Self-Improving Mathematical Reasoning of Large Language Models with a Code-Centric Paradigm

Anonymous ACL submission

Abstract

There is a growing trend of teaching large language models (LLMs) to solve mathematical problems through coding. Existing studies primarily focus on distilling powerful, closed-source models and in-domain data augmentation, equipping LLMs with considerable capabilities for mathematical reasoning via coding. However, the self-improvement of such LLMs by leveraging large-scale, expert-written, diverse math question-answer pairs remains under-explored. To bridge this gap and tackle challenges such as code response assessment, we propose a novel paradigm that uses a code-based critic model to guide steps including question-code data construction, quality control, and complementary evaluation. We also explore different alignment algorithms with self-generated instruction/preference data to foster continuous improvement. Experiments across both in-domain (up to $+5.7\%$) and out-of-domain ($+4.4\%$) benchmarks in English and Chinese demonstrate the effectiveness of self-improving LLMs with the proposed paradigm.

1 Introduction

There has been a growing trend of using code and code interpreters to enhance the performance of large language models (LLMs) in solving mathematical reasoning problems, which helps alleviate the computational burden on LLMs (Chen et al., 2022; Gao et al., 2023b; Zhou et al., 2023). Based on open-source LLMs, a simple yet effective method involves distilling strong closed-source LLMs to generate code-based solutions for given questions, verifying the correctness of the solutions via answer matching, and then training open-source models on the verified data for further self-distillation through sampling, code execution, and answer validation (Wang et al., 2023; Gou et al., 2024; Lu et al., 2024).

An important question remains unanswered: **can such LLMs, which already demonstrate a reasonable ability to solve math problems by writing code, self-improve without continually distilling knowledge from larger closed-source or open-source LLMs?** Previous code-aided studies (Wang et al., 2023; Gou et al., 2024; Lu et al., 2024) primarily focus on in-domain data augmentation using a few representative, small-scale datasets such as GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). However, continually learning from these datasets or their augmented versions is evidently less effective for improving the generalization and performance of LLMs due to the limited diversity.

On the other hand, large-scale, expert-written mathematical question-answer pairs on public web resources remain under-studied for post-training code-aided LLMs. These resources span educational levels from primary school to college and include a variety of question types such as multiple-choice, application, proof, and cloze. To use these resources for the self-improvement of LLMs, **the key challenge is to determine whether the self-generated code responses align with reference answers in diverse formats**. Fortunately, with the aid of an external code interpreter, we are less concerned about computation errors in specific steps (e.g., lines or blocks of code). We assume a code solution is more likely to be correct if its execution result matches the reference answers, thus shifting the focus to comparing the reference answers with the code execution results. Based on our analysis (Section 4.1), we observe that most cases primarily require format conversion between plain text and code syntax (e.g., "(x-5)(xˆ2-4x+7)" vs. "(x-5)*(x**2-4*x+7)" and "(1, -2, 2, -3)" vs. "{A:1, B:-2, C:2, D:-3}") and relatively simple numerical calculations, which do not require advanced logical reasoning abilities.

These observations motivate us to design a critic model that evaluates the correctness of the code execution result against the reference answer by

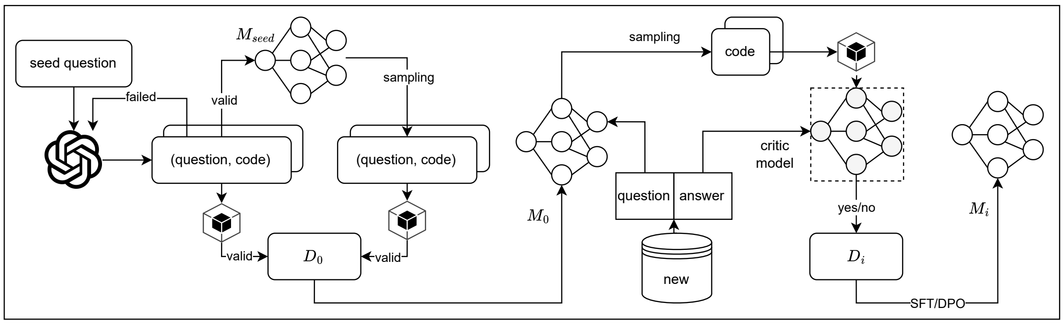

Figure 1: Overview of our code-centric self-improving paradigm.

predicting YES or NO (see examples in Table 1). As illustrated in Figure 1, this critic model is used to guide multiple steps during self-improvement. We first train a model with seed question-code data following previous studies (e.g., (Gou et al., 2024)) and consider it as the initial policy model. In each iteration, we use the current policy model to generate code samples for new questions and keep the highest-scoring valid code responses rated by the critic model for supervised fine-tuning (SFT) in the subsequent iteration. To foster continuous improvement, we also explore different alignment algorithms such as DPO (Rafailov et al., 2024) and ORPO (Hong et al., 2024) with self-generated preference data, where the preference labels are provided by the critic model.

We perform experiments on different model families, such as Llama3-8B (AI@Meta, 2024) and DeepSeek-Coder-7B (Daya Guo, 2024). Experimental results across both in-domain (up to $+5.7\%$) and out-of-domain ($+4.4\%$) benchmarks in English and Chinese demonstrate the effectiveness of self-improving LLMs using our proposed code-centric paradigm with large-scale question-answer pairs from web resources. Notably, we observe a very strong correlation between the traditional heuristic-based evaluation method and the critic model (Section 4.5), with the latter reducing the additional human effort needed to design rules for new math benchmarks. We also find that introducing SFT loss into the DPO training is surprisingly effective in controlling the code response length after preference optimization.

To summarize the contributions of this work:

- We propose a novel paradigm for self-improving the mathematical reasoning abilities of code-aided LLMs by iteratively learning from new web question-answer pairs.
- To better leverage these large-scale resources, we design a critic model that effectively guides various steps such as data construction and filtering. This model can also serve as a complementary evaluation scorer, reducing the reliance on heuristic design for new evaluation tasks.
- Extensive experiments demonstrate the effectiveness of our paradigm, and our comprehensive analysis of the key factors in achieving continuous improvement at different stages may shed light on future studies.

2 Related Work

For automatic math evaluation on well-formatted benchmarks, previous studies mostly use heuristics and external tools (e.g., the Python EVAL() function) to compare answers and predictions (Fourrier et al., 2023; Gao et al., 2023a), which works quite well for single numerical value answers, as seen in datasets such as GSM8K (Cobbe et al., 2021), ASDiv (Miao et al., 2020), and SVAMP (Patel et al., 2021). However, since answers from web resources are diverse in formats and language-code syntactic difference, using carefully designed task-specific heuristics becomes infeasible for comparing answers and code execution results. For datasets that are beyond value-style answers such as MATH (Hendrycks et al., 2021), closed source LLMs are also used for evaluation such as OpenAI-Evals, which however is not cost-effective for rating large-scale code samples.

Several approaches (Li et al., 2023; Yu et al., 2023; Lu et al., 2023; Yuan et al., 2024; Hu et al., 2024) use the LLM itself or use separate critic model (Ouyang et al., 2022; Xu et al., 2024) for scoring or filtering natural-language responses. We focus on tool-assisted assessment of code responses to math questions. Compared to the previously mentioned self-improving studies that use a single LLM to provide feedback on its own generations, we interpret "self" in contrast to distilling knowledge from larger closed-source or open-source LLMs for continuous improvements.

3 Method

3.1 Training an Initial Model

One key factor for a self-improving paradigm is to have a reasonably strong initial model. To train such a model, we first use high-quality seed data to fine-tune a large language model, resulting model M_{seed}. We use M_{seed} to generate code samples and keep up to four predictions per question wherein the result of the code matches the ground truth answer and combines the seed data and the self-distilled data to train M_0, which is further used as the initial model for later self-improving stages. We will introduce more details about the seed data construction in the experiment section.

3.2 Building a Multi-Use Code-Based Critic Model

To self-improve LLMs with large-scale math question-answering data without code annotations, several challenges arise in data utilization, filtering, and evaluation. First, previous studies primarily use pattern-based methods to compare predictions and ground truth answers during validation and evaluation. This works well for GSM-style datasets, where answers are single numbers and well-formatted (e.g., *"72"* in *"...72 clips altogether in April and May.\n #### 72"*). However, pattern-based methods inevitably struggle to handle different types or formats of answers and the gap between natural language and programming language. For example, with the MATH dataset, comparing predictions with the ground truth answers in LaTeX-like format requires numerous human-written patterns. This complexity is compounded when the predictions are in code syntax. Second, given a question, it is still under-explored how to select high-quality, valid code responses.

To address the above challenges, we propose building a code-based critic model optimized by the following objective:

$$L(r_\phi) = -\log r_\phi(y \mid q, a, c, e), \quad (1)$$

where q denotes a question, a is the ground truth answer to q, c represents the code response to q, and e is the execution result of code c. To simplify the task, we let y be either "YES" or "NO". Examples are shown in Table 1. We leave other formulations, such as training a scalar critic model (Ouyang et al., 2022), to future work.

3.3 Code Data Generation

As mentioned previously, our goal is to leverage non-code math data from web resources to continuously improve the mathematical reasoning ability of LLMs, rather than further distilling powerful LLMs. For well-formatted, web-collected data such as APE (Zhao et al., 2020) and CM (Qin et al., 2021), where most answers are one or two numerical values (see an example in Table 2), it is efficient and effective to compare the ground truth answer and the execution result of the code using scripts released by previous studies (Section 4.2). For real-world math data involving multiple types of questions, such as multiple-choice, multiple-question, fill-in-the-blank, application, and proof, using a critic model introduced in the previous section is more flexible and saves the intensive effort of writing task-specific patterns, which is time-consuming and may suffer from relatively low recall. Note that for all math questions, we only use their reference answers to verify the correctness of code execution results instead of directly training on these answers, and we only use benchmarks' training sets.

In the $k+1$-th iteration, for each new question, we use the current policy model π_{θ_k} to generate five code samples and execute them to obtain the results. For questions in the real-world math data, the critic model is then used to predict YES or NO for each response (a_i, c_{ij}, e_{ij}) given q_i. We use the probability of YES or NO as the confidence value for the critic model's judgment. A higher probability score indicates a greater confidence in the code response, either agreeing with or disagreeing with the reference answer.

3.4 Self-Improving with New Data

One natural choice is to perform supervised fine-tuning (SFT) on π_{θ_k} using D_{SFT}:

$$L_{\text{SFT}}(\pi_{\theta_{k+1}}) = -\log \pi_{\theta_{k+1}}(c \mid q) \quad (2)$$

$$D_{\text{SFT}} = \{(q_i, c_{ij}) \mid r_\phi(y = \text{YES} \mid q_i, a_i, c_{ij}, e_{ij})\} \quad (3)$$

As critiques may contain errors, we explore using the probability of each judgement as a con-

System Prompt	Your goal is to evaluate whether the candidate answer provided by the model for a math problem matches the reference answer. Here are the steps to complete the task: – First, carefully read the given math problem. – Next, review the reference answer for the math problem. – Then, examine the candidate answer provided by the model, which may include a program and the result of running that program. – Finally, summarize whether the candidate answer matches the reference answer or can be made to match through simple calculations/conversions. – The response format should be Yes or No.
Instruction	### Question\n\n Given f（1-2x）=3x+1，find f（-3）= __ · ### Reference Answer\n \n 7 \n \n ### Candidate Answer\n \n <code>from sympy import symbols, Eq, solve\n# Define the variable\n x = symbols('x')\n# Establish the equation based on the problem conditions\n equation = Eq(1 - 2*x, -3)\n# Solve the equation\n solution = solve(equation, x, dict=True)\n solution_value = solution[0][x]\n # Calculate f(-3)\n f_neg_3 = 3*solution_value + 1\n f_neg_3</code><solution>7</solution> ### Assessment
Output	Yes
Instruction	### Question\n\n Given the sequence {a<sub>n</sub>} with the sum of the first n terms S<sub>n</sub>=2n-3, what is a<sub>3</sub>? A: 3B: 2C: 1D: 0 ### Reference Answer\n \n B \n \n ### Candidate Answer\n \n <code># Define a function to calculate the nth term\n def calculate_nth_term(n):\n # Based on the derived formula\n an = 2 ** n - 2 ** (n - 1)\n return an\n # Calculate the value of the 3rd term \n a3 = calculate_nth_term(3) # a3 </code><solution>4</solution> ### Assessment
Output	No

Table 1: Example instance of the code-based critique task (we translate the Chinese texts into English).

