]
4     original_rewards = [0.5 if r.isdigit() else 0.0 for r in extracted_responses]
5
6     # ReDit add
7     noisy_rewards = [r + random.uniform(-m * 0.5, m * 0.5) for r in original_rewards]
8     #noisy_rewards = [r + random.gauss(0, 0.5 * m / (3 ** 0.5)) for r in original_rewards]
9     return noisy_rewards
```

1030

#### GSM8K Strict Format Reward Function

```
1 def strict_format_reward_func_with_noise(completions, **kwargs) -> list[float]:
2     pattern = r"^<reasoning>\n[\s\S]*?\n</reasoning>\n<answer>\n[\s\S]*?</answer>$"
3     completion_contents = [completion[0]["content"].strip() for completion in completions]
4     matches = [re.match(pattern, content, re.DOTALL | re.MULTILINE) for content in completion_contents]
5     original_rewards = [1.0 if match else 0.0 for match in matches]
6
7     # ReDit add
8     noisy_rewards = [r + random.uniform(-m * 1.0, m * 1.0) for r in original_rewards]
9     #noisy_rewards = [r + random.gauss(0, 1.0 * m / (3 ** 0.5)) for r in original_rewards]
10    return noisy_rewards
```

1031

#### GSM8K Sort Format Reward Function

```
1 def soft_format_reward_func_with_noise(completions, **kwargs) -> list[float]:
2     pattern = r"^<reasoning>[\s\S]*?</reasoning>[\s\S]*?<answer>[\s\S]*?</answer>$"
3     completion_contents = [completion[0]["content"].strip() for completion in completions]
4     matches = [re.match(pattern, content, re.DOTALL | re.MULTILINE) for content in completion_contents]
5     original_rewards = [1.0 if match else 0.0 for match in matches]
6
7     # ReDit add
8     noisy_rewards = [r + random.uniform(-m * 1.0, m * 1.0) for r in original_rewards]
9     #noisy_rewards = [r + random.gauss(0, 1.0 * m / (3 ** 0.5)) for r in original_rewards]
10    return noisy_rewards
```

1032

### GSM8K Reasoning Format Reward Function

```

1 def xmlcount_reward_func_with_noise(completions, **kwargs) -> list[
  float]:
2     def count_xml(text) -> float:
3         count = 0.0
4         if text.count("<reasoning>\n") == 1:
5             count += 0.125
6         if text.count("\n</reasoning>\n") == 1:
7             count += 0.125
8         if text.count("\n<answer>\n") == 1:
9             count += 0.125
10            #count -= len(text.split("\n</answer>\n")[-1])*0.001
11            if text.count("\n</answer>") == 1:
12                count += 0.125
13                count -= (len(text.split("\n</answer>")[-1]) - 1)*0.001
14            return count
15        contents = [completion[0]["content"] for completion in
16                    completions]
17        original_rewards = [count_xml(c) for c in contents]
18
19        # ReDit add
20        noisy_rewards = [r + random.uniform(-m * 0.5, m * 0.5) for r in
21                        original_rewards]
22        #noisy_rewards = [r + random.gauss(0, 0.5 * m / (3 ** 0.5))
23                        for r in original_rewards]
24        return noisy_rewards

```

1033

1034 As shown in the above code block, ReDit does not need to be modified in a complex way, only the  
 1035 reward function needs to be modified, and any method can be easily integrated. The reward functions  
 1036 of other datasets can be found in the code.

### 1037 D.3 Specific experimental parameters

1038 In this section, we present the experimental parameters, including LoRA parameters, GRPO and  
 1039 other baseline experimental parameters.

Table 4: LoRA Parameters

LoRA Target	LoRA Rank	LoRA Alpha	LoRA Dropout
q & v Proj	8	64	0.05

Table 5: GRPO Parameters

Learning Rate	Num Generations	Epochs
5e-6	4	10

Table 6: DAPO Parameters

Clip Ratio Low	Clip Ratio Low	Clip Ratio C	Num Generations Max
0.2	0.28	10.0	10

## 1040 E More result

1041 In this section, we present detailed numerical results for all experiments.



## E.1 Main Result

In this section, we show the results in Figure 6, the performance of GRPO and GRPO+ReDit on different datasets.

Table 7: Performance Comparison of Different Training Steps on the Math Dataset

Method \Step	0	1000	2000	3000	4000	5000	6000	7000	8000	9000
Instruct model	39	-	-	-	-	-	-	-	-	-
GRPO	-	47.86	49.46	47.18	47.28	47.26	47.57	47.63	47.89	48.01
Uniform ReDit	-	50.02	50.23	50.34	50.78	50.96	51.27	51.37	51.37	51.96
Gauss ReDit	-	49.78	50.73	51.03	51.07	51.53	51.43	52.01	52.01	52.55

Table 8: Performance Comparison of Different Training Steps on the GSM8K Dataset

Method \Step	0	1000	2000	3000	4000	5000	6000	7000	8000	9000
Instruct model	84.91	-	-	-	-	-	-	-	-	-
GRPO	-	85.70	86.01	86.47	86.73	87.13	87.78	88.52	88.73	89.07
Uniform ReDit	-	89.16	89.16	89.31	89.31	89.31	89.99	89.99	89.99	90.76
Gauss ReDit	-	89.02	89.37	89.61	89.54	89.54	89.54	89.61	89.61	90.46

Table 9: Performance Comparison of Different Training Steps on the Geometry3K Dataset

Method \Step	0	1000	2000	3000	4000	5000	6000	7000	8000	9000
Instruct model	40.43	-	-	-	-	-	-	-	-	-
GRPO	-	40.60	42.93	38.77	39.77	38.94	39.10	40.10	41.36	43.10
Uniform ReDit	-	43.37	43.89	44.01	44.23	44.23	44.23	44.12	44.36	44.36
Gauss ReDit	-	43.67	43.98	44.03	44.25	44.25	44.25	44.25	44.67	44.67

Tables 7, 8, 9 show the comparison of ReDit on different datasets. ReDit significantly improves the convergence speed of GRPO. At any same step, ReDit achieves better performance.

## E.2 Baseline Result

In this section, we present all numerical results in Fig. 5. As shown in Table 10, we demonstrate the effect of using ReDit on GSM8K based on the GRPO improvement method. The experimental results show that ReDit can also improve the convergence speed and performance on these algorithms.

## E.3 Variance Result

In this section, we show more results on the performance of ReDit as the perturbation changes. As shown in Figure 12, the variance of uniform perturbation is similar to the variance of Gaussian perturbation, and the appropriate variance can achieve the best performance. The specific numerical results are shown in Tables 11 and 12.

## E.4 Scheduled Perturbation Result

In this section, we show the changing trends of different scheduled perturbation strategies, as shown in Figure 13. We took the perturbation of Gauss distribution as an example and conducted experiments. The experimental results are shown in Table 13. The CosineReverse strategy shows the best performance.

Table 10: Performance Comparison at Different Training Steps on Different Baseline

Method \ Step	1000	2000	3000	4000	5000	6000	7000	8000	9000
DAPO	84.99	86.20	86.35	86.35	86.75	87.04	87.12	87.17	87.52
Uniform ReDit	87.03	87.15	87.26	87.54	87.54	87.69	87.83	88.03	88.57
Gauss ReDit	87.76	87.96	88.01	88.01	88.10	88.37	88.67	88.96	89.34
DR.GRPO	84.69	84.23	84.53	84.91	85.67	85.67	85.67	85.90	86.13
Uniform ReDit	86.27	86.36	86.45	86.54	86.75	87.03	87.26	87.16	87.34
Gauss ReDit	86.47	86.23	87.10	87.16	87.56	87.67	87.67	87.67	87.69
REINFORCE++	84.91	84.69	85.06	85.14	85.14	85.14	86.10	86.17	86.25
Uniform ReDit	86.21	86.11	86.67	86.31	86.75	87.01	87.26	87.59	87.59
Gauss ReDit	86.17	86.27	86.47	86.83	86.83	87.06	87.63	87.76	87.96

Performance Comparison of Different Uniform Variance

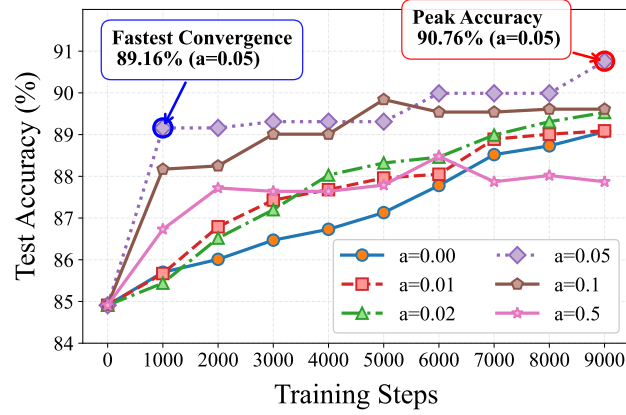


Figure 12: ReDit uniform perturbation performance changes with variance.

Table 11: Performance Comparison of Different variance on the Gauss Perturbation

Variance \ Step	1000	2000	3000	4000	5000	6000	7000	8000	9000
0.01	85.97	87.01	87.40	87.54	87.92	88.76	88.84	89.54	89.54
0.02	86.40	87.70	88.16	89.23	89.39	90.22	90.14	90.14	90.14
0.05	89.02	89.37	89.61	89.54	89.54	89.54	89.61	89.61	90.46
0.1	87.64	89.08	89.69	89.84	90.07	89.84	89.84	89.84	90.07
0.3	87.87	88.48	88.78	88.93	89.39	89.39	89.39	89.46	89.46
0.5	86.81	87.57	87.41	87.64	87.64	87.95	88.32	88.48	88.95

Table 12: Performance Comparison of Different variance on the Uniform Perturbation

Variance \ Step	1000	2000	3000	4000	5000	6000	7000	8000	9000
0.01	85.67	86.79	87.43	87.68	87.96	88.05	88.89	89.01	89.09
0.02	85.44	86.52	87.20	88.03	88.32	88.46	88.99	89.31	89.53
0.05	89.16	89.16	89.31	89.31	89.31	89.99	89.99	89.99	90.76
0.1	88.17	88.25	89.01	89.01	89.84	89.54	89.54	89.61	89.61
0.3	87.49	88.25	88.25	88.02	88.17	87.95	88.93	88.70	88.78
0.5	86.73	87.72	87.64	87.64	87.79	88.48	87.87	88.02	87.87

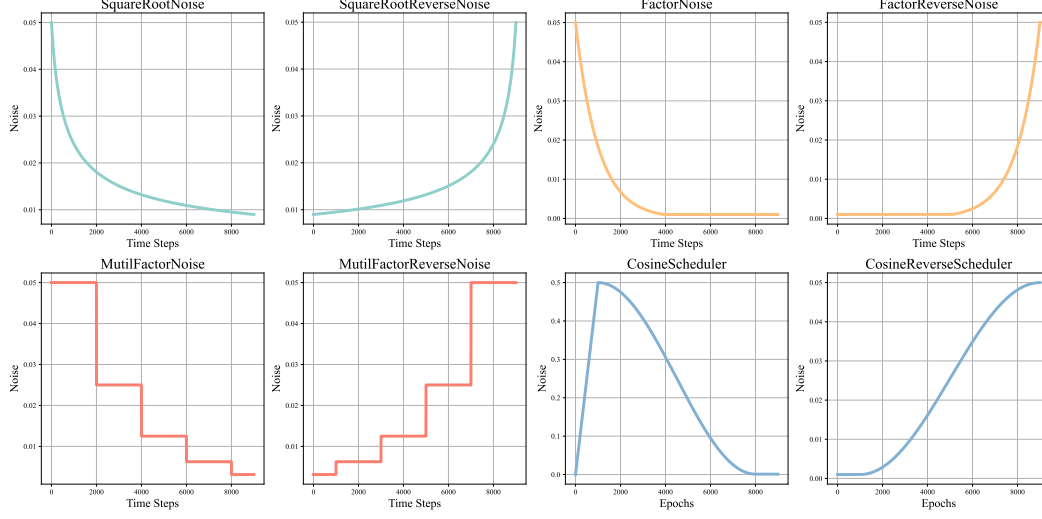


Figure 13: ReDit scheduled perturbation Variance trend with training step (taking the original variance as 0.05 as an example)

Table 13: Performance Comparison of Different Scheduled Perturbation Methods

Method \Step	1000	2000	3000	4000	5000	6000	7000	8000	9000
SquareRoot	88.10	89.31	88.93	89.69	89.46	89.46	89.46	89.46	90.22
SquareRootReverse	88.55	89.54	89.46	90.07	90.07	89.31	89.61	89.54	89.69
Factor	88.25	88.63	89.69	89.46	89.23	89.54	89.46	89.31	89.69
FactorReverse	88.48	88.32	89.39	88.78	88.93	89.54	89.61	89.76	89.46
MutilFactor	87.87	89.31	89.01	89.01	89.01	89.61	89.16	89.61	89.46
MutilFactorReverse	88.17	88.78	88.86	89.01	88.93	88.93	89.39	89.16	89.54
Cosine	88.32	88.32	89.39	89.84	89.76	89.61	90.14	90.46	90.23
CosineReverse	89.08	87.95	89.54	89.08	89.16	90.37	90.07	90.84	91.84