Small Language Models Need Strong Verifiers to Self-Correct Reasoning

Yunxiang Zhang[α*] **Muhammad Khalifa**[α] **Lajanugen Logeswaran**[β]
Jaekyeom Kim[β] **Moontae Lee**[βγ] **Honglak Lee**[αβ] **Lu Wang**[α]
University of Michigan[α] LG AI Research[β] University of Illinois at Chicago[γ]

Abstract

Self-correction has emerged as a promising solution to boost the reasoning performance of large language models (LLMs), where LLMs refine their solutions using self-generated critiques that pinpoint the errors. This work explores whether small ($\leq$ 13B) language models (LMs) have the ability of self-correction on reasoning tasks with minimal inputs from stronger LMs. We propose a novel pipeline that prompts smaller LMs to collect self-correction data that supports the training of self-refinement abilities. First, we leverage correct solutions to guide the model in critiquing their incorrect responses. Second, the generated critiques, after filtering, are used for supervised fine-tuning of the self-correcting reasoner through solution refinement. Our experimental results show improved self-correction abilities of two models on five datasets spanning math and commonsense reasoning, with notable performance gains when paired with a strong GPT-4-based verifier, though limitations are identified when using a weak self-verifier for determining when to correct.[1]

1 Introduction

Recent research shows that large language models (LLMs) (OpenAI, 2023) can self-correct their responses to meet diverse user requirements, ranging from diminishing harmful content to including specific keywords and to debugging code (Madaan et al., 2023; Chen et al., 2023b). Self-correction is typically accomplished by first generating a critique that identifies the shortcomings of the initial response, followed by revising it according to the self-critique—a process that can be iterated.

Self-correction has emerged as an intriguing paradigm for rectifying the flaws in LLM's outputs (Pan et al., 2023). However, models that are effective at self-correction are of very large sizes, and many of them are proprietary and accessible only via APIs. In this work, we focus on the self-correction abilities of small, open-source language models (LMs).[2] Previous studies have shown that these smaller models can learn self-correction in reasoning through distillation from stronger LMs (Yu et al., 2023b; An et al., 2023; Han et al., 2024). Yet this poses security risks for high-stakes domains and hinders the scientific understanding of enhancing LMs' ability to correct errors. We thus ask the question: *To which degree do small LMs require guidance from strong LMs to learn self-correction for reasoning?*

We study this question by leveraging the small model itself to generate supervised training data to enhance its self-correction ability, instead of resorting to stronger LMs. To this end, we draw inspiration from the rejection sampling fine-tuning (RFT) (Touvron et al., 2023; Yuan et al., 2023) method where LLM's reasoning skills are bootstrapped via diverse chain-of-thought sampling and supervised fine-tuning on the correct reasoning chains. We propose **SCORE**—an approach to bootstrap small LMs' **S**elf-**CO**rrection ability in **RE**asoning tasks. Concretely, we devise a pipeline for accumulating **high-quality critique-correction** data from small LMs, which are used for supervised fine-tuning of self-correcting reasoners. First, we leverage correct solutions as hints for the base LMs to critique incorrect answers. By reverse-engineering from the correct answer, the models generate more effective critiques. Second, we filter these critiques for correctness, well-formedness, and clarity using simple rule-based and prompting methods. Finally, we fine-tune the same LMs to be-

[*] Correspondence to `yunxiang@umich.edu`

[1]Our implementation can be accessed at `https://github.com/yunx-z/SCORE`.

[2]While the distinction between small vs. large LMs is often context-dependent (Saunders et al., 2022; Yu et al., 2023b), in this work, we interchangeably use "small" or "weak" LMs to refer to open models with a few billion parameters (e.g., LLaMA-7/13B (Touvron et al., 2023)).

come **self-refining** models using this curated data. By avoiding the use of supervision from stronger LMs, we ensure that our method enables a small LM to bootstrap its self-correction capabilities.

We evaluate our SCORE fine-tuned refiner under both extrinsic and intrinsic self-correction settings (Huang et al., 2023b). The primary difference between these two settings is whether the refiner is allowed to use *external* signals to determine *when* to self-correct (i.e., refine the initial solution only when it is believed to be incorrect). Identifying *when* to self-correct involves verifying the solutions' correctness, which is still challenging for current state-of-the-art LLMs without proper external feedback (Huang et al., 2023b). We adopt a simple baseline for the self-verification problem following Cobbe et al. (2021). Specifically, we fine-tune the same LMs to become verifiers with labels based solely on the correctness of the final answer, conditioning on the question and a candidate solution. As for the extrinsic setting, we simulate strong verifiers with GPT-4 and oracle labels, to show the effectiveness of small LMs as self-correcting reasoners.

We test the SCORE method with the `LLaMA-2-13B-chat` (Touvron et al., 2023) and `Gemma-7b-it` (Team et al., 2024) models on five datasets spanning math and commonsense reasoning. We find that our model with SCORE fine-tuning outperforms the original model by an average of 14.6% when using a `gpt-4`-based verifier. Nevertheless, the model struggles with self-correction when subjected to a weak self-verifier fine-tuned on self-generated solutions.

Our main contributions are summarized below:

1. We introduce SCORE, a novel pipeline to generate self-correction data from a small LM, and subsequently fine-tune the model to be a self-correcting reasoner.

2. Our method effectively augments the self-correction abilities of small LMs on math and commonsense reasoning, when using strong verifiers.

3. To the best of our knowledge, we are the first to demonstrate the potential of small LMs to bootstrap their abilities on self-corrective reasoning *without* distilling training data from stronger LMs or using human annotation.

2 Problem Formulation of Self-Correction

Self-Correct := (Self-)Verify + Self-Refine. We decompose the task of self-correction into two phases: (Self-)Verify and Self-Refine. The LM first generates an initial solution for a reasoning question. A verifier, either the LM itself (intrinsic) or the external signal (extrinsic), then judges the correctness of the initial solution. If correct, the initial solution will be directly used as the final answer. If incorrect, a refiner will revise the solution. While this process can be iterated, we fix the times of iterations as 1 throughout this paper for efficiency and leave multiple iterations as future studies.

Decoupling (Self-)Verify and Self-Refine brings two major advantages over a one-model-does-all design. First, we can freely parameterize each module—for example, by using a fine-tuned and a few-shot prompted model. This allows us to carefully examine the impact of strong vs. weak verifiers on the refiners' performance. On the contrary, previous work on self-correction with small LMs (Yu et al., 2023b; An et al., 2023; Han et al., 2024) conflates Self-Verify and Self-Refine, creating a barrier to fully understanding the distinct capacities of these models in each skill. Second, it reduces the difficulty of training each module, since the model only needs to specialize in one kind of ability, which is either verification or refinement.

Self-Refine := Critique + Correction. The challenge for Self-Refine is that it can be difficult for language models to directly map an initial solution to a revision without any guidance (Welleck et al., 2023). Using **critiques**—assessments that pinpoint the locations of errors within the reasoning steps, explain the causes of these errors, and offer guidance on how to correct them—can significantly enhance the performance of language models when generating revisions (Saunders et al., 2022; Madaan et al., 2023). Therefore, we formulate refinement with two steps: the model will first generate a critique for the initial solutions determined as incorrect, followed by a corrected version, in a single pass. Yet, it is still non-trivial to obtain high-quality critiques to guide the error correction. We address this problem using the correct solutions as hints to facilitate the critique generation, detailed in Section 3.1.

