Mind the Gap: Examining the Self-Improvement Capabilities of Large Language Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Self-improvement is a mechanism in Large Language Model (LLM) pre-training, post-training and test-time inference. We explore a framework where the model verifies its own outputs, filters or reweights data based on this verification, and distills the filtered data. Despite several empirical successes, a fundamental understanding is still lacking. In this work, we initiate a comprehensive, modular and controlled study on LLM self-improvement. We provide a mathematical formulation for self-improvement, which is largely governed by a quantity which we formalize as the *generation-verification gap*. Through experiments with various model families and tasks, we discover a scaling phenomenon of self-improvement – a variant of the generation-verification gap scales monotonically with the model pre-training flops. We also examine when self-improvement is possible, an iterative self-improvement procedure, and ways to improve its performance. We believe our results have several empirical implications, and our study leaves many exciting future directions for understanding the potential and limits of LLM self-improvement.

1 Introduction

Recent work increasingly explores the use of synthetic data in training large language models (LLMs), with applications in both pre-training and post-training (Bai et al., 2022; Meng et al., 2022; Li et al., 2023b; Adler et al., 2024; Dubey et al., 2024; Yang et al., 2024; Hui et al., 2024; Li et al., 2024). While synthetic data, often generated by LLMs, offers a valuable complement to human-generated data, its misuse can harm performance. Bertrand et al. (2023) and Gerstgrasser et al. (2024) showed self-training on model-generated data leads to degradation. To mitigate this, incorporating a "reliable" verifier to label data has shown promise in preventing such performance collapse (Gillman et al., 2024).

A straightforward verification mechanism is to train a reward model on human-annotated data to assess the quality of synthetic data (Lightman et al., 2023; Wang et al., 2024a). However, this approach can be prohibitively expensive and may offer no signal in domains where models exhibit super-human performance. An alternative is to use a stronger model (Chang et al., 2023; Havrilla et al., 2024) for annotation, but this becomes infeasible when the model is at the frontier of current capabilities. A promising solution is to use the model to label its own generations. Motivated by the intuition that "verification is easier than generation", one can hypothesize that the model may act as a better-than-random verifier of its own outputs, enabling *self-improvement* (Zelikman et al., 2022).

Most previous self-improvement algorithms can be summarized as follows: 1) make multiple generations from the model, 2) use the same model to verify the generations, and 3) distill from the reranked/filtered generation (Zelikman et al., 2022; Huang et al., 2022; Wang et al., 2022b; Yehudai et al., 2024; Madaan et al., 2024; Yuan et al., 2024; Xu et al., 2024; Liang et al., 2024). With this framework, self-improvement is also related to improving inference quality (Wang et al., 2022a;

Submitted to the Mathematics of Modern Machine Learning Workshop at NeurIPS 2024. Do not distribute.

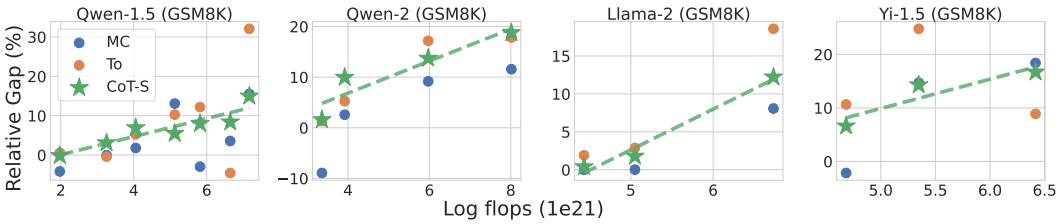


Figure 1: With proper verification method (e.g., CoT-S), *the relative generation-verification gap (Definition 2.2) scales monotonically with respect to the pre-train flops.* We conjecture that in this case, the relative gap is linear with respect to the log of the pre-train flops. MC denotes Multiple Choice verification, CoT-S denotes CoT-Score verification, and To denotes Tournament verification. The description of each verification can be found in Section 3.

Welleck et al., 2024) – if the model can verify its own generation, self-improvement can enhance test-time performance with additional computation towards more generations and updates.

Despite significant empirical progress and some impressive results, a fundamental understanding of LLM self-improvement remains limited. It is uncertain whether these results can be interpreted solely as an indication of the self-improvement capability of LLMs, given the potential for confounders at various stages of the process. Moreover, much of the existing research has concentrated on just one model family or a single verification mechanism, limiting the broader applicability of the findings. In this work we conduct a comprehensive study of the self-improvement capability of LLMs; our contribution is as follows:

- **Self-Improvement Framework:** Section 2 details the mathematical formulation of the self-improvement process, highlighting three critical desiderata. We propose the *generation-verification gap (GV-Gap)* (Definition 2.1) as the central metric for evaluation. GV-Gap captures the additional "precision" gained by using model verification over generations alone. It is defined as the performance improvement obtained by re-weighting generations by the model's self-verification score (e.g., 0 or 1). Our empirical findings indicate that GV-Gap is a more accurate metric for measuring self-improvement versus the previous metric of the performance difference after a model update.

- **Scaling Properties:** In Section 4, we measure the generation-verification gap across multiple model families, verification mechanisms, and tasks. Certain verification methods induce a scaling phenomenon for self-improvement – relative GV-Gap (Definition 2.2) increases monotonically with pre-train flops – shown in Figure 1. We find that in cross-verification, the GV-gap increases with verifier capability and decreases with generator capability. Finally, we observe that most models do not achieve self-improvement in information retrieval and reasoning tasks that exceed their inherent capabilities. By studying multiple types of verification, our results indicate general patterns beyond just prompt engineering.