Question:	Given: Apples cost 6 yuan for 4 kilograms, and oranges cost 11 yuan for 5 kilograms. Uncle Wang buys 16 kilograms of apples and 20 kilograms of oranges. How much should he pay in total?
Answer:	68
Rationale*:	x=6/4*16+11/5*20

Table 2: An example instance of the APE dataset (Zhao et al., 2020) (we translate the question into English; $\star$: we do not use this rationale in our paradigm).

fidence score to filter out noise. Besides, we introduce extra constraints: for each question, we only retain the highest-scoring positive instance $t_{ij} = \{q_i, a_i, c_{ij}, e_{ij}\}$, similar to rejection sampling (Bai et al., 2022), where $t_{ij} \in T_i$ of the same question q_i. To encourage models to learn from more challenging problems, if all instances in T_i are labeled as YES, we discard this question and its corresponding generated code from consideration.

$$\begin{aligned} D_{\text{SFT, H}} = \{(q_i, c_{ij}) \mid\, & r_\phi(y = \text{YES} \mid t_{ij}), \\ & p_{r_\phi}(y = \text{YES} \mid t_{ij}) > \lambda_1, \\ & t_{ij} = \arg\max_{t_{ij} \in T_i} p_{r_\phi}(y = \text{YES} \mid t_{ij}), \\ & \sum_{j=1}^{|T_i|} \mathbf{1}\{r_\phi(y = \text{No} \mid t_{ij})\} \geq \lambda_2\} \end{aligned} \tag{4}$$

where λ_1, λ_2 represent thresholds for filtering and difficulty control.

In addition to supervised fine-tuning a policy model on self-generated SFT data ($D_{\text{SFT, H}}$ or D_{SFT}) iteratively, we can also leverage the negative instances by optimizing the policy on preference data using algorithms such as DPO (Rafailov et al., 2024) and KTO (Ethayarajh et al., 2024). Compared to SFT, these alignment algorithms additionally decrease the probability of losing responses. We mainly focus on DPO in this paper and leave other options for future studies, and we jointly train the policy with the SFT objective to alleviate overfitting to the preference data and ensure a stable update (Hong et al., 2024). See more discussions on the impact of the SFT objective, especially its role in controlling the response length, in Section 4.4.

$$\begin{aligned} L_{\text{DPO}}(\pi_{\theta_{k+1}}) = -\log\sigma\Bigg(& \beta \log \frac{\pi_{\theta_{k+1}}(y_w \mid x)}{\pi_{\theta_k}(y_w \mid x)} \\ & -\beta \log \frac{\pi_{\theta_{k+1}}(y_l \mid x)}{\pi_{\theta_k}(y_l \mid x)}\Bigg) \\ & -\lambda \cdot \log \pi_{\theta_{k+1}}(y_w \mid x) \end{aligned} \tag{5}$$

We can easily leverage our critic model to build preference (c_w, c_l) pairs, where c_w represents the winning code and c_l represents the losing code. For each question, we use the highest-scoring YES response and the highest-scoring NO response to form a preference pair, aiming to maximize the difference between them. See preference data examples in Section A.4.

$$\begin{aligned} D_{\text{DPO}} = \{(q_i, c_{ij}, c_{ik}) \mid\, & r_\phi(y = \text{YES} \mid t_{ij}), \\ & r_\phi(y = \text{NO} \mid t_{ik}), \\ & t_{ij} = \arg\max_{t_{ij} \in T_i} p_{r_\phi}(y = \text{YES} \mid t_{ij}), \\ & t_{ik} = \arg\max_{t_{ik} \in T_i} p_{r_\phi}(y = \text{NO} \mid t_{ik})\} \end{aligned} \tag{6}$$

4 Experiments

4.1 Data

We summarize the statistics of data use for self-improving in Table 3. Due to limited space, see statistics of evaluation benchmarks in Table 12.

Seed Data D_0: To generate the seed data for English, following previous work, we use GPT-4-0613 in an iterative fashion: we repeatedly sample the remaining questions that do not have correct code (i.e., the code execution results match the ground truth answer of the questions) for several iterations. We use questions from the training sets of GSM8K (7.5K) and MATH (7.5K) as the seed questions for imitation learning. For datasets such as GSM8K in which the answers are mostly single numbers, it is easier to compare answer and code execution results. After two iterations, we can annotate 98.5% of questions in GSM8K. For datasets such as MATH wherein the answers are diverse in formats, we simply keep the code that can be successfully executed without errors. For seed questions for Chinese, we randomly sample 20K from (1.13M in total) collected or purchased from educational web resources (Section 5) and follow the same procedure using GPT-4-0613 and self-distillation for code generation to construct the Chinese subset of D_0.

Value-Style D_1: We utilize the initial policy M_0 to generate code samples to questions in training sets of the word math problem datasets APE (200.5K) (Zhao et al., 2020) and CM (13.6K) (Qin et al., 2021), both collected from web resources. Since all the answers are one or two numerical values, for efficiency, we use heuristics to compare the code execution results with ground truth answers for validation. Following the same process used to construct D_0, we keep up to four valid code samples for each question.

Diverse-Format Data D_2: To increase the diversity of the training data, we further consider large-scale mathematical QA pairs (excluding those used for seed data) mentioned previously. For each question, we retain only one positive code and one negative code (if any exists) judged by the critic.

Critic Data: To build the training data for the critic model, we use M_0 to generate code samples for randomly sampled questions from D_2 and execute these code samples. We then prompt GPT-4-0613 with the input (question, code, code result, reference answer) following the template in Table 1. After filtering, we retain 16.8K training instances, of which 48.6% of are judged as YES.

To better understand this task, we analyze the reference answers for 50 instances. Only 14% of them are single numerical values, while 50% involve format conversion (e.g., syntax or structure) when the answers are expressions, equations, coordinates, sets, etc. Another difference between real-world data and well-formatted benchmarks is its inconsistency in the format of reference answers. Specifically, half of them are mixed with CoT-style (Wei et al., 2022) explanations and/or irrelevant contents such as tags and URLs. This makes it difficult to parse short-form answers for easier matching with a few patterns, as done for clean benchmarks (e.g., answer indicators "###" for GSM8K and "BOX" for MATH). For multiple-choice or multi-part questions (8% in total), we additionally require the question context for mapping option labels and their contents, as well as question decomposition. These observations reflect the diversity of question types in the web resources.

Data/Subset		QA Source	Size
D_0	zh	web	76K
	en	GSM8K, MATH	44K
D_1		APE, CM	211K
D_2	SFT	web	893K
	SFT(H)	web	273K
	DPO	web	465K

Table 3: Statistics of training data used in our three-stage paradigm (D_1 and D_2 are Chinese resources).

4.2 Implementation

We use the unified framework LLAMAFACTORY (Zheng et al., 2024) for efficient fine-tuning built upon DeepSpeed (ZeRO-3). Our experiments are conducted using 8XA100 40GB GPUs. We train LLMs with BF16 mixed-precision. The training for the self-improving paradigm explored in our experiments takes approximately 96 hours. With 80 workers in multi-processing mode on a CPU machine, we can execute about 9,003 codes per minute. Each model at each stage is trained for two epochs with a learning rate of 1e-6. We set the SFT loss coefficient (λ in Equation 7) to 1.0. The maximum sequence length is set to 1024, and the batch size is set to 64 for both SFT and DPO. We set λ_1 to 0.8 and λ_2 to 3.

As we focus on solving mathematical problems with code, we experiment with four LLMs to select backbone models — CodeLlama-7B-Python (Roziere et al., 2023),

$\text{Llama3}_{\text{instruct}}$[1] (AI@Meta, 2024), CodeQwen1.5-7B-Chat (Team, 2024), and Deepseek-Coder-7B-instruct-v1.5 (Daya Guo, 2024) — that demonstrate strong coding capabilities on code-related benchmarks. Due to limited computational resources, we use their 7B or 8B versions with their corresponding default templates and leave the model scaling up for future work. We primarily follow the evaluation scripts from previous studies (Liang et al., 2024) for Chinese benchmarks and FastEval[2] for English benchmarks GSM8K and MATH. We also make adjustments to these scripts, as our predicted answers are in code syntax. We use CodeLlama-7B-Python as the backbone LLM to train the code-based critic model for three epochs with the maximum sequence length 4096.

4.3 The Performance of the Initial Policy and Self-Improved LLMs

As shown in Table 4, $\text{DeepSeek}_{\text{code}}$ and $\text{Llama3}_{\text{instruct}}$ show superior average performance across math datasets in both Chinese (APE, CM, and CMATH (Wei et al., 2023)) and English (GSM8K and MATH). Therefore, we consider them as initial policy models (i.e., M_0) for later self-improving experiments. After two additional iterations on the unseen data D_1, and D_2 constructed with the help of our code-based critic model, the resulting models (i.e., M_2) consistently outperform M_0 by a large margin on Chinese benchmarks.

We observe that self-improving the initial policy model with **Chinese-only** data, D_1 and D_2, does not hurt the accuracy of M_2 on English tasks. In fact, it may be beneficial (e.g., $+1.5\%$ on both MATH and GSM8K datasets using $\text{DeepSeek}_{\text{code}}$). Conversely, adding English seed data (36.7% of D_0) consistently improves M_0's average performance on Chinese benchmarks (D_0 vs. $D_{0,\text{zh}}$ in Table 5). To some extent, we may interpret code as a universal language for solving mathematical problems across different languages. The language-specific parts are mainly in the code comments, which are relatively indirect for problem-solving via code execution. Thus, our paradigm may reduce the burden of preparing large-scale, language-specific math data for each language.

We list several general-purpose/math-specified multi-lingual/English LLMs of similar or much larger size for reference. Note that direct comparisons are challenging due to differences in architectures, pertaining corpora, alignment algorithms, and labeled data, though the performance of almost all these LLMs on GSM8K and MATH can be considered in-domain. Moreover, the evaluation scripts, originally designed for plain-text answers instead of code outputs, may cause an underestimation of our methods' performance on datasets such as MATH, where answers involve more expressions and structures beyond numerical values.

4.4 The Comparison of Different Choices of Data and Alignment Methods

Diversity: Based on the experimental results, given D_0 and D_1, we observe that two-stage SFT (first on D_0 for two epochs and then on D_1 for two epochs) under-performs one-stage SFT (over the concatenation of D_0 and D_1 for two epochs) (B vs. C in Table 6). However, incorporating D_2 using either strategy achieves similar performance (E vs. F in Table 6). One possible reason may be that the questions in D_1 are from two web-collected value-style benchmarks, resulting in less diversity compared with D_2, which has a broader range of question types (Section 4.1). Ensuring the diversity of data in each stage may help the model generalize better across various types of math questions, similar to the observations seen when training general-purpose LLMs (e.g., (Shen et al., 2023)).

Denoised SFT Data: As mentioned previously, we use the code-based critic model to construct SFT data. Since the process will inevitably introduce false positive data, we further consider several constraints for filtering (Equation 4 in Section 3.4). Experimental results show that we can achieve similar average accuracy using either $D_{2,\text{SFT,H}}$ or the $D_{2,\text{SFT}}$ (D vs. E in Table 6). However, $D_{2,\text{SFT,H}}$ is only 30.6% of the latter's size, indicating the usefulness of the filtering.

DPO or SFT: Based on a reasonably good model M_1 (trained with D_0 and D_1, such as C in Table 6), we can either self-improve it via SFT or DPO as described in Section 3.4. We compare using the positive (question, code) pairs in the DPO data for another round of SFT, which results in a 1.8% drop in accuracy on downstream tasks (G vs. I in Table 6). Since we do not impose strict constraints on the positive data in DPO, $D_{2,\text{DPO, positive}}$ is 1.7 times the size of $D_{2,\text{SFT,H}}$. Still, using the filtered SFT data $D_{2,\text{SFT,H}}$ achieves slightly better performance (F vs. G), showing the effectiveness of filtering.

DPO with SFT: Our experiments indicate that

[1]We observe little performance difference between $\text{Llama3}_{\text{instruct}}$ and $\text{Llama3}_{\text{base}}$ when both are fine-tuned on D_0.
[2]github.com/FastEval/FastEval/.

Model	Size (B)	CM	APE	CMATH	GSM8K	MATH
GPT-4-1106-Preview$^{\otimes}$	–	–	84.2	89.3	93.6	53.6
Qwen-Chat (Bai et al., 2023)$^{\otimes}$	72	–	77.1	88.1	76.4	31.8
ChatGLM-Math (Xu et al., 2024)$^{\otimes}$	32	–	89.4	85.6	82.6	40.6
Skywork-Math (Yang et al., 2023)$^{\otimes}$	13	–	74.4	77.3	72.3	17.0
Math-InternLM2 (Team, 2023)$^{\otimes}$	20	–	75.2	78.5	82.6	37.7
MathCoder (Wang et al., 2023)	34	–	–	–	81.7	45.2
	7	–	–	–	67.8	30.2
ToRA (Gou et al., 2024)	70	–	–	–	84.3	49.7
	7	–	–	–	72.6	44.6
MinT (Liang et al., 2024)	7	77.6	76.0	–	40.8	–
Initial Model Baselines (M_0)						
CodeLlama	7	77.7	78.0	84.5	69.7	37.6
QWEN$_{\text{code}}$	7	81.9	81.5	86.0	71.9	41.4
DeepSeek$_{\text{code}}$	7	82.7	81.2	87.0	77.4	44.4
Llama3$_{\text{instruct}}$	8	83.3	83.2	87.2	76.8	41.8
Self-Improving (SI) (M_2)						
SI(DeepSeek$_{\text{code}}$)	7	87.3 (+4.6)	85.9 (+4.7)	91.2 (+4.2)	78.9 (+1.5)	45.9 (+1.5)
SI(Llama3$_{\text{instruct}}$)	8	89.0 (+5.7)	86.8 (+3.6)	90.8 (+3.6)	80.5 (+3.7)	41.9 (+0.1)

Table 4: Accuracy across the development sets of math datasets for Chinese ($\otimes$: without using tools).

Model	Training Procedure	Data	CM	APE	CMATH	GSM8K	MATH
DeepSeek$_{\text{code}}$	SFT	$D_{0,\text{en}}$	–	–	–	74.6	43.8
	SFT	$D_{0,\text{zh}}$	81.0	82.4	86.8	–	–
	SFT	D_0	82.7	81.2	87.0	77.4	44.4
	SFT	$D_0 + D_1$	87.0	84.3	88.0	77.6	44.6
	SFT → DPO	$D_0 + D_1$; D_2	**87.3**	**85.9**	**91.2**	**78.9**	**45.9**
Llama3$_{\text{instruct}}$	SFT	$D_{0,\text{en}}$	–	–	–	75.1	37.2
	SFT	$D_{0,\text{zh}}$	82.5	83.3	85.5	–	–
	SFT	D_0	83.3	83.2	87.2	76.8	41.8
	SFT	$D_0 + D_1$	87.6	85.0	89.0	76.6	41.8
	SFT → DPO	$D_0 + D_1$; D_2	**89.0**	**86.8**	**90.8**	**80.5**	**41.9**

Table 5: The self-improving accuracy in different stages on the development sets of different datasets. The best open-sourced performance for each backbone model is highlighted in bold.

ID	Alignment	Data	ACC
A	SFT	D_0	74.4
B	SFT → SFT	D_0 ; D_1	75.4
C	SFT	$D_0 + D_1$	76.0
D	SFT	$D_0 + D_1 + D_{2,\text{SFT}}$	76.1
E	SFT	$D_0 + D_1 + D_{2,\text{SFT,H}}$	76.1
F	SFT → SFT	$D_0 + D_1$; $D_{2,\text{SFT,H}}$	76.2
G	SFT → SFT	$D_0 + D_1$; $D_{2,\text{DPO, positive}}$	76.0
H	SFT → ORPO	$D_0 + D_1$; $D_{2,\text{DPO}}$	77.0
I	SFT → DPO	$D_0 + D_1$; $D_{2,\text{DPO}}$	77.8

Table 6: The self-improving averaged accuracy (ACC) of Llama3$_{\text{instruct}}$ on the development sets of different datasets with various training strategies and data.

DPO training is relatively insensitive to the weight (λ in Equation 7) of the SFT loss. We tested with $\lambda = 1.0$ and $\lambda = 2.0$, both of which resulted in similarly good performance (77.8%). However, as shown in Table 7, removing the SFT loss (i.e., $\lambda = 0$) from DPO training leads to a dramatic increase in response length, especially for Chinese tasks such as CMATH, and yields worse results than the reference policy model (C in Table 6). This observation aligns with discussions on length exploitation issue of the original DPO loss (Park et al., 2024). One possible reason for the length control achieved by adding the SFT loss could be that the positive responses used for the SFT loss are generated by the reference policy model. By setting a larger weight to SFT, we control the deviation from the reference policy, which alleviates a substantial increase in response length. We also experiment with using ORPO (Hong et al., 2024), which removes the need for a reference model and jointly trains with the SFT loss. However, this method is not as effective as jointly training DPO and SFT in our experiments (H vs. I in Table 6). Detailed results can be found in Section A.3.

4.5 Using the Critic Model as An Evaluator

We have shown the effectiveness of using the critic model to construct SFT and preference data. All scores are computed by comparing predictions with ground truth answers, using heuristics-based exact match (EM) following previous studies for fair comparisons. To explore the potential of using the critic model as a complementary evaluator, we examine the correlation between the two evaluation

λ	GSM8K			CMATH		
	ACC	L	$\frac{L}{L_0}$	ACC	L	$\frac{L}{L_0}$
reference model						
-	76.6	323	1.0	89.0	136	1.0
0.0	73.4	1834	5.7	57.5	3160	23.2
0.5	78.8	532	1.6	90.7	201	1.5
1.0	80.5	352	1.1	90.8	136	1.0
1.5	79.0	328	1.0	90.7	135	1.0
2.0	79.8	326	1.0	90.7	134	1.0

Table 7: The impact of the weight of the SFT loss in DPO training on the average accuracy and average response length in words evaluated on GSM8K and CMATH (L_0: response length of the reference policy).

Dataset	ACC_{EM}	ACC_{critic}	$Correlation_{Kendall}$
CM	89.0	84.6	0.66
APE	86.8	86.5	0.76
CMATH	90.8	91.8	0.77
GSM8K	80.5	80.6	0.97
MATH	41.9	48.2	0.79
average	77.8	78.3	0.79

Table 8: Correlation of two evaluation methods: heuristics-based EM and the critic model.

methods on the previously used benchmarks. We use the original ground truth answers (final-step answers if answers are COT-style) (e.g., "3750", "[12, 18]", and "`\\frac{1}{2}`") in these benchmarks. Since all scores are either 0 (NO) or 1 (YES), we report the Kendall's τ between the two methods. As shown in Table 8, **there is a very strong correlation** (0.79) (compared to the very-strong-cutoff value 0.71 and strong-cutoff value 0.49 (Schober et al., 2018)) between the scores computed by the two evaluators. The strong associations in English tasks are surprising, given that the critic model is trained on Chinese-only data. This may be due to (i) the backbone model being a well-instructed model focused on English, and (ii) comparing answers to mathematical questions relying less on language-specific knowledge.

4.6 The Performance of Self-Improved LLMs on More Out-of-Domain Tasks

Considering the above results in Section 4.5, we are now more confident in using the critic model to evaluate models' performance on additional out-of-domain benchmarks, without the need to write extensive heuristics for different tasks. Besides CMATH, we evaluate the out-of-domain performance of our models using MathBench (Liu et al., 2024), a newly released benchmark supporting evaluation in both Chinese and English. The questions in MathBench span various educational stages, from primary school to college levels. We report scores on its two subsets: MathBench-A, which evaluates practical problem-solving skills, and MathBench-T, which assesses theoretical understanding. As shown in Table 9, the self-improved models demonstrate substantial gains on both subsets, with an accuracy improvement of at least 4.2%. On both subsets, the self-improved model consistently outperforms the initial one across all educational levels and subjects with notable improvements particularly in middle school tasks and English theoretical tasks. See detailed performance in Table 10 and Table 11 (Section A.1). Note that we provide the scores of other models for reference, as they are judged by a different scorer.

Compared to practical application questions, it seems that using CoT, LLMs are much better at handling theoretical knowledge questions. In contrast, solving all questions via coding shows balanced and reasonable performance. This shows the advantage of using tools to aid in computation, but also indicates the limitations of relying solely on code to address questions that may not require actual computation. To date, it remains an open question how to (or whether to) use code for assist advanced theoretical reasoning (Liu et al., 2024).

model	Subset-A	Subset-T	$ACC_{average}$
GPT-4-0125-Preview$^{\otimes}$	$58.8^{\dagger}$	$78.4^{\dagger}$	$68.6^{\dagger}$
GLM4$^{\otimes}$	$51.3^{\dagger}$	$73.1^{\dagger}$	$62.2^{\dagger}$
Qwen-Chat-72B$^{\otimes}$	$49.7^{\dagger}$	$77.2^{\dagger}$	$63.5^{\dagger}$
Math-InternLM2-20B$^{\otimes}$	$41.9^{\dagger}$	$64.3^{\dagger}$	$53.1^{\dagger}$
$Llama3_{instruct}$-8B$^{\otimes}$	$36.7^{\dagger}$	$52.1^{\dagger}$	$44.4^{\dagger}$
$SI(Llama3_{instruct})_0$-8B	$62.5^{\star}$	$57.9^{\star}$	$60.2^{\star}$
$SI(Llama3_{instruct})_2$-8B	$66.7^{\star}$	$62.6^{\star}$	$64.6^{\star}$

Table 9: OOD accuracy on the MathBench dataset ($\star$: scored by the critic model; $\dagger$: based on the numbers reported by Liu et al. (2024); $\otimes$: without using tools).

5 Conclusions and Future Work

We introduce a novel paradigm for self-improving LLMs, which employs a code-based critic model to guide stages such as the creation and filtering of question-code data as well as complementary evaluation. We also investigate various alignment algorithms using self-generated instruction/preference data for further improvement. Results show the effectiveness of self-improving LLMs with this proposed paradigm. Future research includes studying post-training on code-only data to enhance the computational capabilities of LLMs and self-improvement of the critic model.

Limitations

Language Diversity of Resources: in this paper, we focus on large-scale question-answer pairs for Chinese, and accordingly, our critic model used for guiding self-improvement is trained on Chinese data. While considering resources in other languages such as English could enhance the the generalizability of LLMs, it would require extensive human efforts for data collection and cleaning, which is beyond the scope of this work. On the other hand, based on our experiments, since the backbone LLMs are pre-trained and aligned on multi-lingual data and our seed data includes English instruction data, the initial policy already exhibits reasonable performance on in-domain (Section 4.3) and out-domain benchmarks (Section 4.6). Self-improving this initial policy model on Chinese data may even improve its performance on English tasks. Finally, experiments show that the critic model is as effective at rating English responses as rating Chinese ones (Section 4.5).

Copyright of Resources: The large-scale question-answer pairs (excluding APE and CM) are either collected or purchased from educational websites. We will not release the full-scale resources. Instead, we will provide question-answer samples, along with all scripts, seed data in English, initial and self-improved policy models, and the critic model to facilitate future studies.

The Usage of Code: Code can be used either directly (Chen et al., 2022; Gao et al., 2023b) or interactively (Wang et al., 2023) during problem-solving. The latter approaches such as ToRA (Gou et al., 2024) and MathCoder (Wang et al., 2023) jointly solve problems using CoT explanation (Wei et al., 2022) and code. One advantage of these interactive methods over code-only methods is that the final step of their solution is usually written in CoT, allowing the easy use of existing scripts designed for CoT-style benchmarks for evaluation. However, the role of using tools multiple times to address a single math problem is unclear based on the performance difference of interactive methods (Table 4). For example, ToRA needs 1.02 tool interaction rounds per question while MathCoder requires 2.05. This work focuses on the direct usage of code as a case study to avoid multi-step inference, and leave the interactive setting for future studies. In addition, as discussed in Section 4.6,

LLM Scalings: Due to limited computational resources, our experiments focus on 7/8-B LLMs. Generally, improving the math reasoning abilities of relatively small LLMs requires a large amount of training data and knowledge distillation (Li et al., 2024; Shao et al., 2024), which may not be necessary for larger LLMs.

References

AI@Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date. https://ai.meta.com/blog/meta-llama-3/.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. 2023. Ultrafeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*.

Dejian Yang Zhenda Xie Kai Dong Wentao Zhang Guanting Chen Xiao Bi Y. Wu Y.K. Li Fuli Luo Yingfei Xiong Wenfeng Liang Daya Guo, Qihao Zhu. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

Clémentine Fourrier, Nathan Habib, Thomas Wolf, and Lewis Tunstall. 2023. Lighteval: A lightweight framework for llm evaluation.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023a. A framework for few-shot language model evaluation.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model. *arXiv preprint arXiv:2403.07691*, 2(4):5.

Chi Hu, Yimin Hu, Hang Cao, Tong Xiao, and Jingbo Zhu. 2024. Teaching language models to self-improve by learning from language feedback. *arXiv e-prints*, pages arXiv–2406.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. 2024. Common 7b language models already possess strong math capabilities. *arXiv preprint arXiv:2403.04706*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023. Self-alignment with instruction backtranslation. *arXiv preprint arXiv:2308.06259*.

Zhenwen Liang, Dian Yu, Xiaoman Pan, Wenlin Yao, Qingkai Zeng, Xiangliang Zhang, and Dong Yu. 2024. MinT: Boosting generalization in mathematical reasoning via multi-view fine-tuning. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 11307–11318, Torino, Italia. ELRA and ICCL.

Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark. *Preprint*, arXiv:2405.12209.

Jianqiao Lu, Wanjun Zhong, Wenyong Huang, Yufei Wang, Fei Mi, Baojun Wang, Weichao Wang, Lifeng Shang, and Qun Liu. 2023. Self: Language-driven self-evolution for large language model. *arXiv preprint arXiv:2310.00533*.

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

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Ryan Park, Rafael Rafailov, Stefano Ermon, and Chelsea Finn. 2024. Disentangling length from quality in direct preference optimization. *arXiv preprint arXiv:2403.19159*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Jinghui Qin, Xiaodan Liang, Yining Hong, Jianheng Tang, and Liang Lin. 2021. Neural-symbolic solver for math word problems with auxiliary tasks. In *ACL*, pages 5870–5881.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation coefficients: appropriate use and interpretation. *Anesthesia & analgesia*, 126(5):1763–1768.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Joel Hestness, Natalia Vassilieva, Daria Soboleva, and Eric Xing. 2023. Slimpajama-dc: Understanding data combinations for llm training. *arXiv preprint arXiv:2309.10818*.

InternLM Team. 2023. Internlm: A multilingual language model with progressively enhanced capabilities.

Qwen Team. 2024. Code with codeqwen1.5.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. *arXiv preprint arXiv:2310.03731*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. 2023. Cmath: can your language model pass chinese elementary school math test? *arXiv preprint arXiv:2306.16636*.

Martin Weyssow, Aton Kamanda, and Houari Sahraoui. 2024. Codeultrafeedback: An llm-as-a-judge dataset for aligning large language models to coding preferences. *Preprint*, arXiv:2403.09032.

Yifan Xu, Xiao Liu, Xinghan Liu, Zhenyu Hou, Yueyan Li, Xiaohan Zhang, Zihan Wang, Aohan Zeng, Zhengxiao Du, Wenyi Zhao, et al. 2024. Chatglm-math: Improving math problem-solving in large language models with a self-critique pipeline. *arXiv preprint arXiv:2404.02893*.

Liu Yang, Haihua Yang, Wenjun Cheng, Lei Lin, Chenxia Li, Yifu Chen, Lunan Liu, Jianfei Pan, Tianwen Wei, Biye Li, et al. 2023. Skymath: Technical report. *arXiv preprint arXiv:2310.16713*.

Xiao Yu, Baolin Peng, Michel Galley, Jianfeng Gao, and Zhou Yu. 2023. Teaching language models to self-improve through interactive demonstrations. *arXiv preprint arXiv:2310.13522*.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*.

Wei Zhao, Mingyue Shang, Yang Liu, Liang Wang, and Jingming Liu. 2020. Ape210k: A large-scale and template-rich dataset of math word problems. *arXiv preprint arXiv:2009.11506*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, et al. 2023. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*.

A Appendices

A.1 Sub-Type Performance on MathBench

The data presented in the tables clearly shows the advantage of SI(Llama3$_{\text{instruct}}$)2 over SI(Llama3instruct)0 across various educational levels and subjects. For both the MathBench-A and MathBench-T datasets, SI(Llama3instruct)2 consistently outperforms SI(Llama3instruct)0. In the MathBench-A dataset, improvements are seen in all levels from Primary to College, with notable jumps in Middle and High school levels (6.7% and 7.0% improvement, respectively). Similarly, the MathBench-T dataset shows improvement across all levels, particularly in the Middle school and English categories, which demonstrate 8.1% and 10.5% increases, respectively. These results indicate that SI(Llama3instruct)$_2$ provides enhanced accuracy in out-of-distribution scenarios, making it a more reliable choice for varied educational contexts.

Level	SI(Llama3$_{\text{instruct}}$)$_0$	SI(Llama3$_{\text{instruct}}$)$_2$
Arith	98.0	**99.0**
Primary	75.7	**80.7**
Middle	56.3	**63.0**
High	50.3	**57.3**
College	32.0	**33.3**
Chinese	56.8	**63.6**
English	66.2	**68.8**

Table 10: Fine-grained OOD accuracy on the MathBench-A dataset scored by the critic model (Arith: arithmetic).

Level	SI(Llama3$_{\text{instruct}}$)$_0$	SI(Llama3$_{\text{instruct}}$)$_2$
Arith	–	–
Primary	66.6	**67.5**
Middle	60.1	**68.2**
High	59.1	**60.6**
College	50.2	**57.9**
Chinese	62.7	**63.6**
English	50.6	**61.1**

Table 11: Fine-grained OOD accuracy on the MathBench-T dataset scored by the critic model (Arith: arithmetic).

A.2 Data Statistics

A.3 Other Alignment Algorithms

As shown in Table 13, DPO demonstrates superior performance compared to ORPO, both with the SFT loss. We leave the exploration of more length-regularized alignment algorithms and the role of the reference policy model in preference optimization to future studies.

$$L_{\text{ORPO}}(\pi_{\theta_{k+1}}) = -\,\lambda \cdot \log \sigma \left(\log \frac{\pi_{\theta_{k+1}}(y_w \mid x)}{1 - \pi_{\theta_{k+1}}(y_w \mid x)} - \log \frac{\pi_{\theta_{k+1}}(y_l \mid x)}{1 - \pi_{\theta_{k+1}}(y_l \mid x)} \right) - \log \pi_{\theta_{k+1}}(y_w \mid x) \tag{7}$$

A.4 Self-Generated Preference Data

Existing preference datasets (Cui et al., 2023; Weyssow et al., 2024) do not include many this kind of inter-disciplinary annotations. We use the current policy to generate preference data. Examples are provided in Table 14 and Table 15.

Dataset	Language	Answer Type	Level	Training	Validation
APE (Zhao et al., 2020)	zh	numerical value	elementary	200,488	5,000
CM (Qin et al., 2021)	zh	numerical value(s)	grades 6—12	13,628	1,703
CMATH (Wei et al., 2023)	zh	numerical value	elementary	–	600
MathBench (Liu et al., 2024)	en, zh	mixed	from primary to college	–	3,709
MATH (Hendrycks et al., 2021)	en	mixed	college	7,500	5,000
GSM8K (Cobbe et al., 2021)	en	numerical value	elementary	7,473	1,319

Table 12: Statistics of evaluation benchmarks. Note that in our experiments, we do not use any rationale in these dataset as we focus on solving problems via coding. We only use the questions and short-form answers from the training set of MATH and GSM8K for constructing the seed data, and we use the questions and short-form answer from the training set of APE and CM for constructing the data for self-improvement.

Model	Alignment	Data	CM	APE	CMATH	GSM8K	MATH	$\text{ACC}_{\text{average}}$
$\text{DeepSeek}_{\text{code}}$	SFT	$D_0 + D_1$	87.0	84.3	88.0	77.6	44.6	76.3
	SFT → ORPO	$D_0 + D_1; D_2$	**87.7**	85.5	**91.2**	76.5	44.5	77.1
	SFT → DPO	$D_0 + D_1; D_2$	87.3	**85.9**	**91.2**	**78.9**	**45.9**	77.8
$\text{Llama3}_{\text{instruct}}$	SFT	$D_0 + D_1$	87.6	85.0	89.0	76.6	41.8	76.0
	SFT → ORPO	$D_0 + D_1; D_2$	88.0	86.4	91.8	76.4	42.1	77.0
	SFT → DPO	$D_0 + D_1; D_2$	**89.0**	**86.8**	**90.8**	**80.5**	**41.9**	77.8

Table 13: The self-improving performance in different stages on the development sets of different datasets. The best open-sourced performance for each backbone model is highlighted in bold.

question The perimeter of an isosceles triangle is 20, and the ratio of the lengths of the two equal sides is 1:2. What is the length of the base? A.10 B.4 C.4 or 10 D.5 or 8

reference answer B

winning code (w)

```
from sympy import symbols, Eq, solve

# Define the unknown
a, b = symbols('a b')

# Establish the equation
# The hypotenuse is twice the base
equation1 = Eq(a, 2*b)

# The perimeter is 20
equation2 = Eq(2*a + b, 20)

# Solve the equation
solution = solve((equation1, equation2), (a, b), dict=True)
solution
```

code result of w [a: 8, b: 4]

losing code (l)

```
from sympy import symbols, Eq, solve

# Define the unknown variable
a = symbols('a')

# Establish the equation
equation = Eq(2 * a + 2 * (2 * a), 20)

# Solve the equation
solution = solve(equation, dict=True)
solution
```

code result of l [a: 10/3]

Table 14: Examples of self-generated data (I). Texts have been translated into English.

question Given that a and b are opposites, c and d are reciprocals, and the square of x equals 4, try to find the value of x$^2 - (a+b+cd)x + (a+b)^2009 + (-cd)^2008$.

reference answer 3 or 7

winning code (w)

```
# Based on the analysis, first solve for the two possible
    ↪ values of x
# Since x squared equals 4, solve for x
x1 = 2
x2 = -2

# Next, calculate the value of the expression based on the
    ↪ value of x
# After simplification, the expression is x^2 - x + 1
# Calculate the value of the expression for x1 and x2
result1 = x1**2 - x1 + 1
result2 = x2**2 - x2 + 1

# Return results
{'x1␣result': result1, 'x2␣result': result2}
```

code result of w {‘x1’: 3, ‘x2’: 7}

losing code (l)

```
# Based on analysis, we have obtained the simplified form of
    ↪ the expression

# Define variables
# The value of a + b, since a and b are opposites, take a = 1,
    ↪ then b = -1
a_b_sum = 1
# The value of cd, since c and d are reciprocals, take c = 1,
    ↪ then d = 1
cd_product = 1
# The square of x
x_squared = 4

# Calculate the value of the expression
# x^2 - (a+b+cd)x + (a+b)^2009 + (-cd)^2008
result = x_squared - (a_b_sum + cd_product) * 2 + a_b_sum**2009
    ↪  + (-cd_product)**2008

# Return result
{"result": result}
```

code result of l {2}

Table 15: Examples of self-generated data (II). Texts have been translated into English.