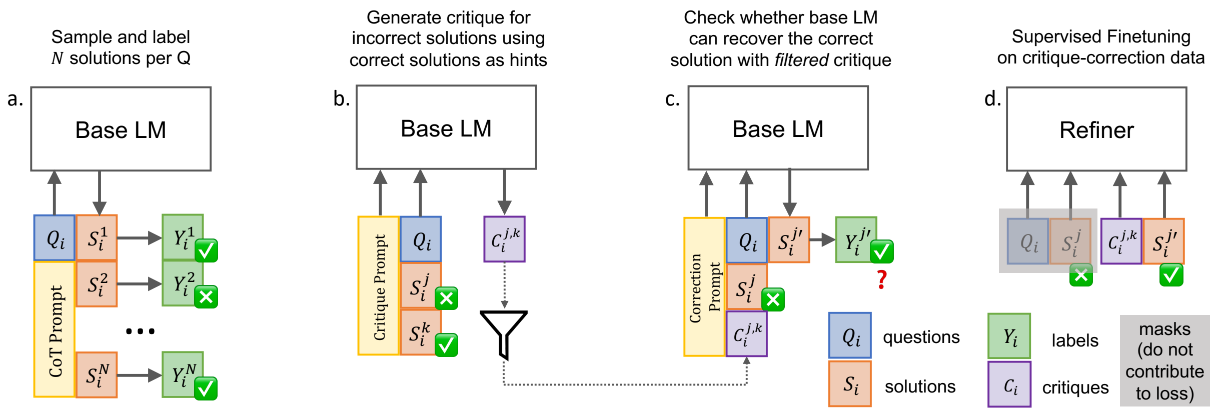

Figure 1: Illustration of the SCORE pipeline to generate critique-correction data from a small LM (step a-c) and fine-tune the same LM to self-correct its reasoning errors (step d), *without* distilling any data from stronger LMs.

3 The SCORE Method

Our approach is inspired by rejection sampling fine-tuning (RFT): sampling diverse solutions for each question and fine-tune LLMs on the self-generated solutions that lead to the correct final answer (Yuan et al., 2023; Huang et al., 2023a; Zelikman et al., 2022). We want to bootstrap the small LM's inherent ability to generate critiques for reasoning steps. We design an end-to-end pipeline to collect self-correction data generated by small LMs at scale, without any distillation from stronger LMs. The self-generated critiques, after filtering, are used to fine-tune the smaller LM itself to bootstrap its ability to self-correct. Concretely, the SCORE pipeline consists of two stages shown in Figure 1 and described below.

Stage 1: Generating and Filtering Critiques. We sample N solutions for each question in the training set by few-shot chain-of-thought prompting a base LM (step a). To enable the base LM to reflect on its incorrect solutions, we include a correct solution for the same question (if exists) in the prompt as a hint (step b). We then filter the self-generated critiques based on their correctness and clarity (step c). This process is detailed in Section 3.1.

Stage 2: Supervised Fine-tuning of the Refiner. The filtered critiques obtained from stage 1 are used in the next stage for fine-tuning the small LM itself. We train a refiner that generates critiques and corrections conditioned on questions and initial solutions (step d). We exclude the hints during fine-tuning. More details are given in Section 3.2.

3.1 Generating and Filtering Critiques

Directly generating critique for an incorrect solution without external supervision signals is difficult. In our preliminary experiments, we find it easier for the LM to generate critiques **using correct solutions as hints**, as the model only needs to compare the different steps between these two solutions and justify the correct ones. In Appendix B, we explain this intuition from a mathematical perspective.

To leverage correct solutions as hints for LMs to generate critiques on incorrect solutions, we label these solutions and collect all possible pairs of incorrect-correct solutions for the same questions (Cartesian product between the sets of incorrect and correct solutions). We craft a few-shot critique prompt (Appendix A) to instruct the base LM to generate critiques for the incorrect solution using the paired correct solution as hints.[3] Step-level critiques are more useful than solution-level ones since they provide more precise and fine-grained supervision (Lightman et al., 2023; Wu et al., 2023; Uesato et al., 2022a) that mitigate the undesirable behavior of LMs using incorrect reasoning to reach the correct final answer (Khalifa et al., 2023; Zelikman et al., 2022). Therefore, we prompt the model to provide feedback for each step of the initial solution, either endorsing the initial answer (e.g., "*this step is correct*") or pinpointing the errors (e.g., "*there are errors in the step because ...*"). To ensure the LM-generated critique is grounded in a specific step, we also ask the model to copy each step before providing feedback on it. Considering

[3]The total number of incorrect-correct solution pairs could be very large so we sample only one critique per pair. This has already provided a sufficient amount of SCORE fine-tuning data after filtering.

these requirements, we design the format of the critique prompt as follows, with a detailed example in Appendix A.

Critique Prompt

```
Q: {question}

Answer 1 (Incorrect):
Step 1: ...
...
Step n: The answer is [x].

Answer 2 (Correct):
Step 1: ...
...
Step n: The answer is [y].

There are reasoning errors in Answer 1. Please go through each step in Answer 1, use Answer 2 as a reference for the correct approach, and provide feedback that helps correct the errors in Answer 1. End your response with [END].

Let's go through the errors in Answer 1 and provide feedback:

Answer 1 (Incorrect):

Step 1: ...
Feedback: This step is correct.
...
Step i: ...
Feedback: This is incorrect. Because ...
...
Step n: The answer is [x].
Feedback: The correct answer, based on the corrected calculations, should be [y].
[END]
```

Note that the model should suggest the corrected final answer, taken from the hint solution, as part of the feedback for the last step. This forces the model to explicitly leverage the information from the hint solution.

Filtering Generated Critiques. After obtaining the raw self-generated critiques, we want to remove the low-quality ones and keep the rest for fine-tuning LMs. Thanks to the well-designed format of critiques, we can apply rule-based filters to remove generated critiques that do not follow the desired format. These criteria include:

- The number of steps and feedbacks (counted by the appearances of "`Step {i}:`" and "`Feedback:`") should be the same.
- Each step should be exactly copied from the initial solution.
- The feedback for the last step should provide the correct answer.

The first two criteria check for the well-formedness of the critique and the third one focuses on the correctness aspect. A critique will be removed if it fails to meet any of the three criteria above.

Given that a critique could still contain errors even if it suggests the correct final answer in the last step, we add an additional stage of prompting-based filtering besides the above rule-based heuristics. Specifically, we prompt the base LM to revise the incorrect solution given the critique that already passes the aforementioned filtering rule. Assuming the base LM has reasonable ability of following instructions, it is expected to give a correct revision if the generated critique is both clear and error-free. We demonstrate such an example of the correction prompt in Appendix A. In other words, we remove critiques that do not result in a correctly revised answer. After the ruled-based and prompting-based filtering, we obtain the high-quality critiques for fine-tuning LMs to self-refine.

3.2 Supervised Fine-tuning of the Refiner

We train the refiner to generate a critique and an improved solution in one pass conditioned on a question and an initial solution. We note that although we provide the correct solutions as hints to generate critiques during data collection, the model is tasked to generate critiques *without* the hints during fine-tuning and inference. Previously we collected critiques for every step to ensure that we can apply multiple filters to obtain high-quality critiques. But in this step, we truncate the critiques to only keep the feedback for the first error step as the fine-tuning target. This is because it is difficult to ask the LMs to identify and correct all the errors in one pass (Yu et al., 2023b) without referring to correct solutions as hints during inference. The refiner is fine-tuned on truncated critiques and corrections collected in the previous stages with cross-entropy loss. We do not include few-shot demonstrations during fine-tuning. We apply masks on the input tokens so that they do not contribute to the loss. Although we only do 1 iteration for the refinement in this work, we later show that small LMs can already achieve great improvement after 1 round of self-correction when paired with a strong verifier.

	GSM8K		CSQA	
	#	%	#	%
Base LM: `LLaMA-2-13b-chat`				
Raw critiques	56,843	100.0	42,705	100.0
After rule-based filering	36,337	63.9	36,436	85.3
After prompting filtering (for SCORE fine-tuning)	14,499	25.5	24,511	57.4
Base LM: `Gemma-7b-it`				
Raw critiques	52,669	100.0	40,604	100.0
After rule-based filering	17,209	32.7	35,929	88.5
After prompting filtering (for SCORE fine-tuning)	4,623	8.8	12,972	31.9

Table 1: Statistics of the critique data generated from our SCORE pipeline. Although Gemma-7B has fewer data left after filtering, it still achieves greater improvement than LLaMA-13B by self-correction (Section 5.1), suggesting that Gemma-7B is more effective at learning self-correction from SCORE.

4 Experimental Setup

Self-Correction Data Collection. As stated in Section 3.1, we sample $N = 10$ solutions from the base model with the chain-of-thought (CoT) prompts shown in Appendix A, label their correctness, and formulate incorrect-correct solution pairs for critique generation. We separately collect data for each base LM and task. This results in the number of raw critiques shown in Table 1. We sequentially apply rule-based filtering and prompting-based filtering to obtain high-quality critiques for the fine-tuning data.

Verifiers. We experiment with verifiers of different levels of capabilities to gauge their impacts on self-correction performance. First, we adopt a simple baseline for training a **self-verifier** following Cobbe et al. (2021). The self-verifier is a model with the same architecture as the base LM, conditioned on the question and a candidate solution to judge the probability that the solution is correct/incorrect. Specifically, we label the solutions sampled from the base LM as incorrect (0) or correct (1) solely based on their final answers and fine-tune the verifier with a binary classification head on the last-layer representation of the last token in the input sequence "`Question: {q} \n Solution: {s} \n Is this solution correct?`". Since the fine-tuning data is imbalanced between correct and incorrect solution, we re-weight the loss for each class with regarding to its proportion. During inference, the verifier model outputs a probability of the initial solution being incorrect, and the refinement is introduced only when the confidence of the verifier's predictions exceeds a certain threshold, which is automatically chosen in a way that maximizes the accuracy on the dev set and then fixed during test-time predictions. Since fine-tuned small LMs are still weak verifiers that bottleneck the performance of self-correction, we also experiment with a second option by using **`gpt-4`** as an off-the-shelf strong verifier to demonstrate the potential of our fine-tuned refiner. We do so by few-shot prompting `gpt-4` to predict the correctness of the initial solution by smaller LMs, with the verifying prompt shown in Appendix A. Finally, we directly use the gold labels of the initial solutions as signals to determine when to self-refine. This **oracle** verifier setting provides an upper bound for the refiners' performance.

Benchmarks and Base Models. To demonstrate the effectiveness of the SCORE method, we conduct experiments on two popular datasets: **GSM8K** (Cobbe et al., 2021) for mathematical reasoning and **CommonsenseQA** (Talmor et al., 2018) for commonsense reasoning. We also conduct transferability studies and evaluate the generalization performance of our fine-tuned refiner on **MATH** (Hendrycks et al., 2021) for mathematical reasoning, **QASC** (Khot et al., 2020) and **RiddleSense** (Lin et al., 2021) for commonsense reasoning. Specifically, for mathematical reasoning, we train self-verifiers and SCORE refiners using only GSM8K training data and evaluate them on the whole GSM8K test set and a subset of MATH test set,[4] following the practice of Hosseini et al. (2024). Similarly, for commonsense reasoning, we fine-tune our models using only CommonsenseQA training data and evaluate them on the whole dev[5] set of CommonsenseQA, QASC, and RiddleSense. Since questions in CommonsenseQA, QASC, and Riddlesense have a multiple-choice format, we also include a random refiner baseline that randomly picks a choice different from the initial answer, following the practice of Huang et al. (2023b).

We explore two open-source smaller language models, namely `LLaMA-2-13B-chat` (Touvron et al., 2023) and `Gemma-7B-it` (Team et al., 2024)

[4]This subset includes a total 181 problems of Level 1 difficulty in MATH with question types of algebra, Counting & probability, prealgebra and number theory, where the final answer is a number and no latex exists in the question.

[5]The test labels of these datasets are hidden, so we use the original dev set as our test set, following Kojima et al. (2022); Kim et al. (2023).

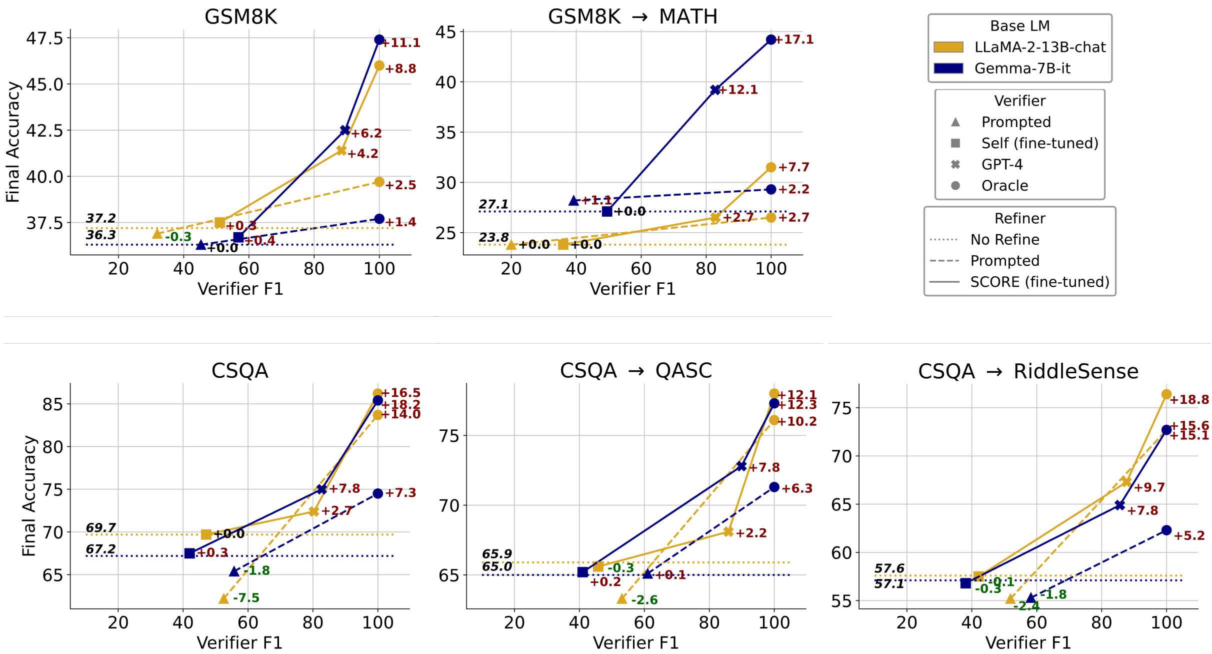

Figure 2: Performance of SCORE models using **LLaMA-2-13B-chat** and **Gemma-7B-it** as base LMs. We show the relationship between F1 scores of the verifiers and final answer accuracies. We also report the difference between the final accuracy after refinement and the corresponding initial answer accuracy (*Init. Accu.*) by few-shot prompting for each base LM. For the concrete numbers of these metrics, please see Table 6 in the appendix. We include test results for training tasks (GSM8K and CommonsenseQA/CSQA), as well as transfer evaluation of GSM8K trained models on MATH subset, CSQA trained models on QASC and RiddleSense. All models use greedy decoding during inference.

as the base LMs to generate self-correction data and evaluate their self-correction abilities. In Appendix C, we also investigate whether our self-correction fine-tuning can be built on top of other fine-tuning methods (e.g., rejection-sampling fine-tuning) to further boost the reasoning performance.

Fine-tuning and Evaluation. We fine-tune the base LM using the LLaMA-Factory library (Zheng et al., 2024b) with LoRA (Hu et al., 2022). We set the low-rank dimension as 32, the learning rate as 2e-5, training epochs as 3, batch size as 32. During inference, we set the temperature as 0 (i.e., greedy decoding) and the max sample length as 2,048. All our experiments can be conducted on 4×A40 GPU with 48GB of memory.

5 Results

In this section, we first present the experimental findings of SCORE method on various models and datasets (Section 5.1). To better understand the performance changes after self-correction, we then analyze the behaviors of verifiers and refiners (Section 5.2) and further highlight several key design decisions of our pipeline with ablation studies (Section 5.3). Lastly, we show the impact of SCORE fine-tuning data size on self-correction performance (Section 5.4).

5.1 Main Findings

Figure 2 presents the primary evaluation results for our fine-tuned models compared to baseline models. The results include two performance metrics: the verifier F1, which assesses the precision and recall of the verifier's predictions; and the final accuracy, which measures the accuracy of the final answer after self-correction. We have four major findings.

1) The critique-correction data collected by our SCORE *pipeline enhances the base LM's capability for self-correction.* Our fine-tuned models consistently bring large improvements on the final accuracy over the initial answer obtained by few-shot prompting. However, the prompting-based self-correction baseline (prompted verifier + prompted refiner in Figure 2) proposed by Madaan et al. (2023) deteriorates the final predictions, as LMs struggle to identify errors in their reasoning (Huang et al., 2023b) and possess limited self-correction abilities before bootstrapping. On the multiple-

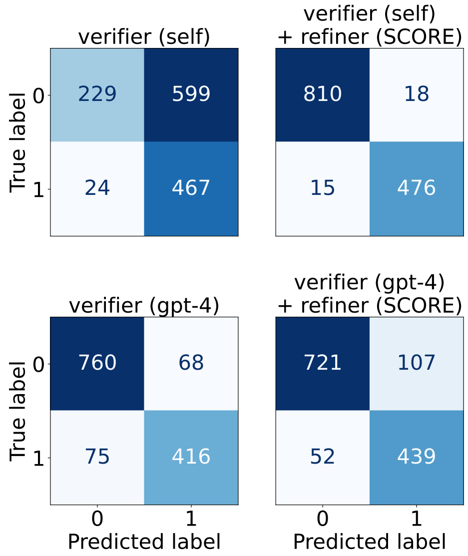

Figure 3: Confusion matrices of the predictions by the verifier and the refiner on GSM8K test set. The base LM is `LLaMA-2-13B-chat`. "True label" means the correctness of the initial solution. The predicted label of the verifier represents whether the verifier judges it as correct (1) or incorrect (0). The predicted label of the verifier + refiner is the correctness of the final answer. The strong verifier (`gpt-4`) makes fewer false positive predictions than the weak self-verifier and unleashes the potential of the small LM to revise an incorrect answer into a correct one more likely than the other way around.

choice CommonsenseQA questions, our SCORE fine-tuned refiner achieves much larger improvement than the random baseline under oracle verifier, indicating that our model is not simply making random guess.

2) Our framework improves self-correction for various base LMs on different types of reasoning tasks. We validate the effectiveness of our SCORE fine-tuning on both math reasoning and commonsense reasoning tasks with two pretrained LMs. In principle, our task-agonistic pipeline can be applied to a variety of datasets whose reasoning could be expressed in a step-by-step format. We also observe that although the initial solutions proposed by Gemma-7B are worse than LLaMA-13B (e.g., 67.2 < 69.7 on CommonsenseQA), Gemma-7B's accuracy surpasses LLaMA-13B after self-correction (e.g., 75.0 > 72.4 on CommonsenseQA). Considering that Gemma-7B is fine-tuned with even less self-correction data (Table 1), we believe Gemma is more effective at learning self-correction skills from SCORE fine-tuning.

3) The self-correction performance is largely bottlenecked by the verifier rather than the refiner. Using the same fine-tuned refiner, the final accuracies vary a lot among different verifiers. The upper bound performance suggested by an oracle verifier demonstrate great potential for self-correction, yet a weak self-verifier can only bring minor improvements, if not misguiding the refiner. Nevertheless, when combined with a more advanced verifier, such as GPT-4, our refiner achieves a significant increase in final accuracy, e.g., an average of +8.3 across five datasets for `Gemma-7b-it`. The confusion matrices in Figure 3 show the system of gpt-4-as-verifier + SCORE-refiner is more likely to modify an incorrect answer to a correct one than the other way round. This observation underscores the necessity of effectively tackling the problem of reasoning verification before significant advances in self-correction can be attained. Future work could focus on the improvement of reasoning verification that is built upon a mechanistic (Yüksekgönül et al., 2023; Yang et al., 2024) and representational (Zou et al., 2023; Zheng et al., 2024a) understanding of LMs' internal reasoning process.

4) The enhanced self-correction skills can transfer across different datasets. When evaluating our fine-tuned refiner on unseen datasets, it still demonstrates consistent improvement over the baselines (up to +12.1 by the GSM8K-trained Gemma-7B on MATH subset). This shows that the model is learning generalizable self-correction skills rather than overfitting to a specific dataset. Additionally, we find that the verifier does not transfer as well as the refiner, reiterating the difficulty of reasoning verification for LMs.

SCORE could be combined with oversample-then-rerank, an orthogonal approach to further improve reasoning (Cobbe et al., 2021; Li et al., 2023b; Hosseini et al., 2024). By aggregating verifications from multiple solutions, the weak verifiers become stronger via ensembling and thus unleash the power of refiner for better self-correction of reasoning. Please refer to Appendix D for details.

5.2 Analysis of Self-Correction Behaviors

Following the methodology of Yu et al. (2023b), we focus on two key metrics to understand the model's self-correction behaviors: 1) the frequency with which the verifier decides to self-correct (Freq.), and 2) the extent to which these self-correction attempts enhance the model's task performance (Contrib.). Self-correction Freq. is measured by the ratio of self-correction attempts to the size of the test set, while self-correction Contrib. is deter-

Verifier	Refiner	GSM8K Freq.	GSM8K Contrib.	CSQA Freq.	CSQA Contrib.
Base LM: `LLaMA-2-13b-chat`					
prompted	prompted	3.7	10.2	17.5	19.6
self	prompted	2.7	2.9	1.2	40.0
oracle	prompted	62.8	4.0	30.3	46.2
self	SCORE (fine-tuned)	19.0	10.8	3.0	33.3
`gpt-4`	SCORE (fine-tuned)	63.3	15.6	38.8	40.9
oracle	SCORE (fine-tuned)	62.8	14.0	30.3	54.3
Base LM: `Gemma-7b-it`					
prompted	prompted	18.7	21.5	20.2	45.3
self	prompted	9.9	9.9	2.9	42.9
oracle	prompted	63.7	2.1	32.8	22.4
self	SCORE (fine-tuned)	27.9	14.1	0.6	57.1
`gpt-4`	SCORE (fine-tuned)	63.4	17.1	38.6	48.4
oracle	SCORE (fine-tuned)	63.7	17.4	32.8	55.6

Table 2: Analysis of self-correction behaviors. The settings are the same as those in Figure 2. Freq. (in percentage) means the frequency with which the verifier decides to self-correct. Contrib. (in percentage) refers to the extent to which these self-correction attempts enhance the model's task performance. A strong verifier (e.g., `gpt-4`) maintains a balanced frequency of self-correction, ideally similar to that of the oracle verifier, and thus enhance the final accuracies in Figure 2. Additionally, the SCORE refiner possesses stronger refinement capabilities, as indicated by a higher contribution score than the prompted refiner.

mined by the number of instances in which these attempts successfully resulted in the correct answer.

Table 2 presents a detailed analysis of the model's self-correction behaviors. Our analysis demonstrates that our fine-tuned refiner has a higher contribution to the final self-correction performance, explaining why it outperforms the prompting-based refiner (Madaan et al., 2023), as shown in Figure 2. Additionally, we find that our fine-tuned verifier and the `gpt-4` verifier maintain a more reasonable frequency of self-correction, striking a better balance between correction attempts and accuracy.

5.3 Ablation Studies

In order to validate the various design decisions made in constructing our pipeline, we have conducted a series of ablation studies. The key findings from Table 3 can be summarized as follows. 1) It is easier for the LM to identify only the first erroneous step, as the performance drops if we challenge it to critique every step. 2) There is no need to separate the SELF-REFINE process into two modules—one for generating critiques and another for corrections.

Critique for *only* the first error step?	Generating critique & correction in *one* pass?	Final Accu.	Oracle Accu.
Initial answers by few-shot prompting		39.4	39.4
✓	✓	**40.2**	**49.5**
✗	✓	39.6	46.4
✓	✗	39.4	43.6
✗	✗	39.9	47.0

Table 3: Ablation study on the format of critique and decoupling critique and correction generation. Results are shown on GSM8K dev set with `LLaMA-2-13B-chat` as base LM.

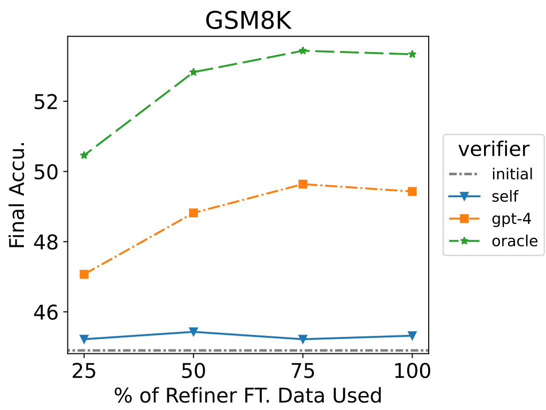

Figure 4: Final accuracy on GSM8K dev set w.r.t percentage of refiner fine-tuning data used and the type of verifier. The base LM is `LLaMA-2-13B-chat`. Under a strong verifier setting, our refiner brings greater performance gain with more fine-tuning data. Yet the performance plateaus with a weak verifier.

Such a separation not only increases the system's complexity and delays inference but also leads to a diminished final accuracy.

5.4 Scaling with Fine-tuning Data Size

We investigate the data-efficiency for refiner fine-tuning. Figure 4 plots the size of the refiner fine-tuning data against the final accuracy with different verification settings on GSM8K. We fine-tune the `LLaMA-2-13B-chat` base model on a random subset (varying from 25% to 75%) of the 14,499 total critique-corrections as previously shown in Table 1. We find that our refiner benefits from more fine-tuning data when paired with strong verifiers (oracle labels or `gpt-4`). Yet this effect is not observed when using a weak self-verifier, again highlighting the importance of verification for self-correction. We find that increasing the fine-tuning dataset size yields accuracy improvements up to a certain point. For instance, beyond approximately 10k examples (representing 75% of the SCORE fine-tuning data), the performance does not further improve.

6 Related Work

Training Small Language Model to Self-Correct. Recent work shows that smaller language model can be fine-tuned on task-specific data to perform self-correct. But existing methods rely either on distilled data from stronger models (An et al., 2023; Yu et al., 2023b; Han et al., 2024; Ye et al., 2023; Zhang et al., 2024) or template-based critiques (Paul et al., 2023; Welleck et al., 2023). Our approach differs from prior studies in this domain as we gather natural language critiques from a small language model without relying on larger models or task-specific heuristics. Furthermore, we split the self-correction process into two phases: (SELF-)VERIFY and SELF-REFINE. This separation contrasts with earlier approaches that often merge the two skills, which not only obscures the true abilities of these models in each respective skill but also complicates the training process. In a nutshell, we demonstrate that strong verifiers unleash the power of small LMs to SELF-REFINE.

Bootstrapping Reasoning in Language Models. As language models become more powerful, human supervision may not be sufficient to improve these models. This trend calls for self-improving LMs that provide training signals for themselves (Zelikman et al., 2022; Gülçehre et al., 2023; Yuan et al., 2023; Wang et al., 2023; Chen et al., 2024). The bootstrapping methods often involve iteratively fine-tuning a base LM on its self-generated examples that obtain a high reward value for correctness, helpfulness, or other desired properties. The bootstrapping process can further leverage label-free data (Huang et al., 2023a; Li et al., 2023a; Yuan et al., 2024) by generating pseudo labels using LLMs themselves. We draw inspiration from this family of methods and bootstrap the self-correction ability of smaller LMs. Our method is complementary to the rejection-sampling finetuning approach and can further improve reasoning performance upon that.

Verifying Reasoning. Verification of reasoning chains involves judging the correctness of the final answer and each reasoning steps. The verifier is often used to rerank multiple over-generated solutions and select the best one as the final output (Cobbe et al., 2021), or guide the LLM decoding through the search space for correct reasoning paths (Khalifa et al., 2023). We leverage a verifier to determine when to self-correct. Verifiers come at different granularity, including process/step-based (Uesato et al., 2022b; Li et al., 2023b; Lightman et al., 2023) and outcome-based supervision (Cobbe et al., 2021; Yu et al., 2023a; Hosseini et al., 2024). We use the latter since it is easier to construct labels automatically. Besides training a verifier with supervision signals, LLMs can also be few-shot prompted to become a verifier (Weng et al., 2023; Madaan et al., 2023; Zhou et al., 2023; Asai et al., 2023). We also explores the possibility of LLM-as-verifier, and demonstrate its usage for self-correction. We highlight the importance of verification in the context of self-correction, which echoes the recent finding that LLMs can successfully solve a problem, but cannot verify the reasoning (Gu et al., 2024; Oh et al., 2024; West et al., 2023). This calls for more effort for building evaluation benchmarks (Jacovi et al., 2024; Chen et al., 2023a; Mao et al., 2024; Lightman et al., 2023) and developing methods (Nguyen et al., 2024; Xu et al., 2024; Hosseini et al., 2024) to improve the reasoning verification.

7 Conclusion

In this study, we investigate how to leverage minimal signals from strong LMs to teach small LMs to self-correct their reasoning. We propose the SCORE method to collect self-correction fine-tuning data solely from small LMs. We find that SCORE-fine-tuned small LMs become better refiner models without relying on knowledge distillation from stronger LMs, yet they still need strong verifiers to be successful at self-correcting their reasoning errors. Our results highlight that the self-verification limitation of LMs currently poses an obstacle to the advancement of intrinsic self-correction in reasoning and thus warrants future research.

Limitations

Generating large amounts of synthetic data from smaller LMs requires intensive GPU computations, yet it removes the reliance on proprietary API models. Comparing the cost-efficiency of these two approaches will help us better trade-off between data generated from smaller LMs and larger LMs. Introducing the verifier and refiner during inference also causes additional latency, which we discuss in Appendix E.

Acknowledgements

This work is supported by LG AI Research. We also thank ARR reviewers for their useful feedback.

References

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. 2023. Learning from mistakes makes LLM better reasoner. *CoRR*, abs/2310.20689.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *CoRR*, abs/2310.11511.

Shiqi Chen, Yiran Zhao, Jinghan Zhang, I-Chun Chern, Siyang Gao, Pengfei Liu, and Junxian He. 2023a. FELM: benchmarking factuality evaluation of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023b. Teaching large language models to self-debug. *CoRR*, abs/2304.05128.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. Self-play fine-tuning converts weak language models to strong language models. *CoRR*, abs/2401.01335.

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

Alex Gu, Wen-Ding Li, Naman Jain, Theo X. Olausson, Celine Lee, Koushik Sen, and Armando Solar-Lezama. 2024. The counterfeit conundrum: Can code language models grasp the nuances of their incorrect generations? *CoRR*, abs/2402.19475.

Çaglar Gülçehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. 2023. Reinforced self-training (rest) for language modeling. *CoRR*, abs/2308.08998.

Haixia Han, Jiaqing Liang, Jie Shi, Qianyu He, and Yanghua Xiao. 2024. Small language model can self-correct. *CoRR*, abs/2401.07301.

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 *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron C. Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. *CoRR*, abs/2402.06457.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023a. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 1051–1068. Association for Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023b. Large language models cannot self-correct reasoning yet. *CoRR*, abs/2310.01798.

Alon Jacovi, Yonatan Bitton, Bernd Bohnet, Jonathan Herzig, Or Honovich, Michael Tseng, Michael Collins, Roee Aharoni, and Mor Geva. 2024. A chain-of-thought is as strong as its weakest link: A benchmark for verifiers of reasoning chains. *CoRR*, abs/2402.00559.

Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. 2023. GRACE: Discriminator-guided chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15299–15328, Singapore. Association for Computational Linguistics.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. 2020. QASC: A dataset for question answering via sentence composition. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*

2020, New York, NY, USA, February 7-12, 2020, pages 8082–8090. AAAI Press.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

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

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023b. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 5315–5333. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *CoRR*, abs/2305.20050.

Bill Yuchen Lin, Ziyi Wu, Yichi Yang, Dong-Ho Lee, and Xiang Ren. 2021. Riddlesense: Reasoning about riddle questions featuring linguistic creativity and commonsense knowledge. In *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 1504–1515. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *CoRR*, abs/2303.17651.

Yujun Mao, Yoon Kim, and Yilun Zhou. 2024. CHAMP: A competition-level dataset for fine-grained analyses of llms' mathematical reasoning capabilities. *CoRR*, abs/2401.06961.

Minh-Vuong Nguyen, Linhao Luo, Fatemeh Shiri, Dinh Phung, Yuan-Fang Li, Thuy-Trang Vu, and Gholamreza Haffari. 2024. Direct evaluation of chain-of-thought in multi-hop reasoning with knowledge graphs. *CoRR*, abs/2402.11199.

Juhyun Oh, Eunsu Kim, Inha Cha, and Alice Oh. 2024. The generative AI paradox on evaluation: What it can solve, it may not evaluate. *CoRR*, abs/2402.06204.

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

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *CoRR*, abs/2308.03188.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2023. REFINER: reasoning feedback on intermediate representations. *CoRR*, abs/2304.01904.

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

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *CoRR*, abs/1811.00937.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

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

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022a. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022b. Solving math word problems with process- and outcome-based feedback. *CoRR*, abs/2211.14275.

Peiyi Wang, Lei Li, Liang Chen, Feifan Song, Binghuai Lin, Yunbo Cao, Tianyu Liu, and Zhifang Sui. 2023. Making large language models better reasoners with alignment. *CoRR*, abs/2309.02144.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 2550–2575. Association for Computational Linguistics.

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. 2023. The generative AI paradox: "what it can create, it may not understand". *CoRR*, abs/2311.00059.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A. Smith, Mari Ostendorf, and Hannaneh Hajishirzi. 2023. Fine-grained human feedback gives better rewards for language model training. *CoRR*, abs/2306.01693.

Xin Xu, Shizhe Diao, Can Yang, and Yang Wang. 2024. Can we verify step by step for incorrect answer detection? *CoRR*, abs/2402.10528.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. 2024. Do large language models latently perform multi-hop reasoning? *CoRR*, abs/2402.16837.

Seonghyeon Ye, Yongrae Jo, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, and Minjoon Seo. 2023. Selfee: Iterative self-revising llm empowered by self-feedback generation. Blog post.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023a. Outcome-supervised verifiers for planning in mathematical reasoning. *CoRR*, abs/2311.09724.

Xiao Yu, Baolin Peng, Michel Galley, Jianfeng Gao, and Zhou Yu. 2023b. Teaching language models to self-improve through interactive demonstrations. *CoRR*, abs/2310.13522.

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

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *CoRR*, abs/2308.01825.

Mert Yüksekgönül, Varun Chandrasekaran, Erik Jones, Suriya Gunasekar, Ranjita Naik, Hamid Palangi, Ece Kamar, and Besmira Nushi. 2023. Attention satisfies: A constraint-satisfaction lens on factual errors of language models. *CoRR*, abs/2309.15098.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Xinliang Frederick Zhang, Carter Blum, Temma Choji, Shalin Shah, and Alakananda Vempala. 2024. Ultra: Unleash llms' potential for event argument extraction through hierarchical modeling and pair-wise refinement. *ArXiv*, abs/2401.13218.

Chujie Zheng, Fan Yin, Hao Zhou, Fandong Meng, Jie Zhou, Kai-Wei Chang, Minlie Huang, and Nanyun Peng. 2024a. Prompt-driven llm safeguarding via directed representation optimization. *arXiv preprint arXiv:2401.18018*.

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

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

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. 2023. Representation engineering: A top-down approach to AI transparency. *CoRR*, abs/2310.01405.

A Prompts

8-Shot CoT Prompts with 2 Shots Shown Below

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?
Explain your reasoning step by step. Your final answer should be a single numerical number, in the form $\boxed{answer}$. End your response with [END].

Step 1: Jason originally had 20 lollipops.
Step 2: After giving some to Denny, Jason has 12 lollipops.
Step 3: The number of lollipops Jason gave to Denny is the difference between the original number and the remaining number, which is 20 - 12 = 8 lollipops.
Step 4: The answer is $\boxed{8}$ lollipops. [END]

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?
Explain your reasoning step by step. Your final answer should be a single numerical number, in the form $\boxed{answer}$. End your response with [END].

Step 1: Initially, there are 15 trees in the grove.
Step 2: After planting, the total number of trees in the grove is 21.
Step 3: The number of trees planted by the grove workers today is the difference between the final number and the initial number of trees, which is 21 - 15 = 6 trees.
Step 4: The answer is $\boxed{6}$ trees. [END]

2-Shot Critique Prompts with 1 Shot Shown Below

Q: Betty is saving money for a new wallet which costs $100. Betty has only half of the money she needs. Her parents decided to give her $15 for that purpose, and her grandparents twice as much as her parents. How much more money does Betty need to buy the wallet?

Answer 1 (Incorrect):

Step 1: Betty needs $100 to buy the wallet.
Step 2: Her parents gave her $15, which is half of the remaining amount.
...
Step 7: The answer is $\boxed{\$55}$.

Answer 2 (Correct):

Step 1: Betty needs $100 to buy the wallet, but she currently has only half of the money, which is $50.
Step 2: Her parents gave her $15, so she now has $50 + $15 = $65.
...
Step 6: The answer is $\boxed{5}$ dollars.

There are reasoning errors in Answer 1. Please go through each step in Answer 1, use Answer 2 as a reference for the correct approach, and provide feedback that helps correct the errors in Answer 1. End your response with [END].

Let's go through the errors in Answer 1 and provide feedback:

Answer 1 (Incorrect):

Step 1: Betty needs $100 to buy the wallet.
Feedback: This step is correct.

Step 2: Her parents gave her $15, which is half of the remaining amount.
Feedback: This is incorrect. Betty's parents gave her $15, but it's not necessarily half of the remaining amount. The question states that Betty has only half of the money she needs, not half of the remaining amount after her parents' contribution.

...

Step 7: The answer is $\boxed{\$55}$.

Feedback: The correct answer, based on the corrected calculations, should be $\boxed{\$5}$. [END]

2-Shot Correction Prompts with 1 Shot Shwon Below

Q: Kelly is grocery shopping at a supermarket and is making sure she has enough in her budget for the items in her cart. Her 5 packs of bacon cost $10 in total and she has 6 packets of chicken which each cost twice as much as a pack of bacon. She also has 3 packs of strawberries, priced at $4 each, and 7 packs of apples, each priced at half the price of a pack of strawberries. If Kelly's budget is $65 then how much money, in dollars, does she have left in her budget?

Step 1: Kelly spends $10 in total for 5 packs of bacon.
Feedback: This step is correct.

Step 2: Each packet of chicken costs twice as much as a pack of bacon, which is 2 x $10 = $20.
Feedback: Each pack of chicken costs twice as much as a pack of bacon, not the total cost of bacon. So we need to calculate the cost of one pack of bacon first (total cost of bacon / total bags of bacon) and use that.

...

Step 7: The answer is $\boxed{\$29}$.

Can you correct the errors in your reasoning based on the feedback provided? Your final answer should be a single numerical number, in the form $\boxed{answer}$. End your response with [END].

Step 1: Kelly spends $10 in total for 5 packs of bacon, so each pack of bacon costs $10 / 5 = $2.
Step 2: Each packet of chicken costs twice as much as a pack of bacon, which is 2 x $2 = $4. Since she has 6 packets of chicken, she spends 6 x $4 = $24 on chicken.
...
Step 7: The answer is $\boxed{\$5}$. [END]

4-Shot Verifying Prompts with 2 Shots Shown Below

Question: Betty is saving money for a new wallet which costs $100. Betty has only half of the money she needs. Her parents decided to give her $15 for that purpose, and her grandparents twice as much as her parents. How much more money does Betty need to buy the wallet?

Solution:
Step 1: Betty needs $100 to buy the wallet.
Step 2: Her parents gave her $15, which is half of the remaining amount.
Step 3: Her grandparents gave her twice as much as her parents, which is 2 x $15 = $30.
Step 4: The total amount of money Betty has now is $15 + $30 = $45.
Step 5: The difference between the amount she needs and the amount she has is $100 - $45 = $55.
Step 6: Betty still needs $55 to buy the wallet.
Step 7: The answer is $\boxed{\$55}$.

Is this solution correct or not? Answer **correct** or **incorrect** and explain your reasoning. End your response with [END].

The solution is **incorrect**. You've ignored the original amount that Betty had saved, which is half the wallet's cost ($100/2 = $50). So, before receiving money from her parents and grandparents, she already had $50. [END]

Question: Betty is saving money for a new wallet which costs $100. Betty has only half of the money she needs. Her parents decided to give her $15 for that purpose, and her grandparents twice as much as her parents. How much more money does Betty need to buy the wallet?

Solution:
Step 1: Betty needs $100 to buy the wallet, but she currently has only half of the money, which is $50.
Step 2: Her parents gave her $15, so she now has $50 + $15 = $65.
Step 3: Her grandparents gave her twice as much as her parents, so she received $15 x 2 = $30 from her grandparents.
Step 4: Betty now has a total of $65 + $30 = $95.
Step 5: To find out how much more money Betty needs, subtract the amount she has from the amount she needs, which is $100 - $95 = $5.
Step 6: The answer is $\boxed{5}$ dollars.

Is this solution correct or not? Answer **correct** or **incorrect** and explain your reasoning. End your response with [END].

This solution is **correct**. Betty only needs $5 more to buy the $100 wallet. This is because she started with $50, received $15 from her parents, and received $30 from her grandparents, which totals $95. Subtracting that from the total cost of the wallet leaves her with needing just $5. [END]

B Proof: Using Correct Solutions as Hints Make Critique Generation Easier

Intuitively, if we provide the LM with a correct solution, it will be easier for the LM to generate a critique for the incorrect solution of the same question. In fact, we can verify this intuition from a mathematical perspective. Given a pair of incorrect-correct solutions (s^-, s^+), our goal is to learn a mapping from the incorrect solution to the correct one, which is modeled by $M(s^+|q, s^-)$, where $M(\cdot)$ is the probability distribution of the base LM. By introducing critique c as intermediate generations, we have:

$$M(s^+|q, s^-) = M(c|q, s^-) \cdot M(s^+|q, s^-, c) \quad (1)$$

The optimal critique c^* that we want to find should be best at facilitating this two-phase generation process, i.e.,

$$c^* = \underset{c}{\operatorname{argmax}}\, M(c|q, s^-) \cdot M(s^+|q, s^-, c) \quad (2)$$

To obtain the optimal critique, we can first ask the model to reverse-engineer a critique $\hat{c}$ with the correct solution s^+ as a hint:

$$\hat{c} = \underset{c}{\operatorname{argmax}}\, M(c|q, s^-, s^+) \quad (3)$$

With Bayes' theorem, Equation 3 can be re-written as:

$$\begin{aligned} \hat{c} &= \underset{c}{\operatorname{argmax}}\, M(c|q, s^-, s^+) \\ &= \underset{c}{\operatorname{argmax}}\, \frac{M(c|q, s^-) \cdot M(s^+|q, s^-, c)}{M(s^+|q, s^-)} \\ &= \underset{c}{\operatorname{argmax}}\, M(c|q, s^-) \cdot M(s^+|q, s^-, c) \\ &= c^* \end{aligned} \quad (4)$$

which is exactly what we want.

This simple proof shows that in principle, the critique generated with the correct solution as a hint should be best at guiding the LM to recover the correct solution from the incorrect one.

C Combining SCORE with Rejection Sampling Fine-Tuning

Aside from self-correction upon the initial solution generated by the base LM as shown in Figure 2, we also explore whether our proposed self-correction method can be combined with other fine-tuning methods (e.g., rejection sampling fine-tuning) to further improve reasoning performance. Here we replace the base LM with the RFT model. Rejection sampling fine-tuning leverages correct generations for training. Yet it ignores rich information in the large amounts of incorrect solutions. We hope weaker LMs can also learn from its own mistakes to become better reasoners. Since we have already obtained a stronger RFT model that avoids some mistakes by base LMs that are easier to fix, we want our refiner to learn to correct errors that require more in-depth thinking. Consequently, to collect the incorrect solutions to be reflected upon, we *resample* 10 solutions for each question from the RFT model, instead of reusing the sampled solutions from the base LM in step (a) of Figure 1. Then we follow the rest of the pipeline to collect critique-correction data for refiner finetuning.

Table 4 shows that our method is complementary to RFT. Concretely, RFT improves the final accuracy upon the few-shot prompting baseline from 37.2 to 42.7, while our self-correction system (gpt-4 as verifier and our finetuned LLaMA as refiner) can further improve the performance from 42.7 to 45.1, demonstrating the effectiveness of our method over the few-shot prompting and RFT baselines.

Table 5 investigates the optimal approach for ini-

Verifier	Refiner	GSM8K Verifier F1	GSM8K Final Accu.
Initial solutions by RFT model		N/A	42.7
ours	SCORE	47.3	42.8 (+0.1)
`gpt-3.5-turbo`	SCORE	59.7	31.5 (-11.2)
w/ SC@10	SCORE	63.6	38.1 (-4.6)
`gpt-4`	SCORE	**89.2**	**46.3** (+3.6)
oracle	SCORE	100.0	51.4 (+8.7)

Table 4: Performance of self-correction with different inputs on GSM8K test set using `LLaMA-2-13B-chat` as base LM. The initial solution is generated by the rejection-sampling fine-tuned model.

Verifier param. init. from	Refiner param. init. from	Refiner ft. on solutions from	Final Accu.	Oracle Accu.
Initial solutions by RFT model			44.9	44.9
RFT	base	RFT	**45.3**	**53.3**
base	base	RFT	45.0	**53.3**
RFT	RFT	RFT	45.2	52.3
base	RFT	RFT	45.0	52.3
RFT	base	base	45.1	**53.3**

Table 5: Ablation study on parameter initialization and refiner's fine-tuning data source. Results are shown on GSM8K dev set with `LLaMA-2-13B-chat` as base LM.

Verifier	Refiner	GSM8K		GSM8K → MATH Subset		CSQA		CSQA → QASC		CSQA → RiddleSense	
		V. F1	ACC	V. F1	ACC	V. F1	ACC	V. F1	ACC	V. F1	ACC
Base LM: `LLaMA-2-13B-chat`											
Initial answers by few-shot prompting		-	37.2	-	23.8	-	69.7	-	65.9	-	57.6
prompted	prompted	31.9	36.9 $_{-0.3}$	20.0	23.8 $_{+0.0}$	52.5	62.2 $_{-7.5}$	53.1	63.3 $_{-2.6}$	51.9	55.2 $_{-2.4}$
oracle	prompted	100.0	39.7 $_{+2.5}$	100.0	26.5 $_{+2.7}$	100.0	83.7 $_{+14.0}$	100.0	76.1 $_{+10.2}$	100.0	72.7 $_{+15.1}$
self	SCORE (fine-tuned)	51.1	37.5 $_{+0.3}$	36.0	23.8 $_{+0.0}$	47.1	69.7 $_{+0.0}$	45.8	65.6 $_{-0.3}$	42.1	57.5 $_{-0.1}$
`gpt-4`	SCORE (fine-tuned)	88.4	41.4 $_{+4.2}$	82.9	26.5 $_{+2.7}$	80.3	72.4 $_{+2.7}$	85.9	68.1 $_{+2.2}$	87.7	**67.3** $_{+9.7}$
oracle	SCORE (fine-tuned)	100.0	46.0 $_{+8.8}$	100.0	31.5 $_{+7.7}$	100.0	86.2 $_{+16.5}$	100.0	78.0 $_{+12.1}$	100.0	76.4 $_{+18.8}$
Base LM: `Gemma-7B-it`											
Initial answers by few-shot prompting		-	36.3	-	27.1	-	67.2	-	65.0	-	57.1
prompted	prompted	45.3	36.3 $_{+0.0}$	39.2	28.2 $_{+1.1}$	55.7	65.4 $_{-1.8}$	61.0	65.1 $_{+0.1}$	58.2	55.3 $_{-1.8}$
oracle	prompted	100.0	37.7 $_{+1.4}$	100.0	29.3 $_{+2.2}$	100.0	74.5 $_{+7.3}$	100.0	71.3 $_{+6.3}$	100.0	62.3 $_{+5.2}$
self	SCORE (fine-tuned)	56.8	36.7 $_{+0.4}$	49.5	27.1 $_{+0.0}$	42.0	67.5 $_{+0.3}$	41.0	65.2 $_{+0.2}$	38.1	56.8 $_{-0.3}$
`gpt-4`	SCORE (fine-tuned)	89.5	**42.5** $_{+6.2}$	82.8	**39.2** $_{+12.1}$	82.7	**75.0** $_{+7.8}$	89.9	**72.8** $_{+7.8}$	85.6	64.9 $_{+7.8}$
oracle	SCORE (fine-tuned)	100.0	47.4 $_{+11.1}$	100.0	44.2 $_{+17.1}$	100.0	85.4 $_{+18.2}$	100.0	77.3 $_{+12.3}$	100.0	72.7 $_{+15.6}$

Table 6: Performance of SCORE models using `LLaMA-2-13B-chat` and `Gemma-7B-it` as base LM. We report F1 score of the verifiers (V. F1) and final answer accuracy (ACC). We include test results for training tasks (GSM8K and CommonsenseQA/CSQA), as well as transfer evaluation of GSM8K trained models on MATH subset, CSQA trained models on QASC and RiddleSense. All models use greedy decoding. We highlight the best-performing system per model without using an oracle verifier. On each dataset, the superior model among the highlighted ones is indicated in **bold**.

tializing the parameters of the refiner and verifier during fine-tuning. Our empirical findings indicate that initializing the verifier from the RFT model and the refiner from the base language model results in superior performance. We hypothesize that stronger reasoning capabilities of RFT model could complement the verification skills. Furthermore, when generating critiques of solutions, those created using solutions by the RFT model outperform those based on the base model.

D Combining SCORE with Oversample-then-Rerank

Self-correction could be combined with oversample-then-rerank, an orthogonal approach to further improve reasoning (Cobbe et al., 2021; Li et al., 2023b; Hosseini et al., 2024). The oversample-then-rerank approach first samples k solutions per question by few-shot prompting, then leverages a trained verifier to score the correctness of each solution, and selects the final answer based on a weighted voting scheme (Li et al., 2023b) as follows:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_{i=1}^{k} \mathbb{1}_{\mathbf{y}_i=\mathbf{y}} \cdot p_v(q, s_i, \mathbf{y}_i),$$

where q is the question, s_i is the i-th sampled solutions, $\mathbf{y}_i$ is the final answer extracted from s_i, and $\mathbb{1}_{\mathbf{y}_i=\mathbf{y}}$ is an indicator function that returns 1 (or 0) if $\mathbf{y}_i = \mathbf{y}$ (or not), and $p_v(\cdot)$ is the probability produced by the verifier[6]

To integrate self-correction with the oversample-then-rerank approach, we apply the SCORE refiner to solutions predicted to be correct with a probability *below* a certain threshold. This threshold is automatically determined to maximize accuracy on the development set and is then fixed for use during test-time predictions. The refiner greedily decodes one refined solution for each solution. Subsequently, we combine the refined solutions with the original solutions predicted as correct to arrive at the final answer using weighted voting for aggregation:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_{i=1}^{k} \big(\mathbb{1}_{\mathbf{y}_i=\mathbf{y}} \cdot p_v^i \cdot \mathbb{1}_{p_v^i \geq C} + \mathbb{1}_{\mathbf{y'}_i=\mathbf{y}} \cdot p_v^{i'} \cdot \mathbb{1}_{p_v^i < C}\big),$$

[6]We find that for answer aggregation, weighted voting performs better than majority voting and selecting the top-1 solution, which echoes the findings of Li et al. (2023b). Therefore, we report the weighted voting results in this section.

# Candidate Solutions	Refine w/ SCORE?	GSM8K	GSM8K→ MATH Subset
Base LM: `LLaMA-2-13B-chat`			
1	✗	37.2	23.8
1	✓	37.5 +0.3	23.8 +0.0
10	✗	44.3 +7.1	**30.9 +7.1**
10	✓	**46.0 +8.8**	29.8 +6.0
Base LM: `Gemma-7B-it`			
1	✗	36.3	27.1
1	✓	36.7 +0.4	27.1 +0.0
10	✗	39.5 +3.2	34.3 +7.2
10	✓	**41.1 +4.8**	**35.9 +8.8**

Table 7: Self-correction combined with the oversample-then-rerank strategy (Cobbe et al., 2021) yield better results than using either one alone. We show results on GSM8K test set and transfer evaluation of GSM8K trained models on MATH subset using `LLaMA-2-13B-chat` and `Gemma-7B-it` as base LMs.

where $p_v^i = p_v(q, s_i, \mathbf{y}_i)$ is the verifier probability for the original solution s_i, $p_v^{i'} = p_v(q, s'_i, \mathbf{y}'_i)$ is the verifier probability for the refined solution s'_i coming from s_i, and C is a threshold for determining when to self-refine.

We sample $k = 10$ solutions for each question with a temperature of 0.9 and use the same fine-tuned self-verifier (not `gpt-4` or oracle verifier) across all the settings in Table 7 that involve a verifier. As shown by previous work, oversample-then-rerank improves performance upon few-shot prompting with only greedy decoding by a large margin, because it better explores the solution space by sampling multiple solutions. Our self-correction method is both *orthogonal* to and *synergistic* with the oversample-then-rerank. By incorporating our self-correction method into the baseline, with only a weak self-verifier, we still observe an improvement in accuracy compared to the baseline: +1.7 for `LLaMA-2-13B-chat` on GSM8K, and +1.6 for GSM8K-trained `Gemma-7B-it` on MATH. By aggregating verifications from multiple solutions, the weak verifiers become stronger via ensembling and thus unleash the power of refiner for better self-correction of reasoning.

E Inference Overhead by SCORE

Although we employ three models for inference (few-shot prompted solution generator, self-verifier, self-refiner), it will not necessarily triple the latency. The self-verifier's latency is minimal because it classifies rather than generates, and the refiner is invoked only if the verifier flags the initial solution as incorrect. The increased inference time of our pipeline over the base model is modest, at x1.3 times for `LLaMA-2-13B-chat` and x1.4 times for `Gemma-7B-it`. Comparatively, the oversample-then-rerank baseline (Appendix D) results in a 4.5 times increase in latency when sampling 10 solutions per question.