- **Iterative Self-Improvement:** In Section 5, we identify that **i)** GV-Gap saturates to 0 in handful rounds of iterative self-improvement; **ii)** the saturation rate is independent from the model capacity, **iii)** the effective diversity degrades during the iterative self-improvement.

- **Verification Mechanisms:** In Appendix A, we consider methods to enhance self-improvement through a fine-grained study on the verification methods. Key observations include: **i)** there is significant non-overlap between different verification mechanisms, **ii)** GV-Gap is not positively correlated with generation accuracy, **iii)** using an ensemble of verification can improve self-improvement.

We believe our results provide an initial step towards a systematic understanding of the intriguing self-improvement framework in LLMs. While our observations provide several practical implications in pre-training, post-training, and test-time inferencing, we also leave several interesting future directions toward a more profound understanding of the mechanism of self-improvement.

Due to space constrain, we defer the related work section to Appendix B.

2 A Dissection of the Self-improvement Framework

In this section, we introduce the setup of the self-improvement framework considered in the paper, which consists of the following three main components:

Response Generation. Let $\mathcal{X}$ be the prompt/context space, and $\mathcal{Y}$ be the response space. Let $\mathcal{F} \subseteq \{f : \mathcal{X} \to \Delta(\mathcal{Y})\}$ be a class of generative models that maps a prompt to a distribution over responses. A task, $\mathsf{task} \in \mathcal{T}$ (e.g., math, code, trivia, puzzles, etc.), defines a distribution $\mu_{\mathsf{task}} \in \Delta(\mathcal{X})$ over the prompt space $\mathcal{X}$, and utility function $u_{\mathsf{task}} : \mathcal{X} \times \mathcal{Y} \to [U_{\mathsf{min}}, U_{\mathsf{max}}]$, where $U_{\mathsf{min}}, U_{\mathsf{max}}$ denote bounds of the utility function.

The goal is to find a generative model that maximizes the expected utility: $f^\star_{\mathsf{task}} := \arg\max_{f \in \mathcal{F}} J_{\mathsf{task}}(f)$, where $J_{\mathsf{task}}(f) := \mathbb{E}_{x \sim \mu_{\mathsf{task}}}\left[\mathbb{E}_{y \sim f(\cdot|x)}[u_{\mathsf{task}}(x, y)]\right]$. We will often drop the subscript task when it is clear from the context.

Verify with Proxy Utility. Without the access to the ground truth utility function u, we rely on a proxy utility function constructed by some model f, denoted as $\widehat{u}_f : \mathcal{X} \times \mathcal{Y} \to [U_{\mathsf{min}}, U_{\mathsf{max}}]$. For example, let $Z = [10] := \{1, 2, \ldots 10\}$ be scores, the proxy utility $\widehat{u}_f(x, y) = \mathbb{E}_{z \sim f(\cdot|x,y,\mathsf{prom})}[z]$ rates the quality of the response with a score from 1 to 10 with an instruction prompt prom. In Section 3, we provide more proxy utility functions used in our experiments.

Update via Reweighting. Let $t \in [T]$ be the iteration index, f_t be the model at iteration t, and $\widehat{u}_{f'}$ be the proxy utility defined by model f'. The reweighted distribution $f_t[w(\widehat{u}_{f'})]$ is defined such that

$$f_t[w(\widehat{u}_{f'})](y \mid x) \propto f_t(y \mid x) \cdot w(\widehat{u}_{f'}(x, y)), \forall x, y \in \mathcal{X} \times \mathcal{Y} .$$

Here $w : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a weight function that maps a utility from the verification procedure to a weight. The specific form of w is determined by the algorithm used for the model update step (we provide two examples below). The objective of this update is to find a model $f_{t+1} \in \mathcal{F}$ such that some distance $\ell(f_{t+1}, f_t[w(\widehat{u}_{f'})])$ is small (i.e., ℓ-projecting $f_t[w(\widehat{u}_{f'})]$ onto $\mathcal{F}$). Note that when $f' = f_t$, we have self-improvement, though the framework can also be used for improving using utilities produced by a different model as is studied in Section 4.2.

Example 2.1 (KL-regularized RL Update)**.** *One can treat the proxy utility $\widehat{u}$ as reward, and perform RLHF style RL update with a reverse KL constraint (Christiano et al., 2017; Ouyang et al., 2022):*

$$f_{t+1} = \arg\max_{f \in \mathcal{F}} \mathbb{E}_{x,y \sim \mu \circ f}\left[\widehat{u}_{f_t}(x, y) - \beta \log\left(\frac{f(y \mid x)}{f_t(y \mid x)}\right)\right].$$

In this case, we have $w(s) = \exp(s/\beta)$, and ℓ can be the KL divergence between f_{t+1} and $f_t[w(\widehat{u}_{f_t})]$ (Hazan, 2016). ◁

Example 2.2 (Rejection Sampling)**.** *In rejection sampling, we first filter the generation by a threshold τ, and then fine-tune the model on the filtered data:*

$$f_{t+1} = \arg\max_{f \in \mathcal{F}} \mathbb{E}_{x,y \sim \mu \circ f_t}[\log(f(y \mid x)) \cdot \mathbb{1}[\widehat{u}_{f_t}(x, y) \geq \tau]].$$

In this case, we have $w(s) = \mathbb{1}[s \geq \tau]$, and ℓ can be the total variation distance between f_{t+1} and $f_t[w(\widehat{u}_{f_t})]$ (Zhang, 2006). ◁

Finally, it is convenient to abuse the notation and allow w and $\widehat{u}$ to take batch input. For example, we can allow w to take a list of score and then set the filtering threshold τ to the n quantile ($n \in [0, 1]$) of the score. We denote this as top-n or quantile-n filtering.

