Critic-CoT: Boosting the Reasoning Abilities of Large Language Model via Chain-of-Thought Critic

Xin Zheng[1,2*] **Jie Lou**[3✉] **Boxi Cao** [1,2] **Xueru Wen** [1,2] **Yuqiu Ji**[3]
Hongyu Lin[1✉] **Yaojie Lu**[1] **Xianpei Han**[1] **Debing Zhang**[3] **Le Sun**[1]
[1] Chinese Information Processing Laboratory, Institute of Software,
Chinese Academy of Sciences [2] University of Chinese Academy of Sciences
[3] Xiaohongshu Inc
{zhengxin2020,boxi2020,wenxueru2022}@iscas.ac.cn
{hongyu,luyaojie,xianpei,sunle}@iscas.ac.cn
{yinyue2,dengyang}@xiaohongshu.com

Abstract

Self-critic has become a crucial mechanism for enhancing the reasoning performance of LLMs. However, current approaches mainly involve basic prompts for intuitive instance-level feedback, which resembles System-1 processes and limits the reasoning capabilities. Moreover, there is a lack of in-depth investigations into the relationship between LLM's ability to criticize and its task-solving performance. To address these issues, we propose Critic-CoT, a novel framework that pushes LLMs toward System-2-like critic capability. Through a step-wise CoT reasoning paradigm and the automatic construction of weak-supervision data without human annotation, Critic-CoT enables LLMs to engage in slow, analytic self-critique and refinement, thereby improving their reasoning abilities. Experiments on GSM8K and MATH and out-of-domain evaluation demonstrate that our enhanced model significantly boosts task-solving performance by filtering out invalid solutions or iterative refinement. Furthermore, we investigate the intrinsic correlation between critique and task-solving abilities within LLMs, discovering that these abilities can mutually reinforce each other rather than conflict. [1]

1 Introduction

Enhancing the reasoning abilities of large language models is essential for creating more intelligent and reliable AI systems, which has drawn extensive attention from researchers (Chollet, 2019; Bubeck et al., 2023; Morris et al., 2024). From a cognitive perspective, the procedure of human reasoning involves constant reflection and revision (Hegel et al., 1991; Kierkegaard, 1989; Popper, 1934), which has inspired increasing focus on integrating self-critic mechanisms in the reasoning process of large-scale models (Kim et al., 2023; Shinn et al., 2023; Madaan et al., 2023). This involves iteratively allowing the model to generate feedback on its own responses and then refining its reasoning based on the feedback. Compared with traditional critic methods that depend on feedback from external sources (Saunders et al., 2022; McAleese et al., 2024), self-critic relies solely on the model's internal capabilities, thus reducing the high cost of additional human annotation, and serving as a promising potential solution to scalable oversight (Leike et al., 2018; Burns et al., 2023; Cao et al., 2024).

However, current studies primarily focus on utilizing LLMs' critique abilities to enhance their performance. Yet, relatively little attention has been given to the investigation and development of the critique ability itself. Firstly, existing critique methods are often overly simplistic, typically relying on a basic prompt to directly point out the error, without stepwise Chain-of-Thought examination or training procedure, which leads to relatively poor self-critic accuracy (Luo et al., 2023; West et al., 2024). Specifically, proposing a valid critique is a complicated task that requires a thorough understanding of statements and precise negativity. However, current LLMs are normally not explicitly trained for critic capability. Therefore, these simple approaches usually tend to "criticize" like System-1 (fast thinking mode), which is more intuitive and likely to make mistakes, rather than more rigorous and deliberate System-2 (slow thinking mode) (Kahneman, 2011; Yu et al., 2024), while shifting LLMs from System-1 toward System-2, which is to perform systematical analysis that fully utilizes the advantage of Chain-of-Thought reasoning, emerges as a promising approach for improving the reasoning capability (OpenAI, 2024). This limitation diminishes the effectiveness of self-critic and, further, self-correct (Huang et al., 2024). Secondly, the capabilities of task-solving and self-critic are both dependent on the model's inherent knowledge,

[*] This work was done when Xin Zheng interned at Xiaohongshu.

[1] Our code and data are available at https://github.com/AlignRM/Critic-CoT

Findings of the Association for Computational Linguistics: ACL 2025, pages 1768–1806
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

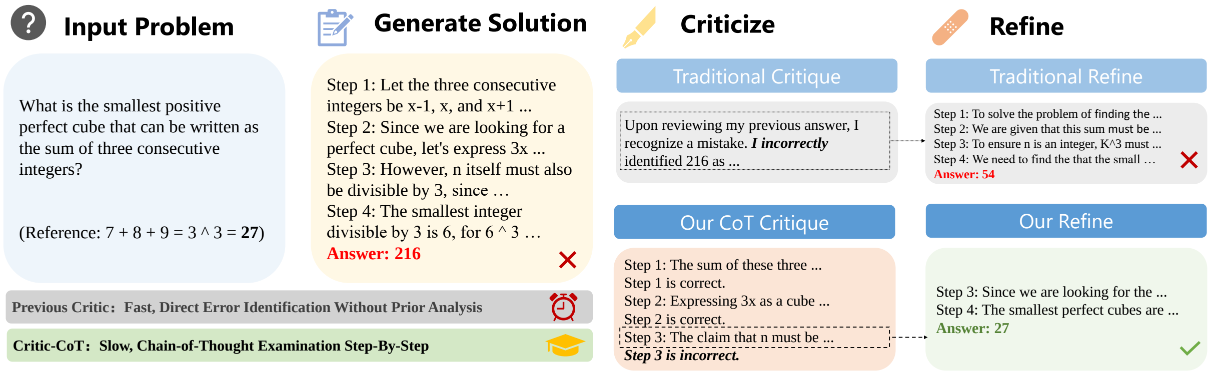


Figure 1: Illustration of Critic-CoT: Previous instance-level critic methods attempt to identify errors directly without any prior analysis, and restart from the beginning during refinement. In contrast, our proposed Critic-CoT framework performs a step-wise examination using the Chain-of-Thought approach. When refining, rather than starting from scratch, our method makes the correction from the specific error step with the help of the corresponding critique.

while there is currently a lack of in-depth exploration regarding the correlation between these two capabilities within LLMs. In that case, it's challenging to balance the task-solving and the self-critic capabilities of the model within the self-critic framework, which poses a significant obstacle to the subsequent development.

To this end, this paper is devoted to diving into the following critical research questions:

- How can we enhance a model's critique ability, pushing it toward System 2 reasoning?
- What is the relationship between a model's critique ability and its task-solving capability?

To answer the above questions, as shown in Figure 1, we propose Critic-CoT, a novel framework designed to enhance LLMs' reasoning abilities. Through step-wise Chain-of-Thought critique format and automated data construction through weak supervision, our method is able to strengthen System-2-like critic ability, without the intensive cost of human annotation. Specifically, during training, we let LLMs criticize and refine their solutions in a complete CoT way, and collect successful pairs that convert wrong solutions into correct ones, or affirm the validity of original right solutions. After supervised fine-tuning on the obtained step-wise critic-refine data, we enable the target LLM to analyze and criticize each step of its generated reasoning procedure, so that it can filter out wrong attempts and preserve the correct ones with greater precision. During inference, to leverage the model's abilities of CoT-critique and refinement, we employ two strategies: 1) majority vote filtering involves using the critic model to evaluate multiple generated solutions and filter out those incorrect; and 2) iterative refinement, on the other hand, involves repeatedly critiquing and refining a solution until no further error is detected.

Through a series of experiments on the in-domain dataset of GSM8K and MATH, together with out-of-domain evaluation on StrategyQA, AGIEval and HumanEval, we find that our trained critic model can fairly distinguish incorrect solutions from correct ones, and improve the reasoning accuracy via iterative refinement or critic filtering. These results demonstrate the helpfulness and effectiveness of our proposed method. Additionally, we observed that our critic model already exhibits noticeable performance improvements in task-solving, even in the absence of additional critique steps during the decoding phase. Such findings reveal that strengthening the ability to critique and refinement would not compromise the task-solving performance, but improve it. This also suggests the presence of an intrinsic mechanism by which critique ability and task-solving capability mutually reinforce one another.

Our main contributions are as follows:

- We propose Critic-CoT, which pushes the critic paradigm of LLMs from System-1-like incentive "thinking" toward System-2-like deliberate "reasoning".
- Through experiments, we find that Critic-CoT can effectively teach the model to criticize and refine its own output step by step, thus noticeably improving the reasoning performance.
- Moreover, we find that for LLMs, the ability of critique and refinement could mutually re-

inforce, which may shed light on designing more advanced self-critic framework designs in future work.

2 Related Works

With the development of LLMs, self-critic has emerged as a widely-adopted mechanism for reasoning, code generation, computer control, hallucination mitigation, retrieval-augmented generation and other tasks (Kim et al., 2023; Shinn et al., 2023; Madaan et al., 2023; Ji et al., 2023; Asai et al., 2024). However, typical self-critic approaches like Reflexion (Shinn et al., 2023), only utilize the LLMs' existing critique ability without further enhancement, and require the gold environment signal to iteratively generate critiques and make refinements. As long as external feedback is not available, off-the-shelf LLMs cannot perform intrinsic self-correct effectively due to limited critique and refinement abilities (Huang et al., 2024; Luo et al., 2023; Zeng et al., 2023). Later, several works are proposed to improve self-reflection via a carefully designed prompting pipeline on frozen LLMs, and no active training process is involved (Zhang et al., 2024b; Yan et al., 2024; Wu et al., 2024). Concurrently, Zhang et al. (2024a) trained a generative reward model on the outcome level rather than the process level, and did not incorporate refinement into the schema. Therefore, given the limited critic ability of current LLMs, how to train a robust and applicable critic model, which conducts detailed Chain-of-Thougt analysis in a step-wise systematic manner, and thus shifts from System-1 reasoning toward more deliberate System-2 reasoning (Kahneman, 2011), is worth investigating.

From the perspective of recursive reward modeling (Leike et al., 2018; Saunders et al., 2022) and scalable oversight (Burns et al., 2023), McAleese et al. (2024) recently trained "CriticGPT" to assist human labelers, which aims to improve the ability of human rather than the base model, i.e., improve the overall recall of error detection, rather than precision. While in this paper, we try to improve the reasoning ability of LLM without costly human annotation.

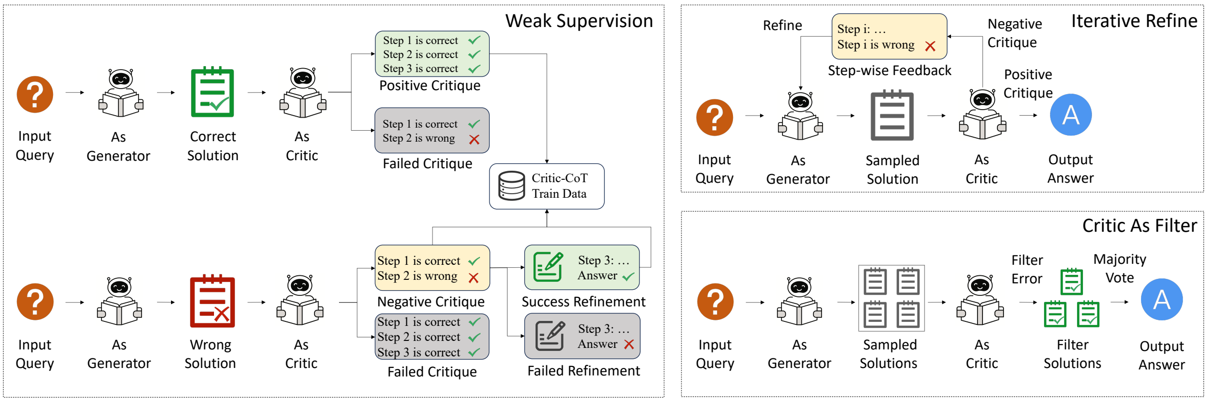
3 Method

Critic CoT is to equip LLMs with the ability to criticize and refine themselves step-by-step. As shown in Figure 2, it consists of two modules, including automated data construction via weak-supervision and self-check at inference-time. The weak-supervision principles are in Section 3.1, followed by the training process in Section 3.1, and the inference strategies in 3.3.

