Step-Controlled DPO: Leveraging Stepwise Errors for Enhancing Mathematical Reasoning of Language Models

Anonymous authors
Paper under double-blind review

Abstract

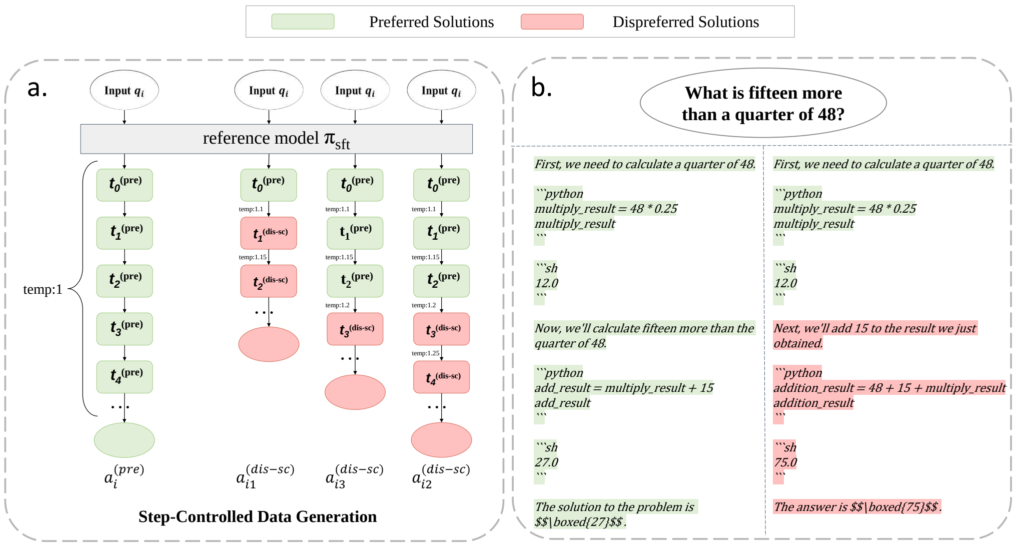
Direct Preference Optimization (DPO) has proven effective at improving the performance of large language models (LLMs) on downstream tasks such as reasoning and alignment. In this work, we propose Step-Controlled DPO (SCDPO), a method for automatically providing stepwise error supervision by creating negative samples of mathematical reasoning rationales that start making errors at a specified step. By applying these samples in DPO training, SCDPO can better align the model to avoid reasoning errors and output accurate reasoning steps. Qualitative analysis of the credit assignment of SCDPO and DPO demonstrates the effectiveness of SCDPO at identifying errors in mathematical solutions. We then apply SCDPO to an InternLM2-20B model, resulting in a 20B model that achieves competitive scores of 88.5% on GSM8K and 58.1% on MATH, rivaling all other open-source LLMs, showing the great potential of our method. The code, models and data are released to inspire future work.

1 Introduction

Recently, Direct Preference Optimization (DPO; Rafailov et al. (2024b)) has emerged as a popular choice for aligning large language models (LLMs) with relative feedback to improve the quality of generated text. Prior works Christiano et al. (2023); Pal et al. (2024); Xu et al. (2024) have demonstrated that reinforcement learning algorithms and DPO can improve the mathematical reasoning abilities of LLMs, making the generated reasoning process more controllable. The final answer to a mathematical problem serves as a natural way to judge the quality of the model's response, since a mathematical problem typically has a single correct answer. As a result, the responses producing the correct final answers are desirable and can serve as the preferred samples, while the ones reaching incorrect final answers are undesirable and can serve as the dispreferred samples.

However, solutions to a mathematical problem can be diverse, with many different reasoning paths arriving at the correct final answer and many subtle ways to make mistakes. Determining the preferred and dispreferred responses based on the final answer is coarse and may be inadequate for capturing *the intricacies of the multi-step mathematical reasoning process.* Previous studies introduce process supervision Lightman et al. (2023), but it requires large amounts of meticulous and expensive human annotation and only applies to traditional RL algorithms.

In this paper, we show how to automatically provide explicit stepwise preference supervision by generating diverse dispreferred solutions that start making errors at a specific step. We propose *Step-Controlled DPO (SCDPO)*, an simple yet effective algorithm that introduces stepwise supervision without necessitating extra human annotation. This approach starts with a model finetuned with question-solution pairs and possessing initial math-solving capabilities, which is used to generate solutions to a set of math problems. We choose the solutions whose final answers match those of the ground truth. We take each of these correct solutions and start generating with the model via modulating the hyperparameter of the model, i.e., increasing the temperature of the final softmax function, from various intermediate steps of that solution, and retain the samples where the final answer is incorrect. In this way, the steps before the intermediate step are the same as the original correct solution, while the steps after are the ones with possible errors. During DPO training, the correct solutions are the preferred samples, and they are paired with the wrong solutions generated



~~~
First, we need to calculate a quarter of 48.

```python
multiply_result = 48 * 0.25
multiply_result
```

```sh
12.0
```

Now, we'll calculate fifteen more than the quarter of 48.

```python
add_result = multiply_result + 15
add_result
```

```sh
27.0
```

The solution to the problem is $$\boxed{27}$$.
~~~

~~~
First, we need to calculate a quarter of 48.

```python
multiply_result = 48 * 0.25
multiply_result
```

```sh
12.0
```

Next, we'll add 15 to the result we just obtained.

```python
addition_result = 48 + 15 + multiply_result
addition_result
```

```sh
75.0
```

The answer is $$\boxed{75}$$.
~~~

Figure 1: Demonstration and example of the step-controlled data generation process. **a.** Step-controlled data generation. First, a solution reaching the correct final answers is collected, which we denote as $a_i^{(\text{pre})}$. Then, erroneous solutions that reach incorrect final answers are generated, starting from intermediate steps of $a_i^{(\text{pre})}$, creating dispreferred solutions $a_{i1}^{(\text{dis-sc})}$, $a_{i2}^{(\text{dis-sc})}$, and $a_{i3}^{(\text{dis-sc})}$. These dispreferred solutions share the steps before the intermediate steps with $a_i^{(\text{pre})}$. The temperature of the newly generated steps gradually increases with each step to make the generation more erroneous. **b.** An example of a pair of preferred and dispreferred solutions. The dispreferred solution starts making errors after a particular intermediate step.

in this way, with the question and the steps before the intermediate step as the prompts. These step-controlled training samples help models learn detailed reasoning abilities and are mixed with naive DPO training data produced by only checking the final answer, which optimizes the general form of the solution.

Our contributions are as follows:

- We introduce SCDPO, a method that automatically provides explicit stepwise supervision to enhance mathematical abilities of LLMs.
- We conduct pilot experiments on chain-of-thought and code-integrated solutions, showing that SCDPO can effectively improve mathematical problem-solving performance of three different SFT models. We also conduct qualitative analysis of credit assignment of SCDPO.
- Using SCDPO, we finetune an InternLM2-20B model, which reaches 88.5% on GSM8K Cobbe et al. (2021) and 58.1% on MATH Hendrycks et al. (2021), demonstrating the great potential of our method.

2 Step-Controlled DPO Pipeline

In this section, we introduce Step-Controlled DPO (SCDPO), a pipeline for automatically generating preferred and dispreferred responses to math problems, with annotations of erroneous solving steps, and using these responses in DPO training to enhance the mathematical reasoning abilities of LLMs. Our method consists of two stages: step-controlled data generation, and step-aware DPO training. The two stages construct a feedback-alignment framework that is both effective and cost-efficient.

Initial Model. Our method starts with an initial model, denoted as π_{SFT}, which has been finetuned with question-solution pairs from math datasets such as GSM8K and MATH. When prompted with a math problem q, π_{SFT} is able to generate a step-by-step solution, denoted as a. a can be broken down into a sequence of reasoning steps, for example, $a = (t_0, \dots, t_m)$. Here, t_i $(i = 0, \dots, m)$ represents either a code reasoning step or a natural language reasoning step within a.

2.1 Step-Controlled Data Generation

The data we collect is in two parts: naive DPO data D_{naive} and Step-Controlled DPO data D_{SC}.

Generation of D_{naive}. D_{naive} contains pairs of preferred-dispreferred samples, used to optimize the general form of the solution. To create D_{naive}, we prompt π_{SFT} with math questions in the training sets of GSM8K and MATH. Each question is presented to π_{SFT} multiple times and various solutions are generated, with a temperature of 1. If a solution reaches the same final answer as the ground truth, and no errors or adjustments occur at any of the reasoning steps (we detect these by looking for strings like “error” or “apologies”), the solution is seen as preferred, while the solutions that reach answers different from the ground truth are considered dispreferred. To find out the frequency of incorrect solution process reaching the correct final answer, We randomly sampled 87 solutions that reach correct final answers, and found that of the 369 reasoning steps in these solutions, only 2 contain errors, which is a very small percentage (about 0.5%). This demonstrates that, in most cases, a correct final answer indicates correct intermediate steps. The solution generation of each question stops when at least one preferred solution and one dispreferred solution are generated, or the number of solutions generated reaches an upper limit of 100. The resulting data can be expressed as:

$$D_{\text{naive}} = \{(q_i, a_i^{\text{(pre)}}, a_i^{\text{(dis)}}) : i = 1, \dots, N_{\text{naive}}\} \tag{1}$$

Here, q_i denotes the ith question, while $a_i^{\text{(pre)}}$ and $a_i^{\text{(dis)}}$ represent the preferred and dispreferred solution to the ith question.

Generation of D_{SC}. In order to generate solutions with stepwise error information for DPO training, we propose a method to automatically generate training data with errors starting to occur at a controlled step. The process is demonstrated in Fig. 1. We first take a preferred solution from D_{naive}, denoted as $a_i^{\text{(pre)}} = (t_0^{\text{(pre)}}, \dots, t_k^{\text{(pre)}}, t_{k+1}^{\text{(pre)}}, \dots, t_{m_i}^{\text{(pre)}})$. Here, $t_k^{\text{(pre)}}$ is a random intermediate step within $a_i^{\text{(pre)}}$. As $a_i^{\text{(pre)}}$ is a correct solution, $t_0^{\text{(pre)}}, \dots, t_{m_i}^{\text{(pre)}}$ can all be seen as correct steps. As shown in Fig. 1 a, to create a solution with errors occurring after step k, we present π_{SFT} with sequence $(q_i, t_0^{\text{(pre)}}, \dots, t_k^{\text{(pre)}})$, and raise the temperature of the final softmax function to affect the generation quality, increasing the occurrence of errors in the following steps. Raising the temperature causes the model performance to become unstable and erroneous. We observe that when the temperature is instantly raised and remains at a high value, the model can generate garbled strings as errors accumulate, which does not represent any reasoning mistakes and contains no valuable information. To avoid this, we adopt a gradually increasing temperature, which initially starts at 1.1, and increases by 0.05 with each generated step, until the generation ends or the temperature reaches 1.4. This setting empirically reduces the frequency of the occurrence of garbled text, while increasing the error rate and diversity of generated errors. We generate the steps following $(q_i, t_0^{\text{(pre)}}, \dots, t_k^{\text{(pre)}})$ multiple times, until one reaching an incorrect answer is generated. Appending the generated steps to $(t_0^{\text{(pre)}}, \dots, t_k^{\text{(pre)}})$, we get a dispreferred solution with step-controlled error, denoted as $a_{ik}^{\text{(dis-sc)}} = (t_0^{\text{(pre)}}, \dots, t_k^{\text{(pre)}}, t_{k+1}^{\text{(dis-sc)}}, \dots, t_{n_i}^{\text{(dis-sc)}})$, where the sequence $(t_{k+1}^{\text{(dis-sc)}}, \dots, t_{n_i}^{\text{(dis-sc)}})$ is erroneous. An example is presented in Fig. 1 b. The resulting data can be expressed as:

$$D_{\text{SC}} = \{(q_i, a_i^{\text{(pre)}}, a_{ik}^{\text{(dis-sc)}}) : i = 1, \dots, N_{\text{SC}}\} \tag{2}$$

Here, q_i denotes the ith question, while $a_i^{\text{(pre)}}$ is the preferred solution, and $a_{ik}^{\text{(dis-sc)}}$ is the dispreferred solution with step-controlled error that occurs after $t_k^{\text{(pre)}}$. N_{SC} is the number of questions in D_{SC}, while m_i is the index of the last step of $a_i^{\text{(pre)}}$.

2.2 Step-Controlled DPO Training

Having collected D_{naive} and D_{SC}, we apply them to DPO training. D_{naive} serves to regulate the general form of solutions, while D_{SC} supervises the model’s reasoning on a step level. During DPO training, samples in D_{naive} and D_{SC} are mixed together randomly, and the DPO loss is applied to each sample. For samples from D_{naive}, the loss is applied to all steps in the preferred and dispreferred solutions, which can be written as:

$$\begin{aligned}&\mathcal{L}_{\text{naive}}(\pi_\theta;\pi_{\text{SFT}})\\&=-\mathbb{E}_{(q_i,a_i^{(\text{pre})},a_i^{(\text{dis})})\sim\mathcal{D}_{\text{naive}}}[\log\sigma(\beta\log\frac{\pi_\theta(a_i^{(\text{pre})}|q_i)}{\pi_{\text{SFT}}(a_i^{(\text{pre})}|q_i)}\\&-\beta\log\frac{\pi_\theta(a_i^{(\text{dis})}|q_i)}{\pi_{\text{SFT}}(a_i^{(\text{dis})}|q_i)})]\end{aligned}\tag{3}$$

For a pair of preferred and dispreferred solutions in D_{SC} where the erroneous steps are generated starting from the kth step of the preferred solution, the preferred solution can be denoted as $a_i^{(\text{pre})}$, and the dispreferred solution can be denoted as $a_{ik}^{(\text{dis-sc})}$. The first k reasoning steps are shared between the pair of solutions, which we denote as $a_{ik-\text{front}}^{(\text{pre})}$. The erroneous steps after the kth step in $a_{ik}^{(\text{dis-sc})}$ is denoted as $a_{ik-\text{end}}^{(\text{dis-sc})}$, while the correct steps in $a_i^{(\text{pre})}$ after the kth step is denoted as $a_{ik-\text{end}}^{(\text{pre})}$. SCDPO directly contrast between $a_{ik-\text{end}}^{(\text{dis-sc})}$ and $a_{ik-\text{end}}^{(\text{pre})}$, applying the DPO loss only on the different steps.

$$\begin{aligned}&\mathcal{L}_{\text{SC}}(\pi_\theta;\pi_{\text{SFT}})=\\&-\mathbb{E}_{(q_i,a_i^{(\text{pre})},a_{ik}^{(\text{dis-sc})})\sim\mathcal{D}_{\text{SC}}}[\log\sigma(\beta\log\frac{\pi_\theta(a_{ik-\text{end}}^{(\text{pre})}|q_i,a_{ik-\text{front}}^{(\text{pre})})}{\pi_{\text{SFT}}(a_{ik-\text{end}}^{(\text{pre})}|q_i,a_{ik-\text{front}}^{(\text{pre})})}\\&-\beta\log\frac{\pi_\theta(a_{ik-\text{end}}^{(\text{dis-sc})}|q_i,a_{ik-\text{front}}^{(\text{pre})})}{\pi_{\text{SFT}}(a_{ik-\text{end}}^{(\text{dis-sc})}|q_i,a_{ik-\text{front}}^{(\text{pre})})})]\end{aligned}\tag{4}$$

Combining $\mathcal{L}_{\text{naive}}$ and $\mathcal{L}_{\text{SC}}$, the final loss function of Step-Controlled DPO is as follows:

$$\mathcal{L}_{\text{SCDPO}}=\mathcal{L}_{\text{naive}}+\mathcal{L}_{\text{SC}}\tag{5}$$

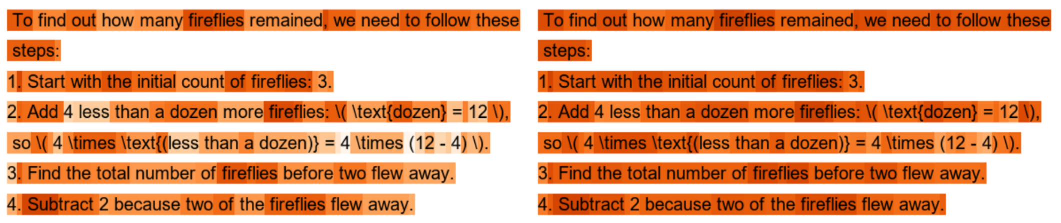
In this way, $\mathcal{L}_{\text{naive}}$ optimizes the general form of the solution, while $\mathcal{L}_{\text{SC}}$ focuses on detailed reasoning steps, thus improving the model's accuracy in solving mathematical problems.

3 Theoretical Explanation of Step-Controlled DPO

Theoretical Insight. In this section, we provide some theoretical insights into why SCDPO can effectively enhance the reasoning ability of LLMs. As explained in Rafailov et al. (2024a), the DPO loss can be cast into token-level MDP. Similarly, we can also derive a step-level MDP for $\mathcal{L}_{\text{SC}}$ as follows:

$$\begin{aligned}&\mathcal{L}_{\text{SC}}(\pi_\theta;\pi_{\text{SFT}})=\\&-\mathbb{E}_{(q_i,a_i^{(\text{pre})},a_{ik}^{(\text{dis-sc})})\sim\mathcal{D}_{\text{SC}}}[\log\sigma((\sum_{j=k+1}^{m_i}\beta\log\frac{\pi_\theta(t_j^{(\text{pre})}|q_i,t_{<j})}{\pi_{\text{SFT}}(t_j^{(\text{pre})}|q_i,t_{<j})})\\&-(\sum_{j=k+1}^{n_i}\beta\log\frac{\pi_\theta(t_j^{(\text{dis-sc})}|q_i,t_{<j})}{\pi_{\text{SFT}}(t_j^{(\text{dis-sc})}|q_i,t_{<j})}))]\end{aligned}\tag{6}$$

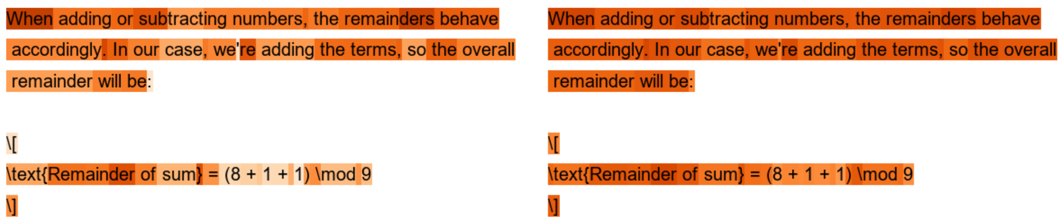
Here, $\beta\log\frac{\pi_\theta(t_j^{(\text{pre})}|q_i,t_{<j})}{\pi_{\text{SFT}}(t_j^{(\text{pre})}|q_i,t_{<j})}$ and $\beta\log\frac{\pi_\theta(t_j^{(\text{dis-sc})}|q_i,t_{<j})}{\pi_{\text{SFT}}(t_j^{(\text{dis-sc})}|q_i,t_{<j})}$ represent the reward of a single preferred or dispreferred step. For naive DPO, all steps in the preferred and dispreferred solutions have their



To find out how many fireflies remained, we need to follow these steps:
1. Start with the initial count of fireflies: 3.
2. Add 4 less than a dozen more fireflies: \(\text{dozen} = 12 \),
so \(4 \times \text{(less than a dozen)} = 4 \times (12 - 4) \).
3. Find the total number of fireflies before two flew away.
4. Subtract 2 because two of the fireflies flew away.

To find out how many fireflies remained, we need to follow these steps:
1. Start with the initial count of fireflies: 3.
2. Add 4 less than a dozen more fireflies: \(\text{dozen} = 12 \),
so \(4 \times \text{(less than a dozen)} = 4 \times (12 - 4) \).
3. Find the total number of fireflies before two flew away.
4. Subtract 2 because two of the fireflies flew away.

Figure 2: Credit assignment of part of a solution for a GSM8K problem. Each token is colored corresponding to the DPO implicit reward as expressed in Eq. 7 (darker is higher). The left is the credit assignment of SCDPO, which correctly highlights the error – 4 less than a dozen is not 4 times (12 - 4), while the credit assignment of DPO on the right fails to highlight it.



When adding or subtracting numbers, the remainders behave accordingly. In our case, we're adding the terms, so the overall remainder will be:

```
\[
\text{Remainder of sum} = (8 + 1 + 1) \mod 9
\]
```

When adding or subtracting numbers, the remainders behave accordingly. In our case, we're adding the terms, so the overall remainder will be:

```
\[
\text{Remainder of sum} = (8 + 1 + 1) \mod 9
\]
```

Figure 3: Credit assignment of part of a solution for a MATH problem. Each token is colored corresponding to the DPO implicit reward as expressed in Eq. 7 (darker is higher). The left is the credit assignment of SCDPO, which correctly highlights the error – as the original question was "Find the remainder when $8 \cdot 10^{18} + 1^{18}$ is divided by 9", the remainders of the terms 8, 10^{18}, and 1^{18} should not be summed, while the credit assignment of DPO on the right fails to highlight the error.

rewards affecting the loss. However, many steps in the dispreferred solution are actually correct, as the error often occurs in a later step. Step-Controlled DPO reduces the range of steps, starting from the $(k+1)$th step, from which the dispreferred steps are more likely to be erroneous due to the raised sampling temperature. The focus of the optimization is thus cast on the errored steps rather than the whole solution, letting the model learn more detailed reasoning abilities.

Qualitative Evaluation of Credit Assignment of SCDPO. We perform qualitative evaluation of credit assignment on two models trained with SCDPO and DPO respectively. For a sequence of tokens $\mathbf{x} = (x_0, \ldots, x_m)$, where x_i is the ith token in the sequence, we denote all the tokens before x_i as $\mathbf{s}_i$, written as $\mathbf{s}_i = (x_0, \ldots, x_{i-1})$. As introduced in recent research Rafailov et al. (2024a), the DPO implicit reward can be expressed as follows:

$$r(\mathbf{s}_i, x_i) = \beta \log \pi(x_i|\mathbf{s}_i) - \beta \log \pi_{\text{SFT}}(x_i|\mathbf{s}_i) \tag{7}$$

Here $r(\mathbf{s}_i, x_i)$ denotes the DPO implicit reward of token x_i, which is the value we visualize as the background color of the token. A darker color represents a higher reward value. As demonstrated in Fig. 2 and Fig. 3, when presented with an incorrect reasoning step, SCDPO more accurately identifies the incorrect tokens compared to DPO. Fig. 2 shows part of a solution for a GSM8K question. In step 2, the solution incorrectly interprets "4 less than a dozen" as "$4 \times (12 - 4)$", when it should have been "$(12 - 4)$". The SCDPO model correctly highlights "$4 \times (12 - 4)$", while the DPO does not. Fig. 3 shows part of a solution for a MATH question. The solution sums the terms in the expression when two of the terms should have been multiplied. SCDPO correctly highlights the incorrect solution, while DPO does not. These examples show that the stepwise supervision provided in SCDPO results in a better token-level understanding of reasoning errors.

4 Experiments

In this section, we first train a 20B model using SCDPO, reaching a performance rivaling all other models of similar scale. Then, we perform a comprehensive empirical comparison between SCDPO

Model	**Size**	**English**							**Chinese**		
		GSM8K	**MATH**	**OCW**	**hung-arian**	**Mathe-matics**	**SVA-MP**	**Simul-eq**	**APE-210K**	**CMA-TH**	**MGSM-zh**
Closed-Source Models											
GPT-3.5	-	80.8	34.1	-	41	-	-	-	-	73.8	-
GPT-4	-	<u>93.6</u>	<u>53.6</u>	**30.1**	**92**	-	-	-	<u>84.2</u>	**89.3**	-
GPT-4 Code Interpreter	-	**97.0**	**69.7**	-	-	-	-	-	-	-	-
GLM-4 [1]	-	91.8	49.0	-	<u>75</u>	-	-	-	**93.5**	<u>89.0</u>	-
Open-Source Models											
MARIO	7B	74.5	48.3	21.7	-	-	-	-	-	-	-
Qwen2	7B	85.7	52.9	-	-	-	-	-	-	-	-
Math-Shepherd	7B	84.1	33.0	-	-	-	-	-	-	-	-
SeaLLM-v2	7B	78.2	27.5	-	-	-	-	-	-	-	64.8
DeepSeekMath-RL	7B	86.7	<u>58.8</u>	-	-	-	-	-	71.9	87.6	<u>78.4</u>
SVPO	7B	81.7	**59.5**	**34.2**	-	-	-	-	-	-	-
Skywork-13B-Math	13B	72.3	17.0	-	39	-	-	-	74.4	77.3	-
InternLM2-Math	20B	80.7	54.3	-	66	-	-	-	-	-	-
MathGenie	20B	<u>87.7</u>	55.7	-	-	85.1	87.3	88.5	-	-	-
ChatGLM3-32B	32B	82.6	40.6	-	73	-	-	-	**89.4**	85.6	-
Yi-Chat	34B	76.0	15.9	-	39	-	-	-	65.1	77.7	-
ToRA	34B	80.7	50.8	5.5	-	77.9	80.5	50.2	-	53.4	41.2
MAmmoTH	70B	76.9	41.8	-	-	65.4	84.3	51.8	-	-	-
MathCoder	70B	83.9	45.1	-	-	74.4	84.9	77.0	-	-	-
WizardMath-v1.0	70B	81.6	22.7	-	-	-	-	-	-	65.4	64.8
InternLM2-SFT	20B	86.4	55.8	21.6	71	84.0	86.9	91.2	77.1	88.4	74.8
InternLM2-SFT-DPO	20B	87.0	57.6	25.5	<u>74</u>	85.6	<u>89.7</u>	<u>92.6</u>	78.7	<u>89.9</u>	76.0
InternLM2-SFT-DPO$_{\text{(d-e)}}$	20B	88.2	57.5	24.5	73	<u>86.3</u>	88.9	91.1	78.8	89.3	76.0
InternLM2-SFT-SCDPO	20B	**88.5**	58.1	<u>29.4</u>	**78**	**87.5**	**90.2**	**93.6**	<u>79.3</u>	**90.3**	**80.4**

Table 1: Performance of open-source and closed-source models on seven English datasets, GSM8K, MATH, OCW, hungarian, Mathematics, SVAMP and Simuleq, and three Chinese datasets, APE210K, CMATH, and MGSM-zh. All results reported are based on greedy decoding. The best models are marked in **bold**, and the second best models are <u>underlined</u>. Our 20B model trained on SCDPO outperforms SFT and naive DPO on all 10 datasets, demonstrating performance rivaling all other open-source models of similar scales.

Method	**Mistral-7B-Ours**		**MetaMath-Mistral-7B**		**MathCoder-Mistral-7B**	
	GSM8K	MATH	GSM8K	MATH	GSM8K	MATH
SFT (Baseline)	76.8	43.2	77.7	28.2	78.1	39.3
SFT-continued	76.3	43.9	76.8	28.5	78.2	40.3
SFT+DPO	78.8	45.1	81.0	28.7	79.2	42.9
SFT+DPO$_{\text{(d-e)}}$	79.0	45.7	81.4	29.0	78.3	41.1
SFT+DPO+SC	**80.1**	**47.7**	**81.7**	**29.3**	**80.4**	**43.4**

Table 2: Effect of using Step-Controlled DPO (SCDPO) on three different SFT models: Mistral-7B-Ours, MetaMath-Mistral-7B and MathCoder-Mistral-7B. "(d-e)" denote the DPO baseline using the same amount of data as SCDPO. In all three cases, SCDPO outperforms the starting SFT model, continue pretraining on correct samples, naive DPO, and naive DPO with equal amount of data.

and DPO on three kinds of Mistral-7B SFT models. We also present ablation studies to further explain the design of increasing temperature during the generation of erroneous steps and combining D_{naive} with D_{SC} during training.

Model	GSM8K	MATH
Mistral-7B-Ours-SFT	76.8	43.2
Mistral-7B-Ours-SCDPO (temperature=1.0)	78.6	45.9
Mistral-7B-Ours-SCDPO (temperature=1.3)	80.0	45.9
Mistral-7B-Ours-SCDPO (ascending temperature)	**80.1**	**47.7**

Table 3: Pilot experiments of using different temperatures when generating error steps. When temperature equals 1.0, the errors are not diverse enough. When temperature equals 1.3, the model generates unintelligible strings due to accumulated errors. The design of ascending temperature offers more diversity while avoids generating meaningless errors, resulting in the best performance.

4.1 20B Model trained with SCDPO

Training Data. We collect solutions for questions in the training set of APE210K Zhao et al. (2020), GSM8K and MATH from GPT-4 Code Interpreter. Combining 169K samples from APE210K, 34K from GSM8K and 47K from MATH, we get an SFT dataset of 250K question-solution pairs. The SCDPO and DPO training data is collected as described before in Sec. 2.1.

Training Settings. We use InternLM2-20B Cai et al. (2024) as the foundation model, as it has demonstrated high performance in previous works Lu et al. (2024); Cai et al. (2024), even surpassing larger models such as Mixtral-8x7B Jiang et al. (2024) and Llama2-70B Touvron et al. (2023) in some cases. In the SFT stage, we finetune the model with a learning rate of 1.0×10^{-5} for 3 epochs, with a context length of 2048 tokens. In DPO and SCDPO training, we use a learning rate of 1.5×10^{-7} to train the SFT model for 2 epochs, with a context length of 1024 and β set to 0.1. The models are trained on 16 NVIDIA A800 80GB GPUs with a batch size of 64.

Evaluation Datasets. Ten representative mathematical datasets are used in evaluating the models: GSM8K Cobbe et al. (2021), MATH Hendrycks et al. (2021), OCWCourses (OCW) Lewkowycz et al. (2022), Hungarian National Exams (hungarian) Paster (2023), Mathematics Saxton et al. (2019), SVAMP Patel et al. (2021), Simuleq Kushman et al. (2014), APE210K Zhao et al. (2020), CMATH Wei et al. (2023b) and MGSM-zh Shi et al. (2023). The first seven datasets consist of English math questions, while the last three consist of Chinese math questions. The evaluation datasets contain a wide range of problem types, covering mathematical problems from grade-school level to college level, comprehensively evaluating the models' mathematical reasoning abilities. We use greedy decoding for all evaluations.

Baselines. We compare our 20B models with powerful closed-source models such as GPT-3.5 (Brown et al., 2020), GPT-4 OpenAI et al. (2024), GPT-4 Code Interpreter OpenAI et al. (2024) and GLM-4 [2], as well as open-source models such as MARIO Liao et al. (2024), Qwen2 Yang et al. (2024), Math-Shepherd Wang et al. (2024), SeaLLM-v2 Nguyen et al. (2024), DeepSeekMath-RL Shao et al. (2024), SVPO Chen et al. (2024), Skywork-13B-Math Yang et al. (2023a), InternLM2-Math [3] Ying et al. (2024), MathGenie Lu et al. (2024), ChatGLM3-32B-RFT-DPO Xu et al. (2024), Yi-Chat Yi (2023), ToRA Gou et al. (2024), MAmmoTH Yue et al. (2023), MathCoer Wang et al. (2023a) and WizardMath Luo et al. (2023).

Main Results. Tab. 1 displays our main results, as well as various closed-source and open-source baselines. Our model achieves a score of 88.5% on GSM8K, 78 on hungarian, 87.5% on Mathematics, 90.2% on SVAMP, 93.6% on Simuleq, 90.3% on CMATH, and 80.4% on MGSM-zh, surpassing all models with published parameters, and obtaining second-best scores among open-source models on APE210K. Our model obtains a score of 58.1% on MATH, which is close to the best and second-best open-source score of 59.5% and 58.8%. While our model rivals the performance of GPT-3.5 on GSM8K and MATH, and surpasses GPT-4 and GLM-4 on MATH, it still underperforms GPT-4 Code Interpreter on GSM8K and MATH, and GLM-4 on APE210K.

Compared to InternLM2-SFT, InternLM2-SFT-SCDPO consistently increases the score on each of the five datasets by approximately 2% to 3%. Compared to both InternLM2-SFT-DPO, which uses the D_{naive} part of InternLM2-SFT-SCDPO's training data, and InternLM2-SFT-DPO$_{\text{(data-equal)}}$, which

[2] https://open.bigmodel.cn/dev/api#glm-4
[3] https://github.com/InternLM/InternLM-Math

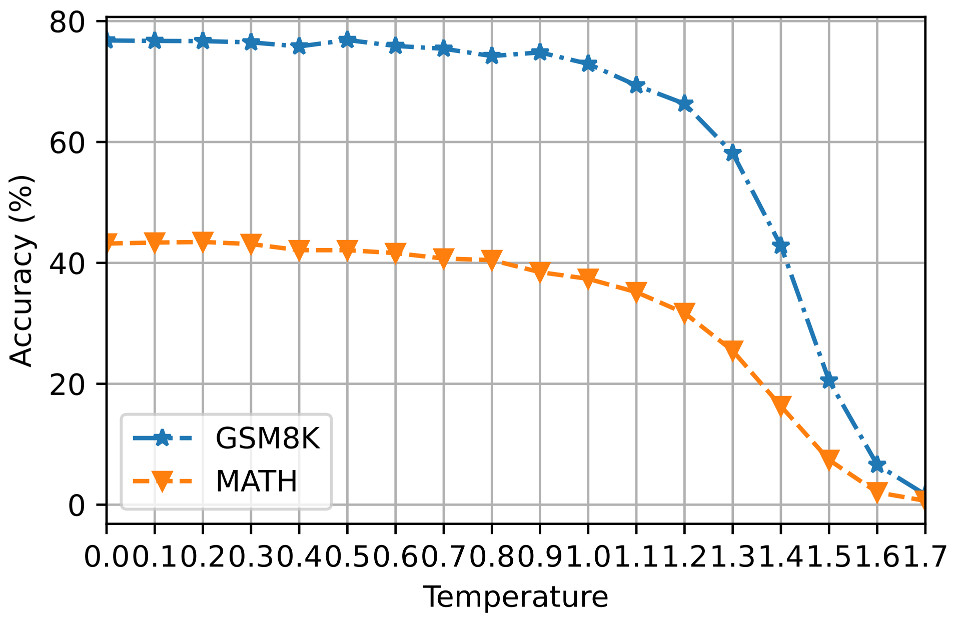


Figure 4: Accuracy of Mistral-7B-Ours (SFT) on GSM8K and MATH when temperature is set at different values.

uses about the same amount of training data as InternLM2-SFT-SCDPO, InternLM2-SFT-SCDPO consistently achieves the best performance across all five datasets, highlighting the effectiveness of SCDPO in enhancing mathematical problem-solving abilities.

4.2 Comparison using Different Starting 7B Models

We validate the generalizability of SCDOP on three baseline SFT models: Mistral-7B-Ours, MetaMath-Mistral-7B, and MathCoder-Mistral-7B. Mistral-7B-Ours is finetuned on the 34K GSM8K samples and 47K MATH samples we collected from GPT-4. MetaMath-Mistral-7B is downloaded from the MetaMath HuggingFace repository[4]. MathCoder-Mistral-7B is finetuned using the MathCodeInstruct dataset Wang et al. (2023a), downloaded from HuggingFace[5]. We collect D_{naive} and D_{SC} as described in 2.1 using problems from GSM8K and MATH. We compare 4 methods of aligning the starting SFT models: 1. Continue finetuning the starting SFT model using supervised finetuning with preferred solutions from D_{naive} (SFT-continued). 2. Doing naive DPO training with D_{naive} (SFT+DPO). 3. Doing naive DPO training with the same amount of training pairs as the SCDPO training, expanded from D_{naive} (SFT+DPO$_{\text{(d-e)}}$). 4. Doing SCDPO training with D_{naive} and D_{SC} (SFT+DPO+SC).

The results are shown in Tab. 2. The purpose of SFT+DPO$_{\text{(d-e)}}$ is to rule out the possibility that the performance gain of SCDPO is the effect of more training samples. SFT-continued shows no obvious gains, likely due to the fact that the models has already been finetuned on many solutions from GSM8K and MATH. As demonstrated in Tab 2, on all three SFT baseline models, SCDPO shows superior performance compared to DPO. This can be attributed to SCDPO's more detailed supervision on the reasoning steps of the math solutions, demonstrating the effectiveness of our method.

4.3 Analysis of the Increasing Temperature Design

We present the result of using different temperature during sampling of erroneous steps in Tab. 3. Originally, we tried sampling for the incorrect solutions at the same temperature as the correct

[4]`https://huggingface.co/meta-math/MetaMath-Mistral-7B`
[5]`https://huggingface.co/datasets/MathLLMs/MathCodeInstruct`

solutions (1.0). However, we observed that generated error steps are less diverse than we hoped. Also, as shown in Fig. 4, the accuracy decreases with the increase of temperature, as the generation becomes less stable. We then tried raising the temperature to 1.3, and found that a notable part of the generated solutions contains incomprehensible strings at later steps due to accumulated errors. Finally, we settled on raising the temperature gradually, which enables more diversity while lowering the frequency of generating unintelligible sentences. As shown in Tab. 3, this method also performs best in the pilot experiments.

5 Related Work

LLM for Mathematical Reasoning. Prior works have explored various methods to enhance mathematical reasoning abilities of LLMs. Prompting methods, such as Chain-of-Thought Wei et al. (2023a), Tree-of-Thought Yao et al. (2023), PAL Gao et al. (2023), Program-of-Thought Chen et al. (2023), and CSV Zhou et al. (2023), use carefully engineered prompts to bring out LLMs' mathematical skills without changing their parameters. Other works optimize parameters of LLMs for enhanced mathematical reasoning through either pretraining or finetuning. Llemma Azerbayev et al. (2024), and MathPile Wang et al. (2023b) continue pretraining LLMs on large amounts of math-related data, while RFT Yuan et al. (2023), Mammoth Yue et al. (2023), MathCoder Wang et al. (2023a), WizardMath Luo et al. (2023), ToRA Gou et al. (2024), MetaMath Yu et al. (2024), MathGLM Yang et al. (2023b), and MathGenie Lu et al. (2024) finetune pretrained models on question-solution pairs. These methods effectively improves LLMs' ability to solve challenging mathematical problems, demonstrating impressive performance on mathematical benchmarks such as GSM8K Cobbe et al. (2021), MATH Hendrycks et al. (2021), etc. Our work builds upon models that have undergone pretraining and finetuning, using DPO to further enhance their mathematical abilities.

Improving Mathematical Reasoning Using Relative Feedback. Reinforcement learning from human (or AI) feedback Christiano et al. (2023); Bai et al. (2022) as well as several direct alignment methods Rafailov et al. (2024b); Azar et al. (2023); Zhao et al. (2023); Pal et al. (2024); Ethayarajh et al. (2024); Liu et al. (2024) have proven effective on various downstream tasks. Our method make use of DPO Rafailov et al. (2024b), introducing a novel way to construct the DPO training data for better enhancement of mathematical abilities of LLMs. Previous works using reinforcement learning or direct alignment methods for improving mathematical reasoning utilize either outcome supervision or process supervision. Outcome supervision such as Shao et al. (2024) is simple and use the outcome of a solution as supervision signal. Lightman et al. (2023) found that process supervision offers better performance than outcome supervision, but needs expert and detailed human or AI annotation, which is difficult to acquire. Math-Shepherd Wang et al. (2023a) and Process Reward Synthesizing **?** estimate process rewards with multiple decoding rationales at each step, and train a reward model with the synthesized rewards. Other works such as Xie et al. (2024), Yuan et al. (2024) and Chen et al. (2024) use tree structure to provide fine-grained supervision, often relying on a critique model to decide the correctness of reasoning steps. Concurrent works such as Setlur et al. (2024) and Lai et al. (2024) rely on GPT-4 to synthesize data or localize errors. In comparison, our method uses increasing temperature to start generating erroneous steps from intermediate steps of a correct solution, and directly contrast erroneous steps with correct steps, offering a simpler, more cost-effective alternative with high performance.

6 Limitations and Future Work

Our work contains the following limitations, and we leave them for future work. Firstly, our work is conducted on purely linguistic models, which struggle to solve mathematical problems requiring an understanding of images. Secondly, due to the stepwise attribute of SCDPO, it is not very effective on solution formats consisting of pure code. It only works on solutions consisting of natural language chain of thought or interleaved natural language and code. A method to properly enhance pure code solutions needs to be derived, which we leave for future work. Thirdly, as with all language models, our models can potentially generate hallucinations or produce misleading solutions, which can have a negative effect. Finally, while the data construction method distributionally narrow down the steps likely to be erroneous, it does not indicate the exact step the error occurs, a problem inherent with synthetic process supervision methods.

7 Conclusion

In this work, we propose Step-Controlled DPO (SCDPO), a method to automatically introduce stepwise error supervision to the process of DPO training by generating dispreferred samples that start making errors at a specified step. SCDPO effectively enhances the mathematical reasoning abilities of LLMs. The 20B model trained with SCDPO on both English and Chinese data achieves high scores on 10 representative mathematical datasets, consistently outperforming naive DPO, demonstrating the effectiveness of our method.

References

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A general theoretical paradigm to understand learning from human preferences, 2023. URL https://arxiv.org/abs/2310.12036.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics, 2024. URL https://arxiv.org/abs/2310.10631.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022. URL https://arxiv.org/abs/2212.08073.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. Internlm2 technical report, 2024. URL https://arxiv.org/abs/2403.17297.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Step-level value preference optimization for mathematical reasoning, 2024. URL https://arxiv.org/abs/2406.10858.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2023. URL https://arxiv.org/abs/2211.12588.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023. URL https://arxiv.org/abs/1706.03741.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization, 2024. URL https://arxiv.org/abs/2402.01306.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023. URL https://arxiv.org/abs/2211.10435.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving, 2024. URL https://arxiv.org/abs/2309.17452.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. In Kristina Toutanova and Hua Wu (eds.), *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 271–281, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1026. URL https://aclanthology.org/P14-1026.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms, 2024. URL https://arxiv.org/abs/2406.18629.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. Mario: Math reasoning with code interpreter output – a reproducible pipeline, 2024. URL https://arxiv.org/abs/2401.08190.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Tianqi Liu, Yao Zhao, Rishabh Joshi, Misha Khalman, Mohammad Saleh, Peter J. Liu, and Jialu Liu. Statistical rejection sampling improves preference optimization, 2024. URL https://arxiv.org/abs/2309.06657.

Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. Mathgenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of llms, 2024. URL https://arxiv.org/abs/2402.16352.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct, 2023. URL https://arxiv.org/abs/2308.09583.

Xuan-Phi Nguyen, Wenxuan Zhang, Xin Li, Mahani Aljunied, Zhiqiang Hu, Chenhui Shen, Yew Ken Chia, Xingxuan Li, Jianyu Wang, Qingyu Tan, Liying Cheng, Guanzheng Chen, Yue Deng, Sen Yang, Chaoqun Liu, Hang Zhang, and Lidong Bing. Seallms – large language models for southeast asia, 2024. URL https://arxiv.org/abs/2312.00738.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Arka Pal, Deep Karkhanis, Samuel Dooley, Manley Roberts, Siddartha Naidu, and Colin White. Smaug: Fixing failure modes of preference optimisation with dpo-positive, 2024. URL https://arxiv.org/abs/2402.13228.

Keiran Paster. Testing language models on a held-out high school national finals exam. https://huggingface.co/datasets/keirp/hungarian_national_hs_ finals_exam, 2023.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems?, 2021. URL https://arxiv.org/abs/2103.07191.

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. From r to q^*: Your language model is secretly a q-function, 2024a. URL https://arxiv.org/abs/2404.12358.

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, 2024b. URL https://arxiv.org/abs/2305.18290.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models, 2019. URL https://arxiv.org/abs/1904.01557.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold, 2024. URL https://arxiv.org/abs/2406.14532.

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

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=fR3wGCk-IXp.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning, 2023a. URL https://arxiv.org/abs/2310.03731.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

Zengzhi Wang, Rui Xia, and Pengfei Liu. Generative ai for math: Part i – mathpile: A billion-token-scale pretraining corpus for math, 2023b. URL https://arxiv.org/abs/2312.17120.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023a. URL https://arxiv.org/abs/2201.11903.

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. Cmath: Can your language model pass chinese elementary school math test?, 2023b. URL https://arxiv.org/abs/2306.16636.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P. Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning, 2024. URL `https://arxiv.org/abs/2405.00451`.

Yifan Xu, Xiao Liu, Xinghan Liu, Zhenyu Hou, Yueyan Li, Xiaohan Zhang, Zihan Wang, Aohan Zeng, Zhengxiao Du, Wenyi Zhao, Jie Tang, and Yuxiao Dong. Chatglm-math: Improving math problem-solving in large language models with a self-critique pipeline, 2024. URL `https://arxiv.org/abs/2404.02893`.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL `https://arxiv.org/abs/2407.10671`.

Liu Yang, Haihua Yang, Wenjun Cheng, Lei Lin, Chenxia Li, Yifu Chen, Lunan Liu, Jianfei Pan, Tianwen Wei, Biye Li, Liang Zhao, Lijie Wang, Bo Zhu, Guoliang Li, Xuejie Wu, Xilin Luo, and Rui Hu. Skymath: Technical report, 2023a.

Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. Gpt can solve mathematical problems without a calculator, 2023b. URL `https://arxiv.org/abs/2309.03241`.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL `https://arxiv.org/abs/2305.10601`.

Yi. A series of large language models trained from scratch by developers at 01-ai. `https://github.com/01-ai/Yi`, 2023.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, Yudong Wang, Zijian Wu, Shuaibin Li, Fengzhe Zhou, Hongwei Liu, Songyang Zhang, Wenwei Zhang, Hang Yan, Xipeng Qiu, Jiayu Wang, Kai Chen, and Dahua Lin. Internlm-math: Open math large language models toward verifiable reasoning, 2024. URL `https://arxiv.org/abs/2402.06332`.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models, 2024. URL `https://arxiv.org/abs/2309.12284`.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. Advancing llm reasoning generalists with preference trees, 2024. URL `https://arxiv.org/abs/2404.02078`.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL `https://arxiv.org/abs/2308.01825`.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning, 2023. URL `https://arxiv.org/abs/2309.05653`.

Wei Zhao, Mingyue Shang, Yang Liu, Liang Wang, and Jingming Liu. Ape210k: A large-scale and template-rich dataset of math word problems, 2020. URL `https://arxiv.org/abs/2009.11506`.

Data	**GSM8K**	**MATH**
D_{SC}	79.0%	46.2%
D_{naive} + D_{SC}	80.1%	47.7%

Table 4: Ablation study of using and not using D_{naive} during training. The starting SFT model is Mistral-7B-Ours.

Yao Zhao, Rishabh Joshi, Tianqi Liu, Misha Khalman, Mohammad Saleh, and Peter J. Liu. Slic-hf: Sequence likelihood calibration with human feedback, 2023. URL `https://arxiv.org/abs/2305.10425`.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification, 2023. URL `https://arxiv.org/abs/2308.07921`.

A Credit Assignment Analysis Examples

In this section, we present several other credit assignment analysis examples, comparing SCDPO to DPO. Fig. 6, Fig. 7 and Fig. 8 show examples of part of the solutions of questions taken from GSM8K and MATH datasets, colored with the DPO implicit reward of each token (darker is higher). As demonstrated in the examples, SCDPO is better than DPO at identifying the errors in the reasoning steps.

B Ablation Study of D_{NAIVE} and D_{SC}

To demonstrate the necessity of combining D_{naive} and D_{SC}, we conduct experiment of using only D_{SC} in DPO training. The results are presented in Tab. 4. As demonstrated in the table, combining D_{naive} and D_{SC} results in better performance than only using D_{SC} during DPO training. This is likely because D_{naive} helps regulate the general format of the generated solutions.

C Analysis of Generated Errors

In this section, we provide quantitative analysis of the erroneous steps generated. We observed seven main kinds of errors: value misusage, condition misinterpretation, coding error, commonsense error, math concept or understanding error, math calculation error, unintelligible strings. The errors are explained as follows:

- Value misusage: misusing values in places where another value should have been used.
- Condition misinterpretation: incorrectly interpreting the meaning or indications of conditions.
- Coding error: making mistakes in code snippets that causes errors.
- Common sense error: misunderstanding of common sense.
- Math concept or understanding error: incorrect recollection or understanding of math concepts.
- Math calculation error: mistakes when making mathematical calculations.
- Unintelligible strings: generation of unintelligible strings that does not represent meaningful reasoning errors.

We randomly sampled 100 incorrect solutions in the training data of SCDPO, and counted the number of each type of error. The result is presented in Fig. 5. As demonstrated in the chart, the reasoning errors generated is diverse, distributed evenly among the different types. Only 4% of the incorrect solutions contain unintelligible strings, demonstrating that the design of gradually increasing temperature can mostly avoid the occurance of meaningless errors.

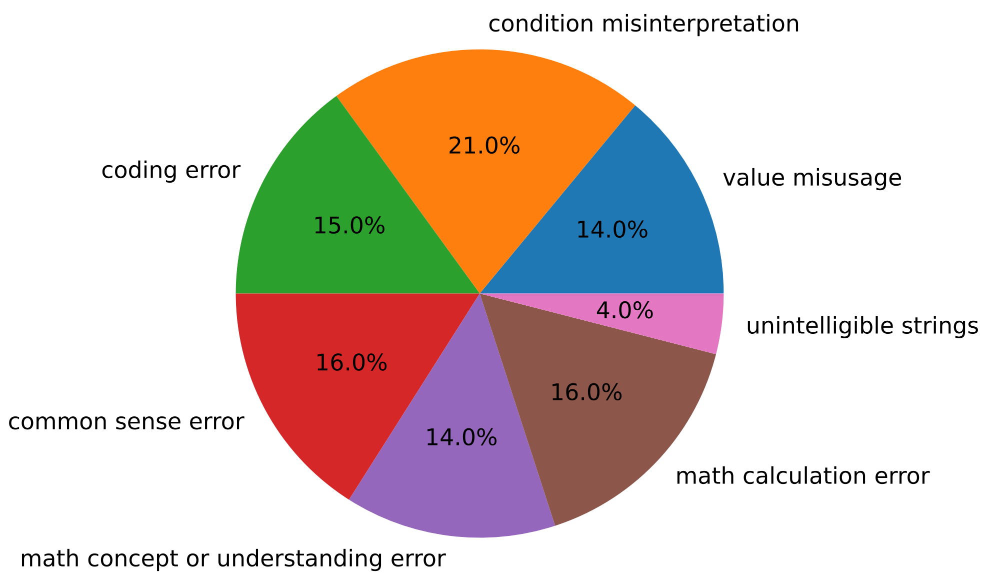
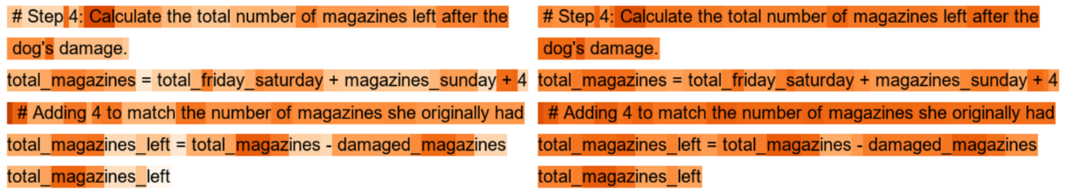


Figure 5: Percentage of each type of error in the 100 examples we sampled.

D Error Rate of Intermediate Steps When the Final Answer is Correct

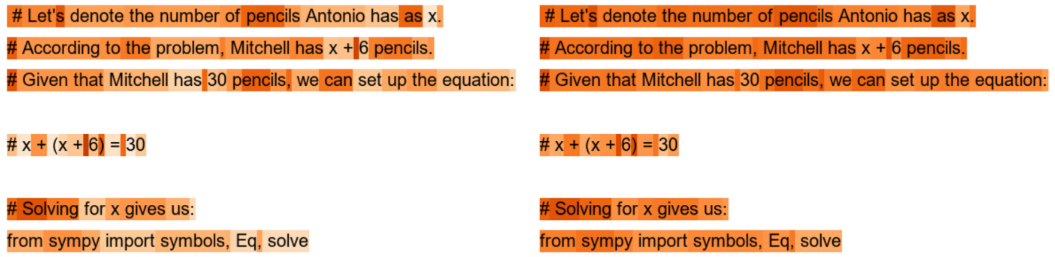
In this section, we discuss the error rate of intermediate steps in solutions that reaches the correct final answer. As we mentioned in the main paper, we randomly sampled 87 solutions that reach correct final answer, and of the 369 reasoning steps in these solutions, only 2 contain errors, which is a very small percentage (about 0.5%). The 2 erroneous steps are in a question whose answer is to ask for “the number of real solutions”. The possible answers to this question is very limited, which facilitates incorrect reasoning steps to happen to reach a correct final answer. We observe that this kind of question is rare in the GSM8K and MATH datasets we chose to perform SCDPO on, so in most cases a correct final answer indicates that the reasoning steps leading to it is highly likely to be correct. We also removed solutions that contain apologies or error messages, so there are no incorrect steps that are later revised in the correct solutions we retained. As a result, the steps before the generated erroneous steps in SCDPO training data are correct with a high confidence.



```
# Step 4: Calculate the total number of magazines left after the
dog's damage.
total_magazines = total_friday_saturday + magazines_sunday + 4
# Adding 4 to match the number of magazines she originally had
total_magazines_left = total_magazines - damaged_magazines
total_magazines_left
```

```
# Step 4: Calculate the total number of magazines left after the
dog's damage.
total_magazines = total_friday_saturday + magazines_sunday + 4
# Adding 4 to match the number of magazines she originally had
total_magazines_left = total_magazines - damaged_magazines
total_magazines_left
```

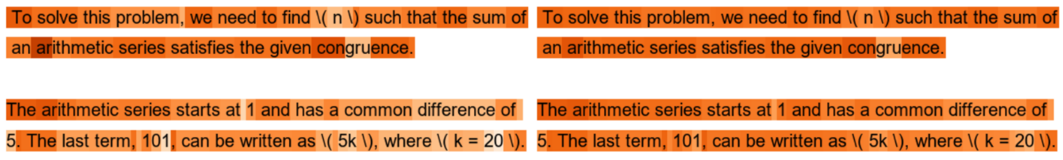
Figure 6: Credit assignment of part of a solution for a GSM8K problem. Each token is colored corresponding to the DPO implicit reward as expressed in Eq. 7 (darker is higher). The left is the credit assignment of SCDPO, which correctly highlighted the error – the number of damaged magazines (which is 4) should not be first added to and then extracted from "total_magazines", while the credit assignment of DPO on the right fails to highlight it.



```
# Let's denote the number of pencils Antonio has as x.
# According to the problem, Mitchell has x + 6 pencils.
# Given that Mitchell has 30 pencils, we can set up the equation:

# x + (x + 6) = 30

# Solving for x gives us:
from sympy import symbols, Eq, solve
```

```
# Let's denote the number of pencils Antonio has as x.
# According to the problem, Mitchell has x + 6 pencils.
# Given that Mitchell has 30 pencils, we can set up the equation:

# x + (x + 6) = 30

# Solving for x gives us:
from sympy import symbols, Eq, solve
```

Figure 7: Credit assignment of part of a solution for a GSM8K problem. Each token is colored corresponding to the DPO implicit reward as expressed in Eq. 7 (darker is higher). The left is the credit assignment of SCDPO, which correctly highlighted the error – Mitchell has 30 pencils, and Antonio has 6 less pencils than Michell, which is $30 - 6$, so the introduction of x is not needed, and $x + (x + 6) = 30$ is incorrect, while the credit assignment of DPO on the right fails to highlight it.



To solve this problem, we need to find \(n \) such that the sum of an arithmetic series satisfies the given congruence.

The arithmetic series starts at 1 and has a common difference of 5. The last term, 101, can be written as \(5k \), where \(k = 20 \).

To solve this problem, we need to find \(n \) such that the sum of an arithmetic series satisfies the given congruence.

The arithmetic series starts at 1 and has a common difference of 5. The last term, 101, can be written as \(5k \), where \(k = 20 \).

Figure 8: Credit assignment of part of a solution for a MATH problem. Each token is colored corresponding to the DPO implicit reward as expressed in Eq. 7 (darker is higher). The left is the credit assignment of SCDPO, which correctly highlighted the error – 101 cannot be written as $5k$ where $k = 20$, while the credit assignment of DPO on the right fails to highlight the error.