2.1 Three Key Factors of Self-improvement

For any meaningful self-improvement, at iteration t, we would like to find f_{t+1} such that $J(f_{t+1}) > J(f_t)$, where recall $J(f)$ is the expected utility under the model f. We identify the three key conditions that may bottleneck improvement on model f:

1. Improvable Generation. Our framework involves reshaping the generation distribution towards increased utility. In order for this to be useful, the utilities of generations must have variability. For example, if generation were done with greedy decoding, no improvement in this process would be possible. Fortunately, the improvable generation phenomenon has been well-observed in LLMs (Li et al., 2022; Brown et al., 2024) (see also Figure 3).

2. Informative verification. Recall that weight function $w(\widehat{u}_g)$ is defined by the proxy utility function $\widehat{u}_g$, which is constructed by a verifier model g. If the verification capability is limited, the weighting may not provide a useful signal for improvement. The following definition quantifies this intuition:

Definition 2.1 (Generation-Verification Gap)**.** *For a generator f and verifier g, we define the generation-verification gap (GV-Gap) between f and g as*

$$\mathsf{gap}(f, g) := J(f[w(\widehat{u}_g)]) - J(f) \,.$$

This is the core metric for our analysis throughout the paper. When the generator f and verifier g are the same, we denote the shorthand $\mathsf{gap}(f) := \mathsf{gap}(f, f)$, which is the self-improvement generation-verification gap. However, the GV-Gap is an absolute quantity, which does not fully capture the various qualities of the generation. Imagine a generator that already achieves 99% accuracy on a task: first, the upper bound for $\mathsf{gap}(f)$ is only 0.01; second, incorrect responses are likely to be very subtle, and thus any improvement in the reweighted distribution might require a very strong verification model. This motivates the relative GV-Gap:

Definition 2.2 (Relative Generation Verification Gap)**.** *For a generator f and verifier g, we define the relative generation-verification gap between f and g as*

$$\mathsf{gap}_{\mathsf{rel}}(f, g) := \mathbb{E}_{x\sim\mu}\left[\frac{\mathbb{E}_{y\sim f[w(\widehat{u}_g)](\cdot|x)}[u(x,y)] - \mathbb{E}_{y\sim f(\cdot|x)}[u(x,y)]}{U_{\mathsf{max}} - \mathbb{E}_{y\sim f(\cdot|x)}[u(x,y)]}\right],$$

That is, we weigh the gap of each prompt by its deficiency to the best possible utility. For simplicity, we will denote the self-GV-Gap as "gap" or gap and relative self-GV-Gap as "relative gap" or $\mathsf{gap}_{\mathsf{rel}}$ when the context is clear. In domains where verification is easier than generation, $\mathsf{gap} > 0$ likely holds, and indicates that there is additional signal that can be exploited. One can also check that, for all prompts $x \in \mathcal{X}$, if the weight function $w(\widehat{u}_g)(x, \cdot)$ and $u(x, \cdot)$ is positively correlated[1] under the distribution of $y \sim f$, then we can always guarantee $\mathsf{gap}(f, g) > 0$.

3. High-fidelity model update. The final condition is that the model f_{t+1} mimics/distills the performance of the reweighted distribution $f_t[w(\widehat{u}_{f_t})]$, i.e., $|J(f_{t+1}) - J(f_t[w(\widehat{u}_{f_t})])| \le \varepsilon_{\mathsf{update}}$, with some small ε. For example, if through MLE we bound the TV-distance between the two by ε', then by Holder's inequality we have $\varepsilon_{\mathsf{update}} \le \varepsilon' U_{\mathsf{max}}$. Combining it with the gap guarantee, we have:

$$J(f_{t+1}) - J(f_t) \ge \mathsf{gap}(f_t, g) - \varepsilon_{\mathsf{update}}.$$

With a sufficiently expressive LLM model class, it is often observed that the distillation error ε is small.

Note that sometimes we might observe $J(f_{t+1}) - J(f_t[w(\widehat{u}_{f_t})]) > 0$. For instance, a benchmark may require outputs in a specific format; in such cases, the finetuned model f_{t+1} might outperform the reweighted distribution $f_t[w(\widehat{u}_{f_t})]$ simply by aligning outputs with the required format, even if the underlying answers remain unchanged. Recent work (Dubois et al., 2024; Zhang et al., 2024c) highlight additional confounders, such as modifications to output length during finetuning, which may inflate perceived improvements without reflecting true model capabilities. Conversely, it is conceivable that intrinsic enhancements in the model's reasoning might occur; for example, mastering simpler tasks could indirectly boost performance on more complex problems requiring similar skills. However, in our experiment we only observe the former scenario. That said, both cases emphasize the need for caution when interpreting such improvements, and this further emphasizes the importance of our modular approach in dissecting the components of self-improvement.

3 Experiment Setup

Our experiment is based on lm-evaluation-harness (Gao et al., 2024). For all tasks we use the following setup: for generations and verification, we use sampling parameters $p = 0.9, t = 0.7$, max length of 512 and 4-shot in-context samples. For each prompt, we sample 128 responses and sample 1 verification for each response[2]. In this work, we consider the following verification mechanisms (a formal description of each is provided in Appendix C along with example prompts in Appendix F):

[1] Even under the case that $w(\widehat{u}_g)(x, \cdot)$ and $u(x, \cdot)$ are negatively correlated, if we have a small set of holdout dataset with ground truth labels $u(x, y)$, we can always define $w(\alpha) := \exp(\alpha \cdot w((\widehat{u}_g)))$, use the holdout set to tune α, and use $w(\alpha)$ to reweight the distribution.