3.1 Chain-of-Thought Critique

In this work, we utilize a step-wise chain-of-thought critique, which makes the critique-refine process both controllable and formalizable, thereby facilitating the collection of weak supervision data. Formally, given the question Q and the corresponding gold answer Ans, we have the n-step attempt $Att = [s_1, ..., s_n]$ with predicted answer $Pred$ sampled by generator G. The corresponding critique Cri then can be represented as $L = [l_1, ..., l_n]$, where the step label $l_i = +1$ indicates that step i is predicted to be correct, and $l_i = -1$ to be incorrect. Then the refinement $Att' = [s'_i, ..., s'_{n'}]$ is start from the first incorrect step i with new answer $Pred'$. We automatically annotate the process labels as follows:

- $Pred \neq Ans, -1 \notin L$: The attempt is wrong, yet the critique did not discover any error step. Thus the critique itself is problematic, and we need to sample another critique.
- $Pred \neq Ans, -1 \in L, Pred' \neq Ans$: The attempt is wrong, and the critique found an error, but still, the refinement is not correct. There could be two cases for this situation: (1) the refinement is unsuccessful; (2) the critique did not detect an earlier mistake. We simply sample another critique and corresponding refinement for this situation.
- $Pred \neq Ans, -1 \in L, Pred' = Ans$: Not only did the critique point out the error, but also the refinement reached the correct answer. We then believe the critique is valid, and collect the critique data instance $C = (Q, Att, Cri)$ and the refinement data $R = (Q, Att, Cri_{-1}, Att')$, where Cri_{-1} is the critique of last step, since explaining why previous steps are correct may not be helpful.
- $Pred = Ans, -1 \notin L$: The attempt is correct, and the critique believes it is correct. So we can collect the positive critique data instance $C = (Q, Att, Cri)$.
- $Pred = Ans, -1 \in L$: The attempt reached the correct answer, yet the critique found an



(a) Auto-Train: Training data construction of Critic-CoT

(b) Self-Check: Inference process of Iterative Refine and Critic As Filter

Figure 2: The Process of Critic-CoT during training (a) and inference (b). For training, we collect the critic-refine data on the generator's samples via weak supervision (Section 3.1). Through fine-tuning, we enable the target model to criticize and refine its own reasoning process. Then, during inference, we can leverage the capabilities via Iterative Refine or Critic As Filter (Section 3.3).

error. Then, the critique could be wrong, and we need to sample another critique.

3.2 Auto Train: Two-Stage Training

To enable the model to acquire self-critiquing and refining capabilities, we first need to provide it with basic critiquing abilities, followed by self-critique for further enhancement. The overall training procedure is divided into two stages.

Stage 1 In the first step, we collect high-quality critique data to provide the model's basic critiquing ability. Specifically, we first sample both positive and negative solutions from a representative instruction-following model $\mathcal{M}_G$ on the dataset D. Then, we utilize LLMs like GPT4-Turbo to serve as critic model M_C. For each generated attempt Att, the critic model will retry at most k times to produce a valid critique until it reaches one of the weak supervision constraints. This will form the critic-refine dataset $D_1 = \{(Q, Att, Cri)\} \bigcup \{(Q, Att, Cri_{-1}, Att')\}$ for fine-tuning the initial model $\mathcal{M}_0$ into the critic model $\mathcal{M}_1$. Note that in this process, we actually distill `Pass1@N` of the teacher model $\mathcal{M}_C$ into `Top1@N` of the student model. So, the theoretical upper bound of the student model is not necessarily limited by the teacher model's performance.

Stage 2 In the second step, we leverage the model's self-critique to enhance its critiquing and refining capabilities further. Namely, we let the learned critic model $\mathcal{M}_1$ criticize and refine its own output. We first sample M correct-answer solutions and M incorrect-answer solutions for each question Q in the original dataset D. Then, for each attempt Att, we employ $\mathcal{M}_1$ to repeatedly criticize and refine at most k times. If the model fails to critique even after k times, we fall back on the critique from a stronger yet frozen model M_C as the final choice. Finally, we collect dataset $D_2 = \{(Q, Att, Cri)\} \bigcup \{(Q, Att, Cri_{-1}, Att')\}$ and use $D_1 \bigcup D_2$ to train the initial model $\mathcal{M}_0$ into the final critic model $\mathcal{M}_2$, which is similar to Wang et al. (2024). This procedure helps the model to learn to criticize and refine its own reasoning outputs better.

3.3 Inference: Self-Check

Iterative Refine One single-turn refinement, which consists of multiple steps, may still contain errors. Therefore, we could iteratively inspect the refined solution, and re-refine once the critique found a mistake, and only output the final solution if it's convincing for the critic, or if it reached the maximum retry. To avoid de-generation after too many refinements, we set the maximum refine depth $d = 8$, and restart from the initial solution after d unsuccessful refinement at most $n = 8$ times. Figure A4 presents a single successful round of critique and refinement.

Critic As Filter Self-consistency is an effective way to reduce variance. With the ability to critique, we can filter out predict-to-be-wrong answers to further boost the performance. Specifically, for the m attempts $S = \{(Att, Pred)\}$, we first let our model $\mathcal{M}$ check each attempt and obtain the stepwise label, which is $S_c = \{(Att, Pred, L)\}$.

Model	Sampling Method	GSM8K	MATH500
Base Model			
Llama-3-70B-Instruct (Dubey et al., 2024)	-	89.6	50.4
	Maj1@96	94.1	62.2
	Maj1@512	-	63.4
Llama-3.1-70B-Instruct (Dubey et al., 2024)	-	94.5	65.7
GPT4-0314 (OpenAI, 2023)	-	92.0	52.6
DeepSeek-V2 Chat-236B (DeepSeek-AI et al., 2024)	-	92.2	56.3
Reasoning-Enhanced Model			
MATH-Minos, Mistral-7B (Gao et al., 2024)	PRM+Maj1@256	87.8	38.6
InternLM-MATH-20B (Ying et al., 2024)	PRM Best-of-100	89.3	50.0
DART, Llama3-70B (Tong et al., 2024)	-	89.6	56.1
Math-Shepherd, DeepSeek-67B (Wang et al., 2023a)	PRM+Maj1@256	92.5	48.1
Ours			
Critic-CoT, Llama-3-70B-Instruct	-	91.7	57.6
	Iterative Refine	**93.3** ↑ 1.6	**57.8** ↑ 0.2
	Maj1@96	94.8	64.6
	Critic + Maj1@96	**95.4** ↑ 0.6	66.6 ↑ 2.0
	Maj1@512	-	65.4
	Critic + Maj1@512	-	**68.4** ↑ 3.0

Table 1: Solution Accuracy of GSM8K and MATH500. Compared with the base model, iterative refinement with our trained model improves from 89.6% to 93.3%% for GSM8K and from 50.4% to 57.8% for MATH500, while the critic filter increases the accuracy to 95.4% for GSM8K and 68.4% for MATH500.

Then those which detect the error at some step are filtered out and reach $S'_c = \{(Att, Pred, L) | -1 \notin L\}$. Finally, we perform the majority vote to get the answer.

4 Experiment

We apply the Critic-CoT training process on the training dataset of GSM8K and MATH (Section 4.1), and observe a noticeable performance improvement (Section 4.2), and out-of-domain evaluations on AGIEval, StrategyQA, and HumanEval further exhibits the generalization of our trained critic ability (Section 4.3) For more analysis, discussion, see Appendix A.3 and Appendix A.4, and the prompt is presented in Appendix A.6.

4.1 Setup

4.1.1 Model

We fine-tune the critic-refine model on Llama-3-70B-Instruct (Dubey et al., 2024), which was pre-trained on more than 15 Trillion tokens and has a context length of 8,192. For critique / refinement sampling, we use GPT4-Turbo (OpenAI, 2023) of the version gpt-4-0125-preview. We use the Huggingface Transformers (Wolf et al., 2020), DeepSpeed (Rajbhandari et al., 2021) and FastChat (Zheng et al., 2023) libraries for training. We use vLLM library (Kwon et al., 2023) for model inference, adapting top-p sampling of $p = 0.95$, with temperature 0.7 for solution sampling, which follows Cobbe et al. (2021a), and 0.5 for critique and refinement. All inferences are zero-shot.

4.1.2 Dataset

Train & In-Domain Eval Separately, we train our model on the problem of GSM8K (Cobbe et al., 2021a) and MATH (Hendrycks et al., 2021). GSM8K is a grade-school-level math word problem dataset, with 7,473 training instances and 1,319 test instances. MATH is a challenging high school math competition dataset, which consists of 7,500 training problems and 5,000 test problems. For the MATH dataset, we also follow the data split of Lightman et al. (2024), which adds 4,500 test problems into a training set and, therefore, contains 12,000 training instances and 500 representative test instances. More details are in Appendix A.1.

Out-of-Domain Eval To further evaluate our critic model's generalization capabilities beyond math, we assess its performance on reasoning tasks using the StrategyQA, AGIEval, and HumanEval datasets, which cover different domains. StrategyQA (Geva et al., 2021) is a multi-step reasoning task constructed from Wikipedia, with binary answers indicating either true or false. AGIEval (Zhong et al., 2023) comprises standardized exam questions from various fields, including college entrance exams, law school admission tests, math competitions, and lawyer qualification tests. Given the overlap with the MATH test set, we evaluated

Model	Acc.
Llama-3-70B-Instruct	56.6
Llama-3.1-70B-Instruct	61.8
DeepSeek-V2 Chat-236B	61.4
GPT4o	65.2
Critic-CoT, GSM8K	54.7
- Iterative Refine	55.6 ↑ 0.8
- Maj1@96	60.7
- Critic + Maj1@96	60.3 ↓ 0.4
Critic-CoT, MATH	59.8
- Iterative Refine	**63.7** ↑ 3.9
- Maj1@96	61.0
- Critic + Maj1@96	**61.2** ↑ 0.2

(a) AGIEval

Model	Acc.
Llama-3-70B-Instruct	76.2
Llama-3.1-70B-Instruct	84.3
DeepSeek-V2 Chat-236B	75.6
GPT4-0314	83.6
Critic-CoT, GSM8K	77.5
- Iterative Refine	78.8 ↑ 1.3
- Maj1@96	78.7
- Critic + Maj1@96	**80.5** ↑ 1.8
Critic-CoT, MATH	78.0
- Iterative Refine	**80.1** ↑ 2.1
- Maj1@96	78.3
- Critic + Maj1@96	79.7 ↑ 1.4

(b) StrategyQA

Model	Pass@1
Llama-3-70B-Instruct	76.2
Llama-3.1-70B-Instruct	80.5
DeepSeek-V2 Chat-236B	81.1
GPT4-0314	86.6
Critic-CoT, GSM8K	77.4
- Iterative Refine	**78.1** ↑ 0.7
Critic-CoT, MATH	84.1
- Iterative Refine	**84.8** ↑ 0.7

(c) HumanEval

Table 2: Solution Accuracy of standardize exam dataset AGIEval (2a), multi-hop reasoning dataset StrategyQA (2b) and code generation dataset HumanEval (2c). Our models generally show robust generalization.

our model using the original 7,500/5,000 split from MATH, rather than the extended 12,000/500 split. HumanEval (Chen et al., 2021) contains 164 handwritten Python program problems, which evaluate the code generation capability.

4.2 Critic-CoT Improves Mathematical Reasoning

The results of in-domain evaluation are shown in Table 1, which demonstrate **the effectiveness of Critic-CoT in improving the model's mathematical reasoning performance.** First, strengthening the ability to critique and refinement would not compromise the task-solving performance, but improve it. After Critic-CoT training, our model's top-1 accuracy increases from 89.6% to 91.7% on GSM8K, and from 50.4% to 57.6% on MATH500. Second, the step-wise self-critique ability of the models can further enhance the reasoning performance during inference, via Iterative Refinement and Critic As Filter. With Iterative Refinement, our model achieves 93.3% accuracy on GSM8K and 57.8% on MATH500. Applying Critic As Filter, the performance improves further on the basis of majority vote. On GSM8K it rises from 94.8% with Maj1@96 to 95.4% and on MATH500 it rises from 65.4% with Maj1@512 to 68.4%. Thus, our model's accuracy surpasses strong baselines of Process Reward Model assisted model MATH-Minos, InternLM-MATH-20B, Math-Shepherd, and rejection sampling finetuning model DART. Overall, the results indicate the model's enhanced abilities to identify mistakes and recover from them, thereby boosting the reasoning performance.

4.3 Critic-CoT Strengthens Out-of-Domain Reasoning

The results of out-domain evaluation are shown in Table 2. In general, our model, especially trained on MATH dataset, achieves positive performance gain with Iteartive Refinement and Critic As Filter, which demonstrates **the generalized critique and refinement abilities beyond the math training domains**. For StrategyQA, our critic models trained on two datasets show a positive performance increase when applying iterative refine and majority vote with the critic filter. On the more challenging dataset AGIEval, the Critic-CoT model trained on MATH performs much better than the model trained on grade-school level GSM8K dataset, and shows significant improvements in iterative refinement, rising to 63.7% compared with 56.6% prior the Critic-CoT training. On the code generation task HumanEval, the majority vote method is not applicable, but we can still observe the positive improvement with iterative refinement: while the base model's pass rate is 76.2%, our model trained on GSM8K and MATH achieved 78.1% and 84.8% respectively. These results highlight the robustness of our Critic-CoT models.

5 Ablation Analysis

To demonstrate the effectiveness of our Critic-CoT designs, we conduct a series of manual examination

Data	Critique of Wrong Attempt	Refinement of Wrong Attempt	Critique of Correct Answer Attempt
GSM8K	86%	97%	100%
MATH	85%	96%	92%

Table 3: Human Evaluation on the critique and refinement of Critic-CoT Training Data. The automatically constructed data maintain high quality, which can well support the critique training process.

Model	Critic				Refine		Majority Vote		
	P	R	F1	Acc.	Init. Acc	Ref. Acc.	Pass1@N	Maj1@N	+Critic
Outcome Label	**95.5**	28.9	44.4	88.0	87.7	89.7	99.0	93.6	93.7
Process Label	67.9	22.8	34.1	89.5	88.0	89.2	99.0	93.0	93.0
Only Refine	30.0	11.4	16.6	90.8	**92.0**	88.2	98.9	**95.2**	95.2
Only Critic	57.1	31.0	40.2	91.9	91.2	91.4	98.9	94.4	94.5
Stage 1	42.5	41.5	42.0	89.3	90.7	91.1	98.9	93.6	94.2
Stage 2	50.0	25.0	33.3	85.5	90.5	91.3	99.0	94.4	94.4
Critic-CoT	53.3	**58.2**	**55.7**	**92.3**	91.7	**93.3**	**99.1**	94.8	**95.4**

(a) GSM8K

Model	Critic				Refine		Majority Vote		
	P	R	F1	Acc.	Init. Acc	Ref. Acc.	Pass1@N	Maj1@N	+Critic
Outcome Label	**84.4**	39.0	53.3	63.0	51.8	53.6	84.0	56.2	56.2
Process Label	80.2	35.9	49.6	63.8	50.4	52.6	78.6	49.4	50.8
Only Refine	62.3	60.1	61.2	66.0	55.4	49.8	**90.4**	63.0	62.8
Only Critic	67.9	75.4	71.5	71.6	52.8	55.8	89.0	60.6	60.6
Stage 1	64.6	**93.7**	**76.5**	69.0	53.2	41.2	**90.4**	63.4	63.0
Stage 2	79.7	45.8	58.2	71.8	57.2	57.4	**90.4**	**64.6**	65.0
Critic-CoT	66.1	73.7	69.7	**72.2**	**57.6**	**57.8**	89.2	**64.6**	**66.6**

(b) MATH500

Table 4: Ablation Study on GSM8K and MATH500. We use the metrics from three aspects: critic, including precision, recall, f1-score and accuracy; Iterative Refine, including accuracy before and after the refinement; and Critic As Filter, including Pass1@96, Maj1@96, and Critic+Maj1@96. The ablation study demonstrates the effectiveness of our Critic-CoT design.

and ablation studies, which confirm that our training data is still in high quality (Section 5.1), our proposed stepwise CoT critique is advantageous (Section 5.2), the composition of training data is beneficial (Section 5.3), and the source of improvement shall be attributed to our Critic-CoT framework rather than distillation (Section 5.4).

5.1 The Quality of Constructed Data

To examine the correctness of constructed Critic-CoT training data, we perform the manual evaluation. We sample 300 entries (100 with the critique of correct answer and 100 with the critique and refinemnt of wrong answer) each from the critic-cot data on GSM8K and MATH, and conduct a manual verification to verify the accuracy of the step-wise critiques. For the critique of the correct answer attempt, it is valid if there is indeed no error in all the intermediate steps; for the critique of the wrong attempt, it is valid if the first error step and the reason for the error are both identified. A refinement is correct, if the continuation steps are flawless.

The results of manual verification are demonstrated in Table 3, with about 85% accuracy on wrong-answer critique, and more than 90% on refinement and correct-answer critique. **Therefore, the data we automatically constructed maintain a high level of accuracy at the step level, which can well support the critique training process**.

5.2 The Necessity of CoT Critique

To assess the necessity of the stepwise CoT critic, we remove CoT and train two baselines, namely "Process Label" and "Outcome Label". Specifically, for Process Label, the model is trained to directly predict the correctness of each step, e.g., "Step

1 is correct. Step 2 is incorrect.". Further, for Outcome Label, we remove the stepwise labels, which is to predict whether the entire solution is correct without providing step-wise details, e.g., "Each step from Step 1 to Step 8 is correct" or "Some step from Step 1 to Step 8 is incorrect". These two baselines can be viewed as the natural language versions of the Process Reward Model and Outcome Reward Model.

The results are shown in Table 4. We find that removing the Chain-of-Thought intermediate analysis and further stepwise labels, negatively impacts the critic accuracy, which indicates **the effectiveness of stepwise CoT critique in improving the discrimination performance**. From the original Critic-CoT model to the Process Label Model and the Outcome Label Model, the critique accuracy drops from 92.3% to 89.5% and further to 88.0% on GSM8K, and from 72.2% to 63.8% and further to 63.0% on MATH500. They fail to detect more errors and the recall metric is lower, despite its tendency to more easily pass correct solutions. Compared with System-1 like reasoning without explicit analysis, System-2 like reasoning with CoT critique can more precisely identify the errors.

5.3 The Impact of Training Data

To evaluate the impact of different data types during training, we perform a vertical ablation by removing either the critic data or the refinement data across both stages. In addition, we perform a horizontal ablation by training only with either stage 1 data or stage 2 data.

As the result shown in Table 4, we find that **training only on the single-type data is less optimal**. If only training to refine, the base model's critic ability remains weak, and still *cannot effectively self-correct*, as on GSM8K the accuracy drops from 92.0% to 88.2% after iterative refinement, and on MATH500 it drops from 55.4% to 49.8%. If only training to critic, the model obtains positive improvement from iterative refinement, with increases from 91.2% to 91.4% on GSM8K and 52.8% to 55.8% on MATH500, but still lags behind to Critic-CoT model that jointly trains the two capabilities. And finally, in terms of critic and reasoning performance, combining the data from two stages is better than training the data only from one stage.

5.4 Comparison with Solution Distilation

Since our data construction process leverages GPT-4 Turbo, one potential concern is whether the performance increase comes from distilling GPT-4, or our proposed Critic-CoT framework. To address this concern and better understand the source of improvement, we also train the traditional rejection sampling fine-tuning (RFT) baseline, using GPT4-Turbo to generate a correct-answer solution for each problem on the dataset of GSM8K and MATH respectively.

Model	GSM8K	MATH500
Llama-3-70B-Instruct	89.6	50.4
GPT4-Turbo Solution	90.7	48.0
Critic-CoT	**93.3**	**57.8**

Table 5: Comparision between solution distillation and our Critic-CoT model with Iterative Refine, with the metric of Top-1 Accuracy. Directly training on the trajectories of advanced LLMs did not necessarily improve the performance.

As the result shown in Table 5, unlike Critic-CoT, **directly distilling from frontier LLMs may not significantly improve the reasoning performance**, with only 90.7% on GSM8K and 48.0% on MAT500, which is less superior. On the one hand, the Llama-3-70B model has already experienced heavy post-training for downstream reasoning tasks (AI@Meta, 2024). On the other hand, as shown in Section 4.2, the improvements can be attributed to two key factors. First, strengthening the ability to critique and refinement, which is under-trained, directly improves the Top-1 accuracy. Moreover, at the inference phase, we can actively leverage the model's ability to reflect on its reasoning and correct mistakes, thus obtaining additional improvements that are not applicable without critic-refine training. **Therefore, our proposed Critic-CoT framework plays a crucial role in driving the observed performance improvements.**

6 Conclusion

In this paper, we introduced the Critic-CoT paradigm to enhance the reasoning abilities of Large Language Models, through a more System-2-like, step-by-step Chain-of-Thought critique. Our approach leverages weak supervision to construct training data for critiques and refinements, thereby reducing the reliance on extensive human annotation. We demonstrated the effectiveness of our method through substantial improvements across the dataset of GSM8K and MATH. Additionally, our results present that training on the capabilities of critique and refinement alone improves task-solving performance, which indicates a mutual-reinforce mechanism within the LLMs. We hope our work may inspire further investigations into the advancement of the self-critic framework and the transition toward System-2 reasoning.

Limitations

In this paper, we propose Critic-CoT, a framework to automatically construct critic-refine training data in the reasoning domain, where the correctness of a solution is verifiable by checking the final answer. For more general tasks, how to filter valid critiques and train robust critic models is worth investigating in future works.

Ethics Statement

All the data and models are acquired from public datasets and pre-trained models, and no human annotators are involved during the data construction procedure.

Acknowledgment

We sincerely thank the reviewers for their insightful comments and valuable suggestions. This work was supported by Beijing Natural Science Foundation (L243006), Beijing Municipal Science and Technology Project (Nos. Z231100010323002), and the Natural Science Foundation of China (No. 62476265, 62306303).

References

AI@Meta. 2024. Llama 3.1 model card.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *Preprint*, arXiv:2303.12712.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeff Wu. 2023. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *Preprint*, arXiv:2312.09390.

Boxi Cao, Keming Lu, Xinyu Lu, Jiawei Chen, Mengjie Ren, Hao Xiang, Peilin Liu, Yaojie Lu, Ben He, Xianpei Han, Le Sun, Hongyu Lin, and Bowen Yu. 2024. Towards scalable automated alignment of llms: A survey. *Preprint*, arXiv:2406.01252.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024a. Alphamath almost zero: process supervision without process. *CoRR*, abs/2405.03553.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. 2024b. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

François Chollet. 2019. On the measure of intelligence. *Preprint*, arXiv:1911.01547.

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. 2021a. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

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. 2021b. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, and et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *CoRR*, abs/2405.04434.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. 2025. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*.

Bofei Gao, Zefan Cai, Runxin Xu, Peiyi Wang, Ce Zheng, Runji Lin, Keming Lu, Junyang Lin, Chang Zhou, Wen Xiao, Junjie Hu, Tianyu Liu, and Baobao Chang. 2024. LLM critics help catch bugs in mathematics: Towards a better mathematical verifier with natural language feedback. *CoRR*, abs/2406.14024.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

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

Alexander Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. 2024. Teaching large language models to reason with reinforcement learning. In *AI for Math Workshop @ ICML 2024*.

Yancheng He, Shilong Li, Jiaheng Liu, Weixun Wang, Xingyuan Bu, Ge Zhang, Zhongyuan Peng, Zhaoxiang Zhang, Wenbo Su, and Bo Zheng. 2025. Can large language models detect errors in long chain-of-thought reasoning? *arXiv preprint arXiv:2502.19361*.

G.W.F. Hegel, T.F. Geraets, W.A. Suchting, and H.S. Harris. 1991. *The Encyclopaedia Logic, with the Zustze: Part I of the Encyclopaedia of Philosophical Sciences with the Zusatze*. Hackett Classics Series. Hackett.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating LLM hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843, Singapore. Association for Computational Linguistics.

Daniel Kahneman. 2011. *Thinking Fast and Slow*. Farrar, Straus and Giroux.

Søren Kierkegaard. 1989. *Kierkegaard's Writings, II, Volume 2: The Concept of Irony, with Continual Reference to Socrates/Notes of Schelling's Berlin Lectures*. Princeton University Press.

Geunwoo Kim, Pierre Baldi, and Stephen Marcus McAleer. 2023. Language models can solve computer tasks. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. 2018. Scalable agent alignment via reward modeling: a research direction. *Preprint*, arXiv:1811.07871.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. 2024. Chain of thought empowers transformers to solve inherently serial problems. In *The Twelfth International Conference on Learning Representations*.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17889–17904, Miami, Florida, USA. Association for Computational Linguistics.

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

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. CriticBench: Benchmarking LLMs for critique-correct reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1552–1587, Bangkok, Thailand. Association for Computational Linguistics.

Yixin Liu, Avi Singh, C. Daniel Freeman, John D. Co-Reyes, and Peter J. Liu. 2023. Improving large language model fine-tuning for solving math problems. *CoRR*, abs/2310.10047.

Liangchen Luo, Zi Lin, Yinxiao Liu, Lei Shu, Yun Zhu, Jingbo Shang, and Lei Meng. 2023. Critique ability of large language models. *Preprint*, arXiv:2310.04815.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. 2024. Improve mathematical reasoning in language models by automated process supervision. *Preprint*, arXiv:2406.06592.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. 2024. Llm critics help catch llm bugs. *Preprint*, arXiv:2407.00215.

Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksandra Faust, Clement Farabet, and Shane Legg. 2024. Position: Levels of AGI for operationalizing progress on the path to AGI. In *Forty-first International Conference on Machine Learning*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

OpenAI. 2024. Openai o1 system card.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Karl Raimund Popper. 1934. *The Logic of Scientific Discovery*. Routledge, New York.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA. Association for Computing Machinery.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022. Self-critiquing models for assisting human evaluators. *Preprint*, arXiv:2206.05802.

Tom Schaul. 2024. Boundless socratic learning with language games. *arXiv preprint arXiv:2411.16905*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *Preprint*, arXiv:2408.03314.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 2024. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *Preprint*, arXiv:2407.13690.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. 2023a. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *CoRR*, abs/2312.08935.

Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. 2024. Self-taught evaluators. *Preprint*, arXiv:2408.02666.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Peter West, Ximing Lu, Nouha Dziri, Faeze Brahman, Linjie Li, Jena D. Hwang, Liwei Jiang, Jillian Fisher, Abhilasha Ravichander, Khyathi Chandu, Benjamin Newman, Pang Wei Koh, Allyson Ettinger, and Yejin Choi. 2024. The generative AI paradox: "what it can create, it may not understand". In *The Twelfth International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin

Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. 2024. Large language models can self-correct with key condition verification. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12846–12867, Miami, Florida, USA. Association for Computational Linguistics.

Hanqi Yan, Qinglin Zhu, Xinyu Wang, Lin Gui, and Yulan He. 2024. Mirror: Multiple-perspective self-reflection method for knowledge-rich reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7086–7103, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, and et al. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

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. 2024. Internlm-math: Open math large language models toward verifiable reasoning. *CoRR*, abs/2402.06332.

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. 2024. Distilling system 2 into system 1. *Preprint*, arXiv:2407.06023.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. 2024. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STar: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*.

Zhongshen Zeng, Pengguang Chen, Shu Liu, Haiyun Jiang, and Jiaya Jia. 2023. Mr-gsm8k: A meta-reasoning benchmark for large language model evaluation. *CoRR*, abs/2312.17080.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024a. Generative verifiers: Reward modeling as next-token prediction. *Preprint*, arXiv:2408.15240.

Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. 2024b. Self-contrast: Better reflection through inconsistent solving perspectives. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3602–3622, Bangkok, Thailand. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *Preprint*, arXiv:2304.06364.

A Appendix

A.1 Training & Evaluation Details

A.1.1 Critic Data Construction

GSM8K On GSM8K, since `GPT-4` already got 92.0% accuracy on the test set (OpenAI, 2023), which makes it hard to obtain negative data, we use `GPT-3.5-Turbo-0125` instead to sample 10 solutions for each question in the training set. Then, we use `GPT-4-Turbo` as the critic-refine model to criticize the solutions (Table A7), with $K = 16$ retry. We obtain 63,485 cases, with 49,832 positive examples and 13,653 negative examples.

In the second stage of GSM8K critique construction, we use the learned critic model to repeatedly sample until we obtain at most 5 positive and 5 negative solutions. For strong LLMs like `LLaMA-3`, it's challenging to get enough negative solutions even among 512 samples, so the size of negative data would be slightly smaller. Then, we use the learned critic model to criticize itself, also with $K = 16$ retry. In stage two, we obtain 62,877 instances, with 39,654 positive and 26,001 negative. Among the two stages, we got 126,362 instances, with 86,708 positive and 39,654 negative.

MATH On MATH, in the first stage, we directly use the 90,074 `GPT-4` generated solutions of PRM800K Dataset (Lightman et al., 2024), with 11,665 positive instances which all the step labels are correct, and 78,409 negative instances which one step label is incorrect. Since the MATH dataset is challenging, in order to reduce retry of `GPT-4-Turbo` and avoid not getting valid critique, for the critique of the negative solution, we additionally append reference solution in the input prompt, and hint it might contain mistakes, as suggested in prior work (Zelikman et al., 2022); for the positive solution, we simply hint it's correct. After obtaining the initial critique, we use `GPT-4-Turbo` again to remove hint phrases like "According to the reference" or "Given the hint" since we do not have any hint or reference during the test time. In stage one, we obtain 1,606 positive cases and 69,775 negative cases.

Similarly, in the second stage of MATH, we use the learned critic model to sample at most 5 positive and negative solutions. Then, we first use the critic model itself to critic its solutions, and without any hints, under $K = 16$ retry, and use `GPT-4-Turbo` to retry another $K = 16$ times with hint if failed. We construct 51,618 positive cases and 65,456 negative cases. Among the two stages, we got 188,455 cases, with 53,224 positive and 135,231 negative.

A.1.2 Answer Extraction

We let the model print the answer in the format `\boxed{answer}`. The model generates the answer following this pattern. We then extract the regular expression `\boxed{.*}` from the model output, and obtain the valid answer expression with matched parenthesis. The Python code for answer extraction is shown in Table A5.

A.1.3 Evaluation Metric

Solution For the evaluation of the solution, we extract the final answer (Appendix A.1.2) and compute the metrics of Top-1 Accuracy `Acc` and Refine Accuracy `Refine-Acc`, in which the original Top-1 predict-answer is replaced with a refined one if the critic model found an error and made iterative refinement (Section 3.3). We also compute Majority Vote Accuracy `Maj1@N` (Wang et al., 2023b) and Majority Vote Accuracy After Critique `Critic + Maj1@N` (Section 3.3), which is to select the most frequent answer among N samples, i.e. $\arg\max_a \sum_{i=1}^{N} \mathbb{1}(\mathbf{a}_i = a)$. Following Liu et al. (2023); Havrilla et al. (2024), we compute `Pass@N`, which select the gold answer g among the N predictions if present, i.e. $\arg\max_a \mathbb{1}(g = a)$.

Critique For the "evaluation of evaluation", we compute Precision, Recall, and F1 for error detection; also, we compute Critic Accuracy, where the critique should find the error in wrong answer solutions and pass the correct answer solution:

$$P = \frac{\{|Pred_i \neq Ans_i \wedge -1 \in L_i|\}}{|\{-1 \in L_i\}|} \tag{1}$$

$$R = \frac{\{|Pred_i \neq Ans_i \wedge -1 \in L_i|\}}{|\{Pred_i \neq Ans_i\}|} \tag{2}$$

$$F1 = \frac{2 * P * R}{P + R} \tag{3}$$

$$\begin{aligned} &CriticAcc = \\ &\frac{1}{N}\sum_{i=1}^{N}(Pred_i = Ans_i \wedge -1 \notin L_i) \\ &\vee (Pred_i \neq Ans_i \wedge -1 \in L_i) \end{aligned} \tag{4}$$

Here, for the i-th instance, $Pred_i$ is the prediction answer, Ans_i is the ground truth answer, and L_i is the predicted step label list.

A.1.4 Implementation Details

The training on GSM8K dataset takes 23 hours, while on MATH dataset it takes 37 hours. The two datasets are under MIT license. For software, Huggingface Transformers, DeepSpeed, vLLM, and FastChat libraries we used are under Apache-2.0 license.

A.2 Additional Result on MATH

Table A1 presents the results of Critic-CoT training on MATH dataset, with the original 7,500/5,000 split setting.

A.3 Analysis

A.3.1 Critic Performance

For both datasets, the critic model's accuracy continues to grow as the sample size N increases, ultimately surpassing the performance of the majority vote, which gradually converges. Specifically, in the MATH dataset, the critic model achieves substantially higher accuracy than the solution accuracy, consistently outperforming the naive majority vote due to the critic filter's superior performance. This stark contrast highlights the critic model's effectiveness in identifying and promoting correct answers. In the GSM8K dataset, despite having a critic accuracy of only 92.3%, the critic model still manages to deliver higher accuracy gains. This outcome suggests that the critic model successfully filters answers to increase the density of correct answers and decrease the density of wrong answers, compared to the normal answer distribution. The overall results demonstrate the critic model's robust capability to enhance accuracy across different datasets, validating its practical utility in improving prediction outcomes.

A.3.2 Inspect on Iterative Refine

The iterative refinement process for the GSM8K and MATH datasets demonstrates different levels of effectiveness due to their complexity, as shown in Table A2. GSM8K, being simpler, shows a higher success rate in refinement. For effective refinement, the number of false answers corrected (False $\rightarrow$ True) must exceed the number of true answers incorrectly changed (True $\rightarrow$ False). Despite occasional mistakes by the critic, correct answers are not always altered incorrectly.

For GSM8K (Table A2a), accuracy improves from 91.7% initially to 93.3% by the seventh round, with significant gains in both true-to-true and false-to-true transformations. In contrast, MATH (Table A2b) starts at 57.6% accuracy, reaching 57.8% by the seventh round. The iterative refinement process tends to converge, which is expected.

A.3.3 Group By Difficulty Level

For the MATH dataset, the difficulty level is given from 1 to 5. For the GSM8K dataset, we set the difficulty level according to the number of expressions n that appeared in the reference solution, i.e., $max(1, min(5, n))$. As illustrated in Figure A2, the performance on the GSM8K dataset shows a gradual decline as the difficulty level increases. This trend is accompanied by the emerging effects of the critic and refine stages, which become more prominent at higher difficulty levels. In contrast, the accuracy on the MATH dataset declines sharply as the problems become more challenging. Generally, the refine stage proves effective across all levels, while the critic stage is beneficial at most levels, with some minor exceptions. These observations suggest potential areas for further improvements in the critic mechanism.

A.4 Discussion

A.4.1 Discriminative Verifier for Mathematics

To further improve the reasoning ability of large language models, one applicable approach is through the use of reward models, which can either be used in reinforcement learning during training (Ouyang et al., 2022) or rejection sampling at test time (Cobbe et al., 2021b). While outcome-supervised reward models (ORMs) allow for the automatic collection of training data based on the signal of the gold answer, process-supervised reward models (PRMs) would be more advantageous for more precise feedback, better interpretability and stronger alignment (Lightman et al., 2024).

To reduce the considerable human labeling cost and difficulty for dense annotation, a series of works based on automatic approaches have been proposed (Wang et al., 2023a; Chen et al., 2024a; Luo et al., 2024; Snell et al., 2024), all under the heuristic that for an incorrect solution, the first error step is where the continuation of previous step would lead to a correct answer. This may bring noise into training data due to false positives and negatives (Luo et al., 2024). Moreover, annotation based on the implicit solution continuation alone does not leverage LLM's emerging ability of critic, which is in a more explicit and analytic way and brings better explainability (Saunders et al., 2022; Yuan et al., 2024; Luo et al., 2023;

Model	Sampling Method	Acc.
Llama-3-70B-Instruct (Dubey et al., 2024)	-	51.0
	Maj1@96	63.5
	Maj1@512	64.3
Llama-3.1-70B-Instruct (Dubey et al., 2024)	-	68.0
DeepSeek-V2 Chat-236B (DeepSeek-AI et al., 2024)	-	53.9
Qwen2-72B (Yang et al., 2024)	-	69.0
GPT4-0314 (OpenAI, 2023)	-	42.5
Critic-CoT, Llama-3-70B-Instruct (*Ours*)	-	**56.2**
	Iterative Refine	**56.6** ↑ 0.4
	Maj1@96	64.2
	Critic + Maj1@96	**65.0** ↑ 0.8
	Maj1@512	64.4
	Critic + Maj1@512	**66.4** ↑ 2.0

Table A1: Solution Accuracy of MATH. The top-1 accuracy of our method increases from 51.0% to 56.2%, and the effect of iterative refinement is moderate but positive improvement of 0.4%, while the performance gain of the critic filter is larger.

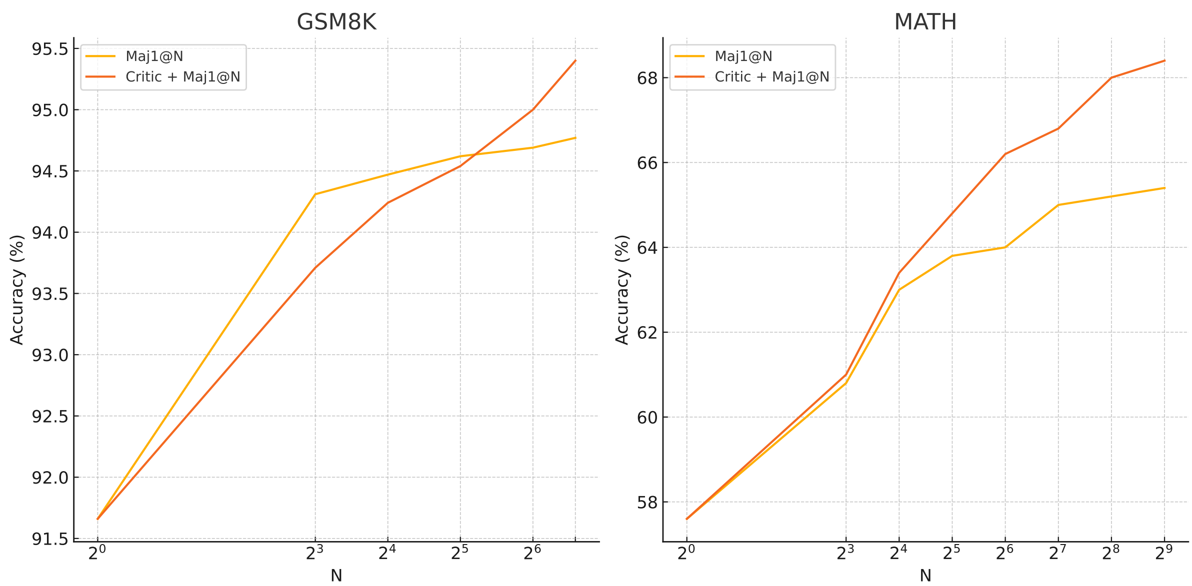


Figure A1: Performance of majority vote on GSM8K and MATH500 Datasets

McAleese et al., 2024). Additionally, binary 0/1 discrimination alone, whether outcome-based or process-based, remains more similar to System-1 reasoning rather than the desirable System-2, thus may not fully leverage the computation power support by empirically successful Chain-of-Thought prompting (Feng et al., 2023; Li et al., 2024).

A.4.2 Differences Between Critic-CoT and Reflexion

We adopt a similar approach to Relexion (Shinn et al., 2023), which leverages natural language critique to facilitate refinement, but our method diverges in the following ways:

Step-wise CoT Critique Reflexion translates and augments the binary reward signal from the environment to natural language, but on an instance level. Instead, fine-grained Chain-of-Thought analysis at the step level, which is more systematic, and enables us to locate the error and start refinement from a specific step, rather than refine the whole attempt.

Enhanced Critic ability While Relexion proposed an in-context learning pipeline for policy optimization under the oracle success/fail binary feedback signal, Huang et al. (2024) showed that without external feedback, vanilla LLMs cannot self-correct effectively due to limited critique ability. Therefore, to teach the LLMs the ability of intrinsic self-critique, our approach tries to learn the critique ability itself, through Critic-CoT train-

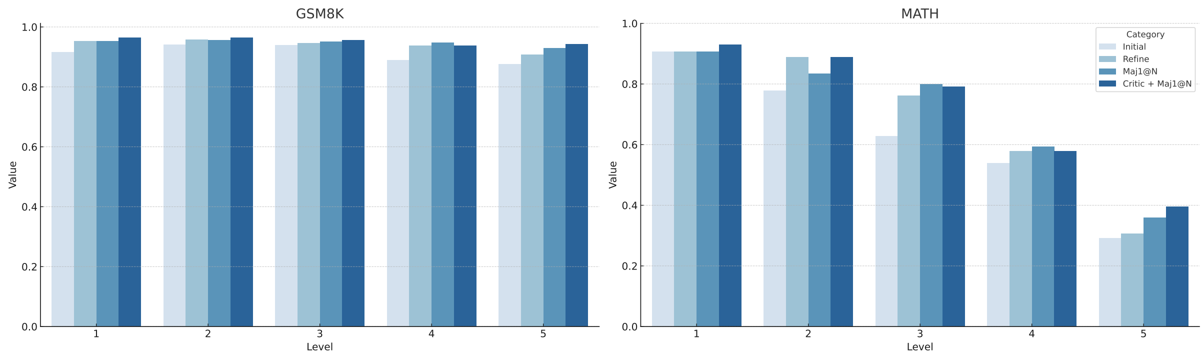


Figure A2: Performance group by difficulty level, on GSM8K and MATH500 Datasets

Round	Refine Acc.	True → True	False → True
0	91.7	-	-
1	91.7	48.2	45.3
2	92.6	78.6	37.5
3	92.7	64.3	**53.1**
4	93.0	73.2	50.0
5	93.2	75.0	**53.1**
6	93.2	76.8	**53.1**
7	**93.3**	**80.4**	50.0
8	**93.3**	**80.4**	50.0

(a) GSM8K

Round	Refine Acc.	True → True	False → True
0	57.6	-	-
1	53.4	29.0	17.7
2	57.2	**65.7**	13.9
3	55.2	48.6	**15.2**
4	57.2	60.9	15.9
5	57.4	60.0	17.1
6	57.6	61.4	17.1
7	**57.8**	60.0	**18.4**
8	**57.8**	62.9	16.5

(b) MATH500

Table A2: Iterative Refine on GSM8K (A2a) and MATH500 (A2b).

ing, and can apply it to test-time situations where the oracle feedback signal is not available.

A.4.3 Differences between Critic-CoT and Long Reasoning Model

Compared with these reasoning models, our method differs in terms of training mechanism and inference method. While reasoning models attempt to improve the accuracy with longer CoT output, our Critic-CoT method introduces explicit, step-wise critique, which helps identify the error more precisely and facilitates refinement. The key distinctions are as follows:

Controllability and Efficiency It could be non-trivial to control the long reasoning models to critic and refine during inference. When to critic, critic at which step, and when to restart depends on the model itself, and they often exhibit "overthinking," performing unnecessary reflections or exploring multiple solutions even for simple problems, leading to increased computational costs without proportionate gains (Chen et al., 2024b). On the contrary, our method can achieve controllable test-time scaling, by specifying the number of reflection rounds for Iterative Refine, and the number of samples for Critic As Filter.

Adaptability While models like R1 (Guo et al., 2025) are trained with reinforcement learning with outcome reward and implicitly learn the critic and refine patterns, this self-reflection behavior is not guaranteed to emerge reliably across all models (Gandhi et al., 2025). Instead, our proposed framework explicitly teaches the model step-level critique and refinement, and can be easily adapted to different LLMs of various sizes.

Moreover, we believe our approach is complementary to long reasoning models. Given the limited self-critic ability of current R1-like models (He et al., 2025), our method could strengthen these models to precisely detect errors while maintaining confidence in correct steps. Reasoning models with enhanced self-critic ability could potentially reduce overthinking, boost performance for more complex tasks (Phan et al., 2025), and move toward self-improvement and scalable oversight (Schaul, 2024).

A.4.4 Comparison Between Inference Methods

The statistics of average tokens and latency per instance are shown in Table A3 and A3. For Iterative

GSM8K				MATH			
Round	Acc.	Avg. Token	Avg. Latency (s)	Round	Acc.	Avg. Token	Avg. Latency (s)
0	91.7	176.7	1.156	0	57.6	296.7	1.941
1	91.7	377.6	2.470	1	53.4	694.5	4.544
2	92.6	395.1	2.585	2	57.2	872.8	5.710
3	92.7	408.2	2.671	3	55.2	1003.0	6.562
4	93.0	418.5	2.738	4	57.2	1114.4	7.291
5	93.2	426.3	2.789	5	57.4	1202.0	7.864
6	93.2	432.6	2.830	6	57.6	1282.7	8.392
7	93.3	436.8	2.858	7	57.8	1343.1	8.787
8	93.3	440.4	2.881	8	57.8	1390.6	9.098

Table A3: Inference cost of Iterative Refine on GSM8K and MATH500.

GSM8K				MATH			
Sample	Acc.	Avg. Token	Avg. Latency (s)	Sample	Acc.	Avg. Token	Avg. Latency (s)
1	91.7	176.7	1.156	1	57.6	296.7	1.941
8	93.7	2974.6	2.433	8	61.0	4446.33	3.636
16	94.2	6261.6	5.121	16	63.4	9198.43	7.523
32	94.5	12524.1	10.242	32	64.8	19228.69	15.726
64	95.0	25057.9	20.493	64	66.2	41709.00	34.110
96	95.4	37574.6	30.729	96	66.6	60907.41	49.811

Table A4: Inference cost of Critic As Filter on GSM8K and MATH500.

Refinement, as the number of rounds increases, the computation cost and latency slowly increase, but the performance gain gradually becomes saturated. The reason for non-linear computation increase is that, refinement only occurs if the critique detects that the attempt is wrong and stops if the critique validates it. Yet, the pipeline of iterative refinement could be more sensitive to error accumulation, which could limit performance.

For Critic As Filter, the computation cost and latency increase linearly as the number of samples increases, and the performance also improves steadily. The linear cost increase is because we need to sample the attempt first, then perform CoT critic for each step. To reduce latency, we can parallelize the inference, and in our setting, we use 32 GPUs for inference, which deploys 8 models in total.

As the results in Table 1 shows, the performance of majority vote and Critic As Filter surpass Iterative Refinement. We believe it's due to the intrinsic challenges of refinement and the relatively limited search space.

On the one hand, for Iterative Refinement to work properly, it requires the model to 1) Detect errors on an attempt; 2) Refine the mistakes; 3) Exit if no further errors are detected. This pipeline could be more sensitive to error accumulation. Moreover, it only edits on a single example and has a limited retry, which is sample-efficient, but may not explore the solution space more actively, as majority vote does. Specifically on the dataset of GSM8K, the invocation statistics are as follows:

- Majority vote: 1319 * 96 = 126,624
- Iterative Refinement: among 1319 test cases, our Critic-CoT model predicts 274 problematic instances and iterates 1627 times (on average 5.94 rounds for each wrong case), which makes in total 1319 * 2 + 1627 * 2 = 5892 invocations, which is 21.5 times fewer than Majority Vote calls.

On the other hand, majority vote is a strong baseline, as it requires massive sampling. It leverages diverse reasoning paths and tries to mitigate the stochastic of a single sample. But under the method of Critic As Filter, we actively filter out problematic attempts and perform the majority vote on the more reasonable candidates, rather than equally account for all the predicted answers as the vanilla majority vote does, which further increases performance. This in turn demonstrates our model's strong ability to critique.

A.4.5 Self-Reflection

Besides the main results, through out-of-domain evaluation in Table 2, we find our model demonstrates generalized ability to critique and refine. While the ability of LLMs to self-reflect still remains an open question, and we hope our work as a valuable exploration could shed light on future studies in this area.

Moreover, as long as we adequately improve the models' ability to critique, we could achieve test-time performance increase in the form of "self-reflection". As the experiment results present, after Critic-CoT training, the ability to critique and generate both improves, though they are not exactly identical. Notably, the critique ability can surpass the task-solving ability, allowing the model to detect errors even when it has a low probability of generating a valid solution, as prior works (Saunders et al., 2022; Lin et al., 2024) also suggest. This indicates that by strengthening the model's CoT critique ability beyond its generation capability, we can leverage this discriminative power to reject imperfect responses and achieve positive performance gains.

A.4.6 Process Correctness of Correct Answer Attempt

We each sample 100 correct answer solutions, on GSM8K by GPT-3.5-Turbo and GPT-4-Turbo, and MATH by GPT-4-Turbo, and manually check if all intermediate steps are correct. The results are demonstrated in Table A6. We find that in general, the correct final answer is a good indicator of correct intermediate steps. Also, from GSM8K to MATH, as the reasoning traces become longer and more complicated, the percentage of correct answer but with wrong intermediate steps increases.

A.4.7 Alternatives of Leveraging Proprietary Models for Data Construction

To mitigate the data bias from GPT-4, we can introduce other open-source LLMs like DeepSeek-V3 or Qwen2.5-Max to perform multi-agent debates (Chan et al., 2024; Liang et al., 2024). This could encourage diverse thinking trajectories and overcome the inherited bias from a single model.

To reduce computation costs, we may substitute with smaller models. While our weak supervision method can effectively filter out problematic critic, to maintain a high success rate of generating valid data for smaller models, we can apply iterative refinement with more rounds, Monte Carlo Tree Search that more efficiently explores the search space, and other techniques. This can facilitate smaller models to produce desirable critic-refine data that is comparable with larger models.

A.5 Examples of Refinement

As presented in Figure A3, in an example of GSM8K, the model forgot to add one year at Step 3; then, through CoT critique, the model found that while Step 1 and Step 2 are correct, Step 3 contains this ignorance error. Finally, guided by the critique of Step 3, the model made a correction and reached the gold answer of 13. Similarly, in an example of MATH (A4), the model identified and successfully fixed the error.

A.6 Prompts

Table A7, Table A10, and Table A8 presents the prompt for critic-refine data collection using GPT4-Turbo, with Table A9 for removing the hint phrases (Section 3.2). Table A11, Table A12, and Table A13 shows the prompt of trained model for solving, critique, and refinement during stage-2-training (Section 3.2) and inference (Section 3.3). Table A14, Table A15 and Table A16 present the prompts and responses of a single-turn critique-refinement, under Critic-CoT, Step-wise Label Critic and Final Label Critic respectively. Table A17 presents a single-turn critique-refinement on HumanEval.

```
import re

def extract_boxed_expressions_custom(text):
  stack = []
  current_expr = ""
  i = 0
  while i < len(text):
    if text[i:i+7] == r"\boxed{":
        if stack:
            current_expr += text[i]
        stack.append("{")
        i += 7
    elif text[i] == "{" and stack:
        stack.append("{")
        current_expr += text[i]
        i += 1
    elif text[i] == "}" and stack:
        stack.pop()
        if stack:
            current_expr += text[i]
        else:
            current_expr = re.split('=', current_expr)[-1]
            return current_expr
        i += 1
    elif stack:
        current_expr += text[i]
        i += 1
    else:
        i += 1
  return ''

def extract_answer(text):
  predict_answer_span = re.findall(r'(\boxed{.*})', text, re.DOTALL)[-1]
  predict_answer = extract_boxed_expressions_custom(predict_answer_span)
  return predict_answer
```

Table A5: Python code for extracting the boxed answer.

Data	Model	Intermediate Accuracy of Correct Answer Attempt
GSM8K	GPT-3.5-Turbo	97%
GSM8K	GPT-4-Turbo	99%
MATH	GPT-4-Turbo	93%

Table A6: Human Evaluation on the Intermediate Steps of Attempts with Correct Answer

Question:
Carlos is planting a lemon tree. The tree will cost $90 to plant. Each year it will grow 7 lemons, which he can sell for $1.5 each. It costs $3 a year to water and feed the tree. How many years will it take before he starts earning money on the lemon tree?

Model Attempt:
Step 1: Calculate the annual income from selling lemons. This is done by multiplying the number of lemons grown per year (7) by the price per lemon ($1.5). So, 7 * $1.5 = $10.50.
Step 2: Subtract the annual cost of watering and feeding the tree ($3) from the annual income from selling lemons ($10.50). So, $10.50 - $3 = $7.50. This is the net income per year after expenses.
Step 3: Determine when Carlos will start earning money back. This is done by dividing the initial cost of planting the tree ($90) by the net income per year ($7.50). So, $90 / $7.50 = 12 years.
Therefore, it will take Carlos $\boxed{12}$ years before he starts earning money on the lemon tree.

Model Critique:
Step 1: The attempt calculates the annual income from selling lemons by multiplying the number of lemons grown per year (7) by the price per lemon ($1.5), resulting in $10.50.
Conclusion: Step 1 is correct.

Step 2: The attempt subtracts the annual cost of watering and feeding the tree ($3) from the annual income from selling lemons ($10.50), resulting in a net income per year of $7.50.
Conclusion: Step 2 is correct.

Step 3: The attempt divides the initial cost of planting the tree ($90) by the net income per year ($7.50), resulting in 12 years. However, this calculation does not account for the fact that Carlos will start earning money back in the first year, even though he won't break even until the 13th year.
Conclusion: Step 3 is incorrect.

Model Refinement:
Step 3 (Corrected): Determine when Carlos will start earning money back. This is done by dividing the initial cost of planting the tree ($90) by the net income per year ($7.50). So, $90 / $7.50 = 12 years.
However, since Carlos will start earning money back in the first year, we need to add one more year to this calculation.
Therefore, it will take Carlos $\boxed{13}$ years before he starts earning money on the lemon tree.

Figure A3: Criticize and refine a problem in the GSM8K test set.

Question:
What is the smallest positive perfect cube that can be written as the sum of three consecutive integers?

Model Attempt:
Step 1: Let the three consecutive integers be $x-1$, x, and $x+1$. The sum of these three integers is $3x$.
Step 2: Since we are looking for a perfect cube, let's express $3x$ as a cube of some integer n, where $n^3 = 3x$.
Step 3: To find the smallest such n, we need to minimize x. However, since n^3 must be divisible by 3, and x is an integer, n itself must also be divisible by 3.
Step 4: The smallest integer n that is divisible by 3 and gives us a perfect cube when cubed is $n = 6$, because $6^3 = 216$.
Step 5: Substituting $n = 6$ into $n^3 = 3x$, we get $216 = 3x$, which simplifies to $x = 72$.
Step 6: Therefore, the smallest positive perfect cube that can be written as the sum of three consecutive integers is $\boxed{216}$.

Model Critique:
Step 1: Let the three consecutive integers be $x-1$, x, and $x+1$. The sum of these three integers is indeed $3x$.
Conclusion: Step 1 is correct.

Step 2: Expressing $3x$ as a cube of some integer n, where $n^3 = 3x$, correctly sets up the equation needed to solve the problem. This step accurately translates the problem statement into a mathematical expression.
Conclusion: Step 2 is correct.

Step 3: The claim that n must be divisible by 3 because n^3 must be divisible by 3 is a misunderstanding. In reality, n^3 being divisible by 3 does not imply that n itself must be divisible by 3; rather, it simply means that n^3 will always be divisible by 3 regardless of whether n is divisible by 3. This step introduces an unnecessary restriction on n.
Conclusion: Step 3 is incorrect.

Model Refinement:
Step 3 (Corrected): Since we are looking for the smallest positive perfect cube that can be expressed as $3x$, we should start by identifying the smallest perfect cubes and checking if any of them can be represented in this form.
Step 4: The smallest perfect cubes are $1^3 = 1$, $2^3 = 8$, $3^3 = 27$, $4^3 = 64$, $5^3 = 125$, $6^3 = 216$, and so on. Among these, 27 is the first number that can be expressed as $3x$, where x is an integer ($x = 9$).
Step 5: Therefore, the smallest positive perfect cube that can be written as the sum of three consecutive integers is $\boxed{27}$.

Figure A4: Criticize and refine a problem in the MATH test set. In the attempt, the model made a hallucination in Step 3, then through stepwise CoT critique, the model detected this error, and accordingly refined the solution starting from Step 3 successfully.

Prompt

How do you evaluate the following attempt with respect to the problem?

<problem>
{problem}
</problem>

<attempt>
{attempt}
</attempt>

—
Notes:
- Please think step by step.
- Your reasoning should precede any claims or conclusions you make to avoid unwarranted assertions.
- At the end of the evaluation for each step, YOU MUST articulate the conclusion using the format ”Conclusion: Step [i] is correct" or ”Conclusion: Step [i] is incorrect". Words like ”partially correct" are prohibited.
- You shall not evaluate multiple steps at a time, so words like "Step 7 to Step 24:" or "Step 4 through 6" are forbidden.
- Once a mistake is identified and stated, stop the evaluation, and enumerate the corrected steps starting from the step where the mistake was detected, and label this part of your response with <correction> at the start and </correction> at the end. Also, the final answer should be a single number, in the form \boxed{}, at the final step.

Table A7: The prompt for the collection of critique and refinement on GSM8K, using GPT4-Turbo.

Prompt

How do you evaluate the following attempt with respect to the problem, with the help of reference solution?
Hint: There could be a mistake.
 	
`<problem>`
{problem}
`</problem>`
 	
`<reference_solution>`
{reference_solution}
`</reference_solution>`
 	
`<attempt>`
{attempt}
`</attempt>`
 	
—
Notes:
- Please think step by step.
- Your reasoning should precede any claims or conclusions you make to avoid unwarranted assertions.
- Please ensure that the output text does not include phrases implying the use of a reference solution or hint, even though these resources are being utilized.
- At the end of the evaluation for each step, YOU MUST articulate the conclusion using the format ”Conclusion: Step [i] is correct" or ”Conclusion: Step [i] is incorrect". Words like ”partially correct" are prohibited.
- You shall not evaluate multiple steps at a time, so words like "Step 7 to Step 24:" or "Step 4 through 6" are forbidden.
- Once a mistake is identified and stated, stop the evaluation, and enumerate the corrected steps starting from the step where the mistake was detected, and label this part of your response with `<correction>` at the start and `</correction>` at the end. Also, the final answer should be in the form \boxed{}, at the final step.

Table A8: The prompt for the collection of critique and refinement on MATH incorrect attempt, using GPT4-Turbo.

Prompt
For the following text, remove any phrases like "reference solution" or "hint", and keep all the other content. Do not miss the ”`<correction>`" and ”`</correction>`" labels that exist in the text. Do not respond to anything else. — {critique_refinement}

Table A9: The prompt for removing the hint of critique and refinement on MATH, using GPT4-Turbo.

Prompt

How do you evaluate the following attempt with respect to the problem?
Hint: All the steps are correct, and the attempt reached a correct answer.

<problem>
{problem}
</problem>

<attempt>
{attempt}
</attempt>

—
Notes:
- Please think step by step.
- Your reasoning should precede any claims or conclusions you make to avoid unwarranted assertions.
- Please ensure that the output text does not include phrases implying the use of a reference solution or hint, even though these resources are being utilized.
- At the end of the evaluation for each step, YOU MUST articulate the conclusion using the format ”Conclusion: Step [i] is correct" or ”Conclusion: Step [i] is incorrect". Words like ”partially correct" are prohibited.
- You shall not evaluate multiple steps at a time, so words like "Step 7 to Step 24:" or "Step 4 through 6" are forbidden.
- Once a mistake is identified and stated, stop the evaluation, and enumerate the corrected steps starting from the step where the mistake was detected, and label this part of your response with <correction> at the start and </correction> at the end. Also, the final answer should be in the form \boxed{}, at the final step.

Table A10: The prompt for the collection of critique on MATH correct attempt, using GPT4-Turbo.

Prompt

Problem
{problem}

—
Solve the problem step by step, marking each step as "Step [i]:".
Your final answer should be in the form \boxed{answer}, at the end of your response.

Table A11: The prompt for solution sampling.

Prompt
How do you evaluate the following attempt with respect to the problem? <problem> {problem} </problem> <attempt> {attempt} </attempt> — **Notes**: - Please think step by step. - Your reasoning should precede any claims or conclusions you make to avoid unwarranted assertions. - At the end of the evaluation for each step, YOU MUST articulate the conclusion using the format ”Conclusion: Step [i] is correct" or ”Conclusion: Step [i] is incorrect". Words like ”partially correct" are prohibited.

Table A12: The prompt for critique.

Prompt
How do you refine the following attempt with respect to the problem, given the criticism? <problem> {problem} </problem> <attempt> {attempt} </attempt> <criticism> {wrong_step_criticism} </criticism>

Table A13: The prompt for refinement.

Prompt of Critique under Critic-CoT

How do you evaluate the following attempt with respect to the problem?

<problem>
Consider the geometric sequence $\frac{125}{9}, \frac{25}{3}, 5, 3, \ldots$. What is the eighth term of the sequence? Express your answer as a common fraction.
</problem>

<attempt>
Step 1: Identify the first term of the sequence.
The first term a_1 is:

$$a_1 = \frac{125}{9}$$

Step 2: Calculate the common ratio r.
The common ratio r can be found by dividing the second term a_2 by the first term a_1:

$$r = \frac{a_2}{a_1} = \frac{\frac{25}{3}}{\frac{125}{9}} = \frac{25}{3} \times \frac{9}{125}$$

Calculating the right-hand side:

$$= \frac{25 \cdot 9}{3 \cdot 125} = \frac{225}{375}$$

Now simplify $\frac{225}{375}$:

$$= \frac{225 \div 75}{375 \div 75} = \frac{3}{5}$$

Thus, the common ratio is:

$$r = \frac{3}{5}$$

Step 3: Use the formula for the n-th term of a geometric sequence.
The n-th term of a geometric sequence can be calculated using the formula:

$$a_n = a_1 \cdot r^{n-1}$$

Step 4: Calculate the eighth term a_8.
Substituting $n = 8$ into the formula:

$$a_8 = a_1 \cdot r^{8-1} = \frac{125}{9} \cdot \left(\frac{3}{5}\right)^7$$

Step 5: Calculate $\left(\frac{3}{5}\right)^7$.
Calculating $\left(\frac{3}{5}\right)^7$:

$$\left(\frac{3}{5}\right)^7 = \frac{3^7}{5^7} = \frac{2187}{78125}$$

Step 6: Substitute back into the expression for a_8.
Now we compute:

$$a_8 = \frac{125}{9} \cdot \frac{2187}{78125}$$

Step 7: Multiply the fractions.

Prompt of Critique under Critic-CoT

To multiply:

$$a_8 = \frac{125 \cdot 2187}{9 \cdot 78125}$$

Calculating the denominator:

$$9 \cdot 78125 = 703125$$

Calculating the numerator:

$$125 \cdot 2187 = 273375$$

Thus:

$$a_8 = \frac{273375}{703125}$$

Step 8: Simplify the fraction $\frac{273375}{703125}$.
To simplify: First, calculate the GCD of the numerator and denominator. We can find that both 273375 and 703125 are divisible by 135.
Calculating:

$$\frac{273375 \div 135}{703125 \div 135} = \frac{2025}{5205}$$

Continuing to simplify, both 2025 and 5205 are divisible by 15:

$$\frac{2025 \div 15}{5205 \div 15} = \frac{135}{347}$$

This is now fully simplified because 135 and 347 have no common factors.
Final Answer:
Thus, the eighth term of the sequence is:

$$\boxed{\frac{135}{347}}$$

`</attempt>`

—
Notes:
- Please think step by step.
- Your reasoning should precede any claims or conclusions you make to avoid unwarranted assertions.
- At the end of the evaluation for each step, YOU MUST articulate the conclusion using the format ”Conclusion: Step [i] is correct" or ”Conclusion: Step [i] is incorrect". Words like ”partially correct" are prohibited.

Response of Critique under Critic-CoT

Step 1: The first term $a_1 = \frac{125}{9}$ is identified correctly.
Conclusion: Step 1 is correct.
Step 2: The common ratio r is calculated as:

$$r = \frac{a_2}{a_1} = \frac{\frac{25}{3}}{\frac{125}{9}} = \frac{25}{3} \times \frac{9}{125} = \frac{225}{375} = \frac{3}{5}$$

This calculation is correct.
Conclusion: Step 2 is correct.

Response of Critique under Critic-CoT

Step 3: The formula for the n-th term is correctly stated as:

$$a_n = a_1 \cdot r^{n-1}$$

Conclusion: Step 3 is correct.
Step 4: The eighth term is computed as:

$$a_8 = \frac{125}{9} \cdot \left(\frac{3}{5}\right)^7$$

This is correct.
Conclusion: Step 4 is correct.
Step 5: Calculating $\left(\frac{3}{5}\right)^7$:

$$\left(\frac{3}{5}\right)^7 = \frac{2187}{78125}$$

This calculation is correct.
Conclusion: Step 5 is correct.
Step 6: Substituting back:

$$a_8 = \frac{125}{9} \cdot \frac{2187}{78125}$$

This is correct.
Conclusion: Step 6 is correct.
Step 7: Multiplying the fractions:

$$a_8 = \frac{125 \cdot 2187}{9 \cdot 78125}$$

The denominator calculation is correct:

$$9 \cdot 78125 = 703125$$

However, for the numerator:

$$125 \cdot 2187 = 273375$$

This is also correct.
Conclusion: Step 7 is correct.
Step 8: The goal is to simplify the fraction $\frac{273375}{703125}$. The attempt made to find the GCD and simplify. To simplify the fraction, we need to find the greatest common divisor (GCD) of 273375 and 703125. The standard approach involves either prime factorization or using the Euclidean algorithm.
Prime Factorization:
- 273375:
- Dividing by 5: $273375 \div 5 = 54675$
- Dividing by 5 again: $54675 \div 5 = 10935$
- Dividing by 5 again: $10935 \div 5 = 2187$
- The number 2187 is 3^7.
- Therefore, $273375 = 5^3 \cdot 3^7$.

- 703125:
- Dividing by 5: $703125 \div 5 = 140625$
- Dividing by 5 again: $140625 \div 5 = 28125$
- Dividing by 5 again: $28125 \div 5 = 5625$

Response of Critique under Critic-CoT

- Dividing by 5 again: $5625 \div 5 = 1125$
- Dividing by 5 again: $1125 \div 5 = 225$
- Dividing by 5 again: $225 \div 5 = 45$
- Finally, $45 = 3^2 \cdot 5^1$.
- Therefore, $703125 = 5^6 \cdot 3^2$.

The GCD can now be derived from the lowest powers of the common factors:
- For 5: 5^3
- For 3: 3^2

Thus, the GCD is:

$$5^3 \cdot 3^2$$

Therefore, 273375 and 703125 are not divisible by 135.
Conclusion: Step 8 is incorrect.

Prompt of Refinement under Critic-CoT

How do you refine the following attempt with respect to the problem, given the criticism?

<problem>
Consider the geometric sequence $\frac{125}{9}, \frac{25}{3}, 5, 3, \ldots$. What is the eighth term of the sequence? Express your answer as a common fraction.
</problem>

<attempt>
Step 1: Identify the first term of the sequence.
The first term a_1 is:

$$a_1 = \frac{125}{9}$$

Step 2: Calculate the common ratio r.
The common ratio r can be found by dividing the second term a_2 by the first term a_1:

$$r = \frac{a_2}{a_1} = \frac{\frac{25}{3}}{\frac{125}{9}} = \frac{25}{3} \times \frac{9}{125}$$

Calculating the right-hand side:

$$= \frac{25 \cdot 9}{3 \cdot 125} = \frac{225}{375}$$

Now simplify $\frac{225}{375}$:

$$= \frac{225 \div 75}{375 \div 75} = \frac{3}{5}$$

Thus, the common ratio is:

$$r = \frac{3}{5}$$

Step 3: Use the formula for the n-th term of a geometric sequence.
The n-th term of a geometric sequence can be calculated using the formula:

$$a_n = a_1 \cdot r^{n-1}$$

Prompt of Refinement under Critic-CoT

Step 4: Calculate the eighth term a_8.
Substituting $n = 8$ into the formula:

$$a_8 = a_1 \cdot r^{8-1} = \frac{125}{9} \cdot \left(\frac{3}{5}\right)^7$$

Step 5: Calculate $\left(\frac{3}{5}\right)^7$.
Calculating $\left(\frac{3}{5}\right)^7$:

$$\left(\frac{3}{5}\right)^7 = \frac{3^7}{5^7} = \frac{2187}{78125}$$

Step 6: Substitute back into the expression for a_8.
Now we compute:

$$a_8 = \frac{125}{9} \cdot \frac{2187}{78125}$$

Step 7: Multiply the fractions.
To multiply:

$$a_8 = \frac{125 \cdot 2187}{9 \cdot 78125}$$

Calculating the denominator:

$$9 \cdot 78125 = 703125$$

Calculating the numerator:

$$125 \cdot 2187 = 273375$$

Thus:

$$a_8 = \frac{273375}{703125}$$

Step 8: Simplify the fraction $\frac{273375}{703125}$.
To simplify: First, calculate the GCD of the numerator and denominator. We can find that both 273375 and 703125 are divisible by 135.
Calculating:

$$\frac{273375 \div 135}{703125 \div 135} = \frac{2025}{5205}$$

Continuing to simplify, both 2025 and 5205 are divisible by 15:

$$\frac{2025 \div 15}{5205 \div 15} = \frac{135}{347}$$

This is now fully simplified because 135 and 347 have no common factors.
Final Answer:
Thus, the eighth term of the sequence is:

$$\boxed{\frac{135}{347}}$$

</attempt>

Prompt of Refinement under Critic-CoT

<criticism>
Step 8: The goal is to simplify the fraction $\frac{273375}{703125}$. The attempt made to find the GCD and simplify.
To simplify the fraction, we need to find the greatest common divisor (GCD) of 273375 and 703125.
The standard approach involves either prime factorization or using the Euclidean algorithm.
Prime Factorization:
- 273375:
- Dividing by 5: $273375 \div 5 = 54675$
- Dividing by 5 again: $54675 \div 5 = 10935$
- Dividing by 5 again: $10935 \div 5 = 2187$
- The number 2187 is 3^7.
- Therefore, $273375 = 5^3 \cdot 3^7$.

- 703125:
- Dividing by 5: $703125 \div 5 = 140625$
- Dividing by 5 again: $140625 \div 5 = 28125$
- Dividing by 5 again: $28125 \div 5 = 5625$
- Dividing by 5 again: $5625 \div 5 = 1125$
- Dividing by 5 again: $1125 \div 5 = 225$
- Dividing by 5 again: $225 \div 5 = 45$
- Finally, $45 = 3^2 \cdot 5^1$.
- Therefore, $703125 = 5^6 \cdot 3^2$.

The GCD can now be derived from the lowest powers of the common factors:
- For 5: 5^3
- For 3: 3^2
Thus, the GCD is:

$$5^3 \cdot 3^2$$

Therefore, 273375 and 703125 are not divisible by 135.
Conclusion: Step 8 is incorrect.
</criticism>

Prompt of Refinement under Critic-CoT

<correction>
Step 8: Simplify $\frac{273375}{703125}$. First, we divide it by 375.
Calculating:

$$\frac{273375 \div 375}{703125 \div 375} = \frac{729}{1875}$$

Step 9: Further simplify $\frac{729}{1875}$. The numerator 729 is 3^6 and the denominator 1875 is $3 \cdot 625$. The GCD is 3.
Calculating:

$$\frac{729 \div 3}{1875 \div 3} = \frac{243}{625}$$

This fraction is fully simplified.
Final Answer: Thus, the eighth term of the sequence is:

$$\boxed{\frac{243}{625}}$$

</correction>

Table A14: Full example of Critic-CoT. The model got the wrong common factor in Step 8, and through CoT critique, identified this mistake, then took the refinement and reached the correct answer.

Prompt of Critique under Step-wise Label Critic

How do you evaluate the following attempt with respect to the problem?

<problem>
[...]
</problem>

<attempt>
[...]
</attempt>

—
Notes:
- For each step, you only need to output ”Step [i] is correct" if right, or ”Step [i] is incorrect" if wrong. Do not provide anything else in your output.

Response of Critique under Step-wise Label Critic

Step 1 is correct.
Step 2 is correct.
Step 3 is correct.
Step 4 is correct.
Step 5 is incorrect.

Prompt of Refinement under Step-wise Label Critic

How do you refine the following attempt with respect to the problem, given the criticism?
 	
`<problem>`
`[...]`
`</problem>`
 	
`<attempt>`
`[...]`
`</attempt>`
 	
`<criticism>`
Step 5 is incorrect.
`</criticism>`

Response of Refinement under Step-wise Label Critic

<correction>

Step 5: Calculate $\left(\frac{3}{5}\right)^7$.

$$\left(\frac{3}{5}\right)^7 = \frac{3^7}{5^7} = \frac{2187}{78125} \text{ (correct)}$$

Step 6: Substitute back into the expression for a_8.

$$a_8 = \frac{125}{9} \cdot \frac{2187}{78125}$$

Step 7: Multiply the fractions. Calculating the numerator:

$$125 \cdot 2187 = 273375 \text{ (correct)}$$

Calculating the denominator:

$$9 \cdot 78125 = 703125 \text{ (correct)}$$

Step 8: Simplify the fraction $\frac{273375}{703125}$. To simplify: Calculating the GCD shows that both 273375 and 703125 are divisible by 675, so:

$$\frac{273375 \div 675}{703125 \div 675} = \frac{405}{1045}$$

Continuing to simplify, both 405 and 1045 are divisible by 5:

$$\frac{405 \div 5}{1045 \div 5} = \frac{81}{209}$$

Thus, the final answer is:

$$\boxed{\frac{81}{209}}$$

</correction>

Table A15: Full example of Step-wise Label Critic. The problem and the attempt are the same in Table A14, so we skip them. Without CoT critique, the model incorrectly believed that Step 5 is wrong, then made the refinement starting from Step 5, which is unsuccessful.

Prompt of Critique under Outcome Label Critic

How do you evaluate the following attempt with respect to the problem?

<problem>
[...]
</problem>

<attempt>
[...]
</attempt>

—
Notes:
- For the whole attempt, you only need to output ”Each step from Step 1 to Step 8 is correct." if right, or ”Some step from Step 1 to Step 8 is incorrect" if wrong. Do not provide anything else in your output.

Response of Critique under Outcome Label Critic

Some step from Step 1 to Step 8 is incorrect.

Prompt of Refinement under Outcome Label Critic

How do you refine the following attempt with respect to the problem, given the criticism?

<problem>
[...]
</problem>

<attempt>
[...]
</attempt>

<criticism>
Some step from Step 1 to Step 8 is incorrect.
</criticism>

Response of Refinement under Outcome Label Critic

<correction>
Step 1: Identify the first term of the sequence.
The first term a_1 is:

$$a_1 = \frac{125}{9}$$

Step 2: Calculate the common ratio r.
The common ratio r can be found by dividing the second term a_2 by the first term a_1:

$$r = \frac{a_2}{a_1} = \frac{\frac{25}{3}}{\frac{125}{9}} = \frac{25}{3} \times \frac{9}{125} = \frac{225}{375}$$

Now simplify $\frac{225}{375}$:

$$= \frac{225 \div 75}{375 \div 75} = \frac{3}{5}$$

Response of Refinement under Outcome Label Critic

Thus, the common ratio is:

$$r = \frac{3}{5}$$

Step 3: Use the formula for the n-th term of a geometric sequence.
The n-th term of a geometric sequence can be calculated using the formula:

$$a_n = a_1 \cdot r^{n-1}$$

Step 4: Calculate the eighth term a_8.
Substituting $n = 8$ into the formula:

$$a_8 = a_1 \cdot r^{8-1} = \frac{125}{9} \cdot \left(\frac{3}{5}\right)^7$$

Step 5: Calculate $\left(\frac{3}{5}\right)^7$.
Calculating $\left(\frac{3}{5}\right)^7$:

$$\left(\frac{3}{5}\right)^7 = \frac{3^7}{5^7} = \frac{2187}{78125}$$

Step 6: Substitute back into the expression for a_8.
Now we compute:

$$a_8 = \frac{125}{9} \cdot \frac{2187}{78125}$$

Step 7: Multiply the fractions.
To multiply:

$$a_8 = \frac{125 \cdot 2187}{9 \cdot 78125}$$

Calculating the denominator:

$$9 \cdot 78125 = 703125$$

Calculating the numerator:

$$125 \cdot 2187 = 273375$$

Thus:

$$a_8 = \frac{273375}{703125}$$

Step 8: Simplify the fraction $\frac{273375}{703125}$.
To simplify: Finding the GCD, both 273375 and 703125 are divisible by 675:

$$\frac{273375 \div 675}{703125 \div 675} = \frac{405}{1042}$$

Now we check if $\frac{405}{1042}$ can be simplified further. Both numbers do not have common factors.
Final Answer:
Thus, the eighth term of the sequence is:

$$\boxed{\frac{405}{1042}}$$

</correction>

Table A16: Full example of Final Label Critic. The problem and the attempt are also the same in Table A14. With only the outcome label, the model managed to detect the attempt was incorrect, then made the refinement starting from the first step, which was unsuccessful.

Prompt of Code Generation on HumanEval

Python code problem

```
def split_words(txt):
    '''
    Given a string of words, return a list of words split on
    whitespace, if no whitespaces exists in the text you
    should split on commas ',' if no commas exists you should
    return the number of lower-case letters with odd order in the
    alphabet, ord('a') = 0, ord('b') = 1, ... ord('z') = 25
    Examples
    split_words("Hello world!") -> ["Hello", "world!"]
    split_words("Hello,world!") -> ["Hello", "world!"]
    split_words("abcdef") == 3
    '''
```

Before writing the code, think step by step, marking each step as "Step [i]:"
Your final function split_words should be in the form ```python\n[code]\n```, at the end of your response.

Response of Code Generation on HumanEval

Step 1: Create a function named `split_words` that takes a string `txt` as an input.
Step 2: Check if there are any whitespace characters in the string. If there are, split the string into words using the 'split()' function with the default separator (whitespace).
Step 3: If there are no whitespace characters, check if there are any commas in the string. If there are, split the string into words using the 'split()' function with the comma as the separator.
Step 4: If there are neither whitespace characters nor commas in the string, iterate over the characters in the string, keeping track of the count of lowercase letters with odd alphabetical order (based on their ASCII value modulo 2).
Step 5: Return the result based on the conditions met in steps 2, 3, or 4.
Here is the Python code for the function:
```python

```
def split_words(txt):
    if ' ' in txt:
        return txt.split()
    elif ',' in txt:
        return txt.split(',')
    else:
        count = sum(1 for char in txt if char.islower() and
        ord(char) % 2 != 0)
        return count
```

```

This function first checks for whitespace characters, then for commas, and finally counts the lowercase letters with odd alphabetical order if neither of the first two conditions is met.

**Prompt of Critique on HumanEval**

How do you evaluate the following attempt with respect to the problem?
<problem>

```python
def split_words(txt):
    '''
    Given a string of words, return a list of words split on
    whitespace, if no whitespaces exists in the text you
    should split on commas ',' if no commas exists you should
    return the number of lower-case letters with odd order in the
    alphabet, ord('a') = 0, ord('b') = 1, ... ord('z') = 25
    Examples
    split_words("Hello world!") -> ["Hello", "world!"]
    split_words("Hello,world!") -> ["Hello", "world!"]
    split_words("abcdef") == 3
    '''
```

</problem>

<attempt>
Step 1: Create a function named `split_words` that takes a string `txt` as an input.
Step 2: Check if there are any whitespace characters in the string. If there are, split the string into words using the 'split()' function with the default separator (whitespace).
Step 3: If there are no whitespace characters, check if there are any commas in the string. If there are, split the string into words using the 'split()' function with the comma as the separator.
Step 4: If there are neither whitespace characters nor commas in the string, iterate over the characters in the string, keeping track of the count of lowercase letters with odd alphabetical order (based on their ASCII value modulo 2).
Step 5: Return the result based on the conditions met in steps 2, 3, or 4.
Here is the Python code for the function:
```python
def split_words(txt):
Line 1:     if '␣' in txt:
Line 2:         return txt.split()
Line 3:     elif ',' in txt:
Line 4:         return txt.split(',')
Line 5:     else:
Line 6:         count = sum(1 for char in txt if char.islower() and
                ord(char) % 2 != 0)
Line 7:         return count
```
This function first checks for whitespace characters, then for commas, and finally counts the lowercase letters with odd alphabetical order if neither of the first two conditions is met.
</attempt>

Response of Critique on HumanEval

Step 1: Define a function called `split_words` that takes a string `txt` as its parameter.
Conclusion: Step 1 is correct.
Step 2: Verify if there are any whitespace characters in the string. If they exist, divide the string into words using the `split()` function with the default separator (whitespace).
Conclusion: Step 2 is correct.
Step 3: If no whitespace characters are found, check for commas in the string. If commas are present, split the string into words using the `split()` function with the comma as the separator.
Conclusion: Step 3 is correct.
Step 4: Otherwise, loop through the characters in the string, counting the number of lowercase letters that have an odd alphabetical order (based on their ASCII value modulo 2). The mistake in Step 4 lies in the interpretation of "odd alphabetical order." The problem statement refers to the alphabetical order of lowercase letters in the alphabet, not their ASCII values. The correct approach is to count the lowercase letters whose alphabetical order (where 'a' is 0, 'b' is 1, ..., 'z' is 25) is odd.
Conclusion: Step 4 is incorrect.

Prompt of Refinement on HumanEval

How do you refine the following attempt with respect to the problem, given the criticism? You shall write another complete Python function, in the format ```python\n[code]\n```.

```
<problem>
[...]
</problem>

<attempt>
[...]
</attempt>
```

<criticism>
Step 4: Otherwise, loop through the characters in the string, counting the number of lowercase letters that have an odd alphabetical order (based on their ASCII value modulo 2). The mistake in Step 4 lies in the interpretation of "odd alphabetical order." The problem statement refers to the alphabetical order of lowercase letters in the alphabet, not their ASCII values. The correct approach is to count the lowercase letters whose alphabetical order (where 'a' is 0, 'b' is 1, ..., 'z' is 25) is odd.
Conclusion: Step 4 is incorrect.
</criticism>

Response of Refinement on HumanEval

<correction>
Here is the corrected Python function:
```python
def split_words(txt):
    if ' ' in txt:
        return txt.split()
    elif ',' in txt:
        return txt.split(',')
    else:
        count = sum(1 for char in txt if char.islower()
        and (ord(char) - ord('a')) % 2 != 0)
        return count
```
This function correctly counts the lowercase letters with odd alphabetical order by subtracting the ASCII value of 'a' from the ASCII value of each lowercase letter to get its alphabetical order, and then checking if this order is odd. If the order is odd, it increments the count.
</correction>

Table A17: Full example of a single-turn refinement on HumanEval. The attempt made mistakes in Step 4 and Line 6, which use the ASCII value directly rather than subtract the value of 'a'. The critique detect this error, and made a successul refinement.