[2] Note that an ideal verification should be sampling multiple verifications per generation. We only sample one due to computational constraints and we leave multiple verifications along with understanding verification compute scaling for future work.

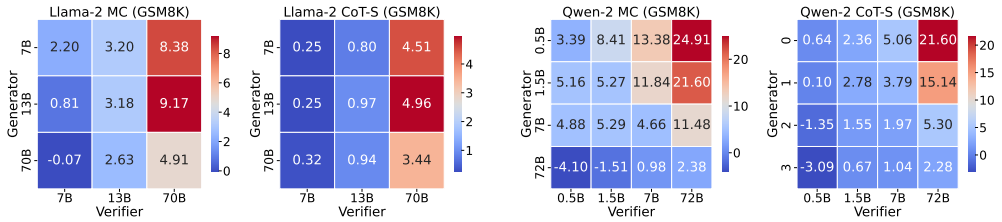


Figure 2: Gaps (%) in cross-improvement. For each row (a fixed generator), gaps increases as verifier capacity goes up. For each column (a fixed verifier), gap decreases as generator capacity goes up.

1. **Multiple Choice (MC)** (Li et al., 2023a; Gao et al., 2024) asks the LM to label responses as "Correct" and "Incorrect" and uses the probability of "Correct" as a continuous score.
2. **Chain of Thought (CoT)** (Wei et al., 2022) asks the LM to score responses and to provide the justification (i.e. CoT) and the score is parsed from the answer. Scores can be on a scale from 1 to 10 (**CoT-Score**) (Yuan et al., 2024; Liang et al., 2024) or binary (**CoT-Binary**).
3. **Tournament (To)** involves sampling a batch of generations and having a verifier compare generation pairs in a single elimination tournament to produce a new generation distribution. We can repeat the process until there is only one response left from the batch.

We consider the following models families: Qwen-1.5 (Bai et al., 2023), Qwen-2 (Yang et al., 2024), Llama-2 (Touvron et al., 2023), Llama-3, Llama-3.1 (Dubey et al., 2024) and Yi-1.5 (Young et al., 2024). To avoid the confounding effect of the post-training, all experiments in this paper are performed on *base* models. Finally, all inference in this paper is performed with vLLM (Kwon et al., 2023).

4 Scaling Properties of Generation-verification Gap

In this section, we conduct a comprehensive study on measuring the scaling property of the generation-verification gap due to its valuable practical guidance in both pre-training and downstream tasks (Kaplan et al., 2020; Hernandez et al., 2021; Isik et al., 2024; Ruan et al., 2024).

4.1 Scaling Results

We start with the GSM8K benchmark (Cobbe et al., 2021), with 1320 questions on the test data split. The ground truth utility $u(x, y) = 1$ if the end of response y is the correct answer to the question x, or $u(x, y) = 0$ otherwise. We compute $\mathsf{gap}(f)$ for each model f and verification method, and we record the full results in Tables 4 and 5. In particular, we observe the following phenomenon:

Small Models can not Self-improve. For small (in terms of pre-train flops) models such as Qwen-1.5 0.5B, Qwen-2 0.5B and Llama-2 7B, $\mathsf{gap}(f)$ is non-positive for nearly all verification methods, even though the models have non-trivial generation accuracy. We also observe this phenomenon in Pythia (Biderman et al., 2023) and OPT (Zhang et al., 2022) model families. We believe this result indicates that self-improvement requires a minimal level of instruction following and reasoning capabilities, which is not present in these small models. We will further illustrate this point in Section 4.4.

CoT Verification is More Stable than MC. Some MC verification incurs non-positive gap even for medium-sized models such as Qwen-1.5 14/32B and Llama-3/3.1 8B models, while CoT verification always has a positive gap for medium/large-sized models. Our results align with recent studies showing that MC evaluation might be unreliable, especially for small models (Dominguez-Olmedo et al., 2024). We perform a more in-depth analysis on this point in Appendix A.

$\mathsf{gap}_{\mathsf{rel}}(f)$ **Scales with Pre-training Flops.** We observe that with certain verification methods (such as CoT-Score), the relative gap grows monotonically with the pre-train flops, demonstrating a scaling property. We visualize the scaling results in Figure 1, where we plot $\mathsf{gap}_{\mathsf{rel}}(f)$ with respect to the log of pre-train flops. Specifically, we hypothesize that in the case where the verification elicits the scaling property, $\mathsf{gap}_{\mathsf{rel}}(f)$ scales linearly with respect to the log of the pre-train flops. However, note that we should not expect the slope for each model family to be the same. In Figure 7, we repeat the same plot for $\mathsf{gap}(f)$, but we do not observe a similar trend with the absolute gap.

Table 1: Gap (%) on Natural Question for Qwen-2 models. While all models have a non-trivial generation accuracy, all gaps are near 0, indicating that the task is unimprovable.

	0.5B	1.5B	7B	72B
Generation Accuracy	6.51	13.87	29.09	41.45
MC (top 0.8)	-0.06	0.04	0.79	0.28
MC ($\tau = 0.8$)	-0.05	0.02	-0.05	-0.05

Table 2: Generation accuracy, gap and relative gap (%) on Sudoku for Qwen-2 models. Only the 72B models can self-improve. For the 72B models, the improvement is around 200%.

	0.5B	1.5B	7B	72B
Generation Accuracy	0.66	0.62	2.09	8.82
gap	-0.09	0.04	-0.07	16.99
gap_{rel}	-0.15	-0.61	-0.01	20.81

4.2 Cross Verification

In self-improvement, both generator and verifier change when transitioning between different models. To better understand the relationship between generation/verification ability and model capacity, we perform a cross-verification study, where we only alter either the generator or the verifier at a time. We consider the Llama-2 and Qwen-2 model families, and the two most representative verification methods: MC with quantile threshold and CoT-Score. We present the results in Figure 2. We observe that the results are consistent with our intuition on the difficulty of verification: fix a generator model f, $\mathsf{gap}(f, g)$ increases as the model capacity (defined by pre-train flops) of the verifier model g increases. On the other hand, fix a verifier model g, $\mathsf{gap}(f, g)$ decreases as the model capacity of the generator model f increases, as the error of the generator model becomes more difficult to detect.

At first glance, the results seem to imply that selecting the largest model as the verifier, akin to a teacher-student setup, is advantageous. However, considering the computational costs associated with larger verifier models, this approach might be suboptimal. Alternatively, a weak-to-strong setup, where a smaller model verifies a larger one, might be more cost-effective, but our findings indicate that a positive gap cannot always be assured. We believe an interesting future direction is to explore the compute-optimal configuration for cross-verification. This, however, might require a combinatorially large number of experiments to pinpoint the optimal verifier for each generator.

Takeaway on scaling of self-improvement

LLMs demonstrate clear scaling trends in self- and cross-improvement:

- **Self-Improvement:** With stable verification, the relative gap grows monotonically with pre-training flops.
- **Cross-Improvement:** The gap scales **directly** with the verifier's flops and **inversely** with the generator's flops.

If the relative gap scales linearly with the logarithm of pre-training flops, this relationship could guide decisions on synthetic data generation strategies in self-improvement. Additionally, results from cross-verification suggest that a compute-optimal combination may exist to maximize efficiency in cross-improvement contexts.

4.3 Unimprovable Tasks

The primary objective of self-improvement is predicated on the assumption that "verification is easier than generation". As such, it is also worthwhile to consider tasks where such intuition would not hold. One such scenario involves factual tasks that require generating a factually correct answer to a trivia question. We hypothesize that the capability to generate a correct answer is contingent solely on whether the model has been trained with the relevant factual knowledge, and verification would provide little additional signal. To test this, we measure $\mathsf{gap}(f)$ on the Natural Question dataset (Kwiatkowski et al., 2019), where $u(x, y) = 1$ if y is one of the candidate answers to the question x, and $u(x, y) = 0$ otherwise. Our analysis on a test subset of 3610 questions, presented in Table 1, reveals that despite all models achieving non-trivial generation accuracy, the gap remains smaller than 1%, or is even negative, across all models. This suggests that certain tasks may not benefit from the current self-improvement framework. We include the full results in Table 6.

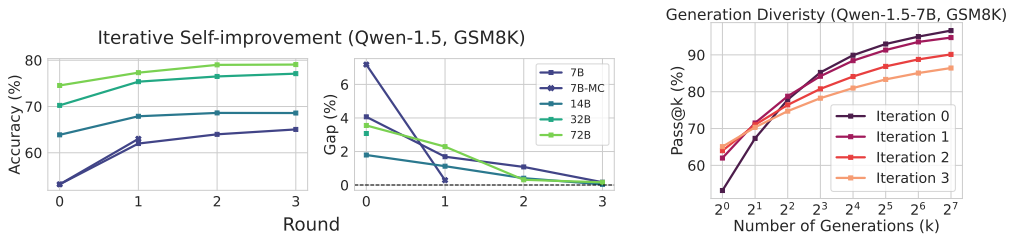


Figure 3: **Left**: The generation accuracy and gap along the iterative self-improvement process for Qwen-1.5 model family with CoT-Binary and MC verification. The horizontal line denotes 0.005. **Right**: The change of effective generation diversity along the iterative self-improvement process for Qwen-1.5 7B model, measured by pass@k for different k.

4.4 Sudoku

Generalized sudoku is a canonical example where the generation (NP-hard) is harder than the verification (P) (Haythorpe, 2016). We consider 4 by 4 sudoku puzzles, each with a unique solution, with 288 puzzles in total. We task the models to use CoT reasoning for both generation and verification. The results, presented in Table 2 and detailed further in Appendix D.3, reveal a surprising pattern: only the largest models, such as Qwen-1.5/2 72B and Llama 3.1 70B, exhibit non-trivial gaps. For these models, the improvement is indeed more significant ($50\% - 300\%$ improvement in accuracy) than the improvement in the math task.

While the second observation aligns with common intuition, the first may be unexpected, as most models demonstrate the ability to self-improve on tasks where the gap between generation and verification appears even narrower, such as in GSM8K. We hypothesize that, despite sudoku verification being simpler than generation, it still necessitates a certain level of reasoning and planning, even with explicit verification guidelines. This requirement is similar in mathematical tasks; however, it is likely that most models have been exposed to math verification during pre-training, unlike sudoku verification. Consequently, smaller models may lack the requisite reasoning capabilities to improve on sudoku tasks. Although our analysis is primarily post-hoc, an interesting avenue for future research would be to develop a metric to predict a model's "self-improvability" on specific tasks.

Takeaway on improvable tasks

LLMs do not universally self-improve across all tasks:

- **Trivia Tasks:** There is no significant generation-verification gap, given the similarity in complexity between generation and verification.
- **Sudoku:** Despite the exponential complexity separation between generation and verification in generalized sudoku, most models fail to self-improve. When improvement occurs, it is notably significant.

These findings suggest that the crucial factor for general self-improvement is the model's inherent reasoning and planning capabilities developed during pre-training.

5 Iterative Self-improvement

Building on our understanding of single-round self-improvement, a natural extension is to study iterative self-improvement. As there is no additional information introduced in the process, it is unrealistic to expect indefinite improvement. Thus in this section, we study the dynamics of the iterative self-improvement, and its relationship with model scales.

In our experiment, we perform iterative self-improvement on the Qwen-1.5 model family with CoT-Binary verification on GSM8K. We defer the finetuning hyperparameters to Appendix E. We present the results in Figure 3. We observe that 1) the gap diminishes nearly to zero within two or three rounds of self-improvement; this is consistent with the observation with previous works (Yuan et al., 2024; Liang et al., 2024). 2) The rate of saturation is similar across models with different capacities. 3) Notably, for the 7B and 14B models, the model accuracy at iteration 1 exceeds the sum

of the generation accuracy and the gap at iteration 0, i.e., $J(f_1) > J(f_0[w(\widehat{u}_{f_0})])$. This increase is attributed to improved adherence to the required answer format post-finetuning – the discrepancy between "flexible match" and "exact match" (extract the answer from the required answer format) disappears after the first round. As we argued in the previous section, this additional accuracy gain is not due to the self-improvement capability of the model, and thus our modular study reduces the confounding factors in understanding the self-improvement capability of the model.

To compare the dynamics between verification methods, in Figure 3 we also plot MC (top 0.7) verification for the 7B model. We observe that the gap immediately drops to near 0 after the first round of self-improvement, and thus multi-round self-improvement with MC verification is unlikely. This rapid saturation is consistent across other thresholds for MC verification. We provide a more detailed study on the cause of this phenomenon in Appendix A.1.

We also examine the "effective diversity" of generations throughout the iterative self-improvement process using the metric pass@k[3]. We present the results in Figure 3. We observe when k is small, pass@k increases with the number of rounds of self-improvement, validating the success of the self-improvement process. However, when k is large, pass@k decreases with the number of iterations, indicating that the diversity of the generations is reduced through the self-improvement process. This trend may result from the model's inability to verify rare, yet correct, answers, potentially leading to convergence on incorrect solutions during the self-improvement process.

Takeaway on iterative self-improvement

LLMs can perform iterative self-improvement with an effective verification method:

- **Saturation Limit:** Without new information, iterative self-improvement typically saturates after two or three rounds, regardless of the model's capacity (measured in pre-training flops).
- **Cause of Saturation:** A potential cause for this saturation is a decrease in effective diversity, leading to convergence on incorrect answers for certain questions.

Addressing the reduction in diversity could potentially extend the duration and effectiveness of the self-improvement process.

6 Conclusion and Discussion

In this paper, we conduct a comprehensive and modular study on the LLM self-improvement framework through multiple model families, tasks and verification mechanisms. We structure the mathematical framework of the self-improvement process and pinpoint the generation-verification gap as a critical metric. Our results reveal several intriguing properties such as the scaling properties of the relative gap, saturation of iterative self-improvement and enhancement of verification via ensemble methods. These insights are likely to have practical implications for improving pre-training, post-training, and test-time inference. Additionally, our research opens several promising avenues for future exploration:

- While our scaling analysis is primarily observational (Ruan et al., 2024), pursuing a more extensive scaling law study (Kaplan et al., 2020) based on our preliminary findings could provide robust empirical guidelines.
- Our results hint at an inference-time scaling law (Wu et al., 2024) is possible for self-improvement (or with cross-improvement (c.r. Section 4.2). Identifying compute-optimal methods for self-improvement across different tasks remains a critical challenge.
- The decline in the effective diversity of generations during iterative self-improvement presents a significant obstacle. Developing strategies to mitigate this issue offers considerable empirical benefits.
- The distinct non-overlap property of verification mechanisms, despite their functional similarities, suggests that combining compositional verification could significantly enhance self-improvement. Exploring this potential further could yield fruitful results.

[3]Given a question, pass@k is 1 if at least one of the k generations of the model is correct, or 0 otherwise.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*, 2024.

Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard G Baraniuk. Self-consuming generative models go mad. *arXiv preprint arXiv:2307.01850*, 2023.

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D Chang, and Prithviraj Ammanabrolu. Critique-out-loud reward models. *arXiv preprint arXiv:2408.11791*, 2024.

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

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

Hritik Bansal, Arian Hosseini, Rishabh Agarwal, Vinh Q Tran, and Mehran Kazemi. Smaller, weaker, yet better: Training llm reasoners via compute-optimal sampling. *arXiv preprint arXiv:2408.16737*, 2024.

Quentin Bertrand, Avishek Joey Bose, Alexandre Duplessis, Marco Jiralerspong, and Gauthier Gidel. On the stability of iterative retraining of generative models on their own data. *arXiv preprint arXiv:2310.00429*, 2023.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Martin Briesch, Dominik Sobania, and Franz Rothlauf. Large language models suffer from their own output: An analysis of the self-consuming training loop. *arXiv preprint arXiv:2311.16822*, 2023.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Jonathan D Chang, Kiante Brantley, Rajkumar Ramamurthy, Dipendra Misra, and Wen Sun. Learning to generate better than your llm. *arXiv preprint arXiv:2306.11816*, 2023.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

Cheng-Han Chiang and Hung-yi Lee. Can large language models be an alternative to human evaluations? *arXiv preprint arXiv:2305.01937*, 2023.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

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

Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70, 2005.

Elvis Dohmatob, Yunzhen Feng, and Julia Kempe. Model collapse demystified: The case of regression. *arXiv preprint arXiv:2402.07712*, 2024.

Ricardo Dominguez-Olmedo, Florian E Dorner, and Moritz Hardt. Training on the test task confounds evaluation and emergence. *arXiv preprint arXiv:2407.07890*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, et al. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data. *arXiv preprint arXiv:2404.01413*, 2024.

Nate Gillman, Michael Freeman, Daksh Aggarwal, Chia-Hong Hsu, Calvin Luo, Yonglong Tian, and Chen Sun. Self-correcting self-consuming loops for generative model training. *arXiv preprint arXiv:2402.07087*, 2024.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Ryuichiro Hataya, Han Bao, and Hiromi Arai. Will large-scale generative models corrupt future datasets? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 20555–20565, 2023.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024.

Michael Haythorpe. Reducing the generalised sudoku problem to the hamiltonian cycle problem. *AKCE International Journal of Graphs and Combinatorics*, 13(3):272–282, 2016.

Elad Hazan. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

Liang Huang and David Chiang. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pp. 144–151, 2007.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Berivan Isik, Natalia Ponomareva, Hussein Hazimeh, Dimitris Paparas, Sergei Vassilvitskii, and Sanmi Koyejo. Scaling laws for downstream task performance of large language models. *arXiv preprint arXiv:2402.04177*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kalpesh Krishna, Yapei Chang, John Wieting, and Mohit Iyyer. Rankgen: Improving text generation with large ranking models. *arXiv preprint arXiv:2205.09726*, 2022.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Haoran Li, Qingxiu Dong, Zhengyang Tang, Chaojun Wang, Xingxing Zhang, Haoyang Huang, Shaohan Huang, Xiaolong Huang, Zeqiang Huang, Dongdong Zhang, et al. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *arXiv preprint arXiv:2402.13064*, 2024.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 5 2023a.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023b.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Yiming Liang, Ge Zhang, Xingwei Qu, Tianyu Zheng, Jiawei Guo, Xinrun Du, Zhenzhu Yang, Jiaheng Liu, Chenghua Lin, Lei Ma, et al. I-sheep: Self-alignment of llm from scratch through an iterative self-enhancement paradigm. *arXiv preprint arXiv:2408.08072*, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Gonzalo Martínez, Lauren Watson, Pedro Reviriego, José Alberto Hernández, Marc Juarez, and Rik Sarkar. Combining generative artificial intelligence (ai) and the internet: Heading towards evolution or degradation? *arXiv preprint arXiv:2303.01255*, 2023.

Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. Generating training data with language models: Towards zero-shot language understanding. *Advances in Neural Information Processing Systems*, 35:462–477, 2022.

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

Yangjun Ruan, Chris J Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of language model performance. *arXiv preprint arXiv:2405.10938*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

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

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

Tianxiang Sun, Xiaotian Zhang, Zhengfu He, Peng Li, Qinyuan Cheng, Hang Yan, Xiangyang Liu, Yunfan Shao, Qiong Tang, Xingjian Zhao, et al. Moss: Training conversational language models from synthetic data. *arXiv preprint arXiv:2307.15020*, 7:3, 2023.

Rohan Taori and Tatsunori Hashimoto. Data feedback loops: Model-driven amplification of dataset biases. In *International Conference on Machine Learning*, pp. 33883–33920. PMLR, 2023.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024a.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022b.

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

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. In *Forty-first International Conference on Machine Learning*, 2024.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models. *arXiv preprint arXiv:2406.16838*, 2024.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Wenda Xu, Guanglei Zhu, Xuandong Zhao, Liangming Pan, Lei Li, and William Wang. Pride and prejudice: Llm amplifies self-bias in self-refinement. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15474–15492, 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Asaf Yehudai, Boaz Carmeli, Yosi Mass, Ofir Arviv, Nathaniel Mills, Assaf Toledo, Eyal Shnarch, and Leshem Choshen. Genie: Achieving human parity in content-grounded datasets generation. *arXiv preprint arXiv:2401.14367*, 2024.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Di Zhang, Jiatong Li, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*, 2024a.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction, 2024b. URL https://arxiv.org/abs/2408.15240.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Tong Zhang. From ε-entropy to kl-entropy: Analysis of minimum information complexity density estimation. *The Annals of Statistics*, pp. 2180–2210, 2006.

Xuanchang Zhang, Wei Xiong, Lichang Chen, Tianyi Zhou, Heng Huang, and Tong Zhang. From lists to emojis: How format bias affects model alignment. *arXiv preprint arXiv:2409.11704*, 2024c.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. Starling-7b: Improving llm helpfulness & harmlessness with rlaif, 2023.

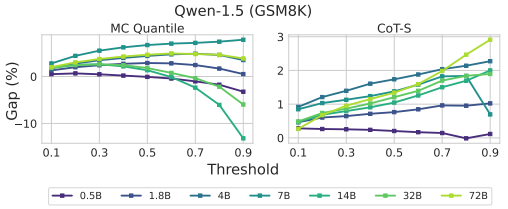


Figure 4: Change in the gap as we vary the threshold for each verification method. We only present the results for MC with quantile threshold and CoT-Score, because CoT-Binary's gap does not change as we change the threshold.

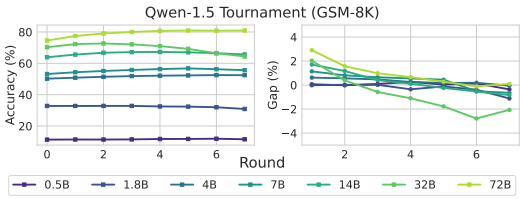


Figure 5: Change in the generation accuracy of the filtered dataset (left) and gap (right) with respect to the round of tournament. The right figure plots the gap with respect to the accuracy from the previous round instead of the base accuracy.

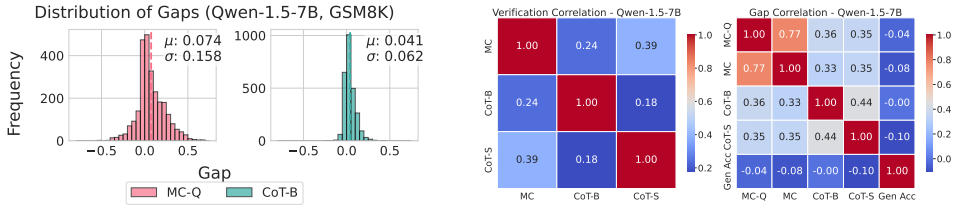


Figure 6: **Left:** The empirical distribution of gaps of MC Quantile and CoT-Binary of Qwen-1.5-7B on GSM8K. We cluster gaps in bins of intervals with width 0.005. We label the mean (μ) and standard deviation (σ) of each distribution. **Right:** the correlation plot of the output of each verification $\widehat{u}$ and the correlation plot of the gap from each verification and generation accuracy.

A A Fine-grained Study on Verification

Among the three components of self-improvement, the verification step offers the most flexibility, whereas generation and update follow more fixed procedures. Therefore, this section presents a detailed examination of the verification mechanisms. Through this focused study, we aim to uncover practical ways to enhance the overall self-improvement process.

A.1 Generalization of Verification Thresholds

In the rejection sampling framework, selecting an appropriate threshold for filtering generations based on verification is a crucial practical concern. We explore verification methods adaptable to various thresholds, including MC, CoT-Score, and Tournament. Our analysis focuses on how the performance gap changes with different thresholds for these methods. We present results for the Qwen-1.5 model family using MC, MC-quantile and CoT-Score in Figure 4, and results using Tournament in Figure 5. For tournament verification, the threshold is defined as the number of rounds of tournament. We defer the results of other models to Appendix D.4. In Tournament, we note that the gap with respect to the accuracy in the previous iteration generally decreases monotonically; this trend occurs as the verification error is more likely to be exploited by the remaining generations at later stages of the tournament.

We observe that the relationship between the gap and the threshold is consistent across most models when using any fixed verification method. For MC and Tournament, the gap follows a concave curve relative to the threshold, while for CoT-Score, it increases monotonically. In addition, the majority of models agrees on the optimal thresholds for each verification across model families, and we use the optimal thresholds to report our results for this paper. However, in general, one should not expect the optimal threshold transfers between different tasks. That said, the consistency in threshold effects suggests a practical approach: if determining the optimal threshold for a large model is costly, one might first establish it for a smaller model and then apply it to the larger model.

A.2 Correlation Studies between Verification Methods

As the verification methods are functionally similar, one might question the necessity to study multiple verifications. To address this, We start by comparing the distribution of gaps induced by different

Table 3: Gaps of combining verification with AND operation on Qwen-2 model family. For MC, we use a quantile threshold of 0.8, for CoT-S, we use the global threshold of 8, as they are the best-performing thresholds from the analysis in the previous sections.

	MC	CoT-B	CoT-S	MC+CoT-B	MC+CoT-S	CoT-B+CoT-S	All
0.5B	3.39	0.21	0.64	3.12	3.01	0.72	3.11
1.5B	5.27	1.26	2.78	5.08	7.68	3.15	7.61
7B	4.66	2.36	1.97	5.24	4.96	3.35	5.46
72B	2.38	2.51	2.08	3.21	3.13	3.35	3.65

verification methods. We use Qwen-1.5 7B model as an example and we present the bar plot of the gap distribution in Figure 6. Notably, there are significant discrepancies, especially between MC and CoT methods – the variance in the gap is considerably larger with MC. This aligns with our previous findings in Section 5 where iterative self-improvement with MC verification saturates more quickly than with CoT. While CoT slightly improves the accuracy on most questions, MC will drive the accuracy to the extreme in one round of self-improvement. We observe that this pattern holds across all models, as detailed in Figure 10.

To further compare the verification methods, we calculate the Pearson correlation coefficient between the outputs of the proxy utility $\widehat{u}$ the gaps, shown in Figure 6. We use Qwen-1.5 7B as an example and defer full results to Figs. 11 and 12. We observe that the correlations between $\widehat{u}$ are generally low, suggesting potential benefits in combining different verification methods. Notably, the correlation between MC verifications and the correlation between CoT verifications are generally the highest, and larger models tend to have a higher correlation between the gaps. Surprisingly, the gaps of any verification method do not positively correlate with generation accuracy, reinforcing the idea that the relative gap may be a more appropriate metric for measuring self-improvement capability.

A.3 Improvement via Ensemble

The non-overlap property of different verification methods suggests the potential for enhanced verification performance through their combination. We again focus on the rejection sampling setup. In the rejection sampling framework, we employ a logical AND operation, keeping samples only if they pass all verification filters. We provide an example result from Qwen-2 model family in Table 8, and we defer the full result to Appendix D.6. We observe that combining any verifications with non-trivial gaps improves the verification performance (with the exception of CoT for 0.5B model with near 0 gap). This promising outcome indicates that despite functional similarities, different verification mechanisms can still be combined to improve self-improvement efficacy. The consistent improvements across different model sizes also suggest that strategies developed using smaller models can be effectively applied to larger ones, if all verifications are valid.

Takeaway on verification mechanisms

A fine-grained study on verification reveals several implications for practice:

- **Verification Consistency:** The distribution of the gaps and optimal verification threshold typically generalize across models.
- **Verification Distinction:** Despite functional similarities, the outputs and gaps of verification methods show non-trivial differences among each other.
- **Ensemble Heuristic:** Simple verification ensemble heuristics can improve the performance.

The consistency result suggests that configurations from smaller models can be applied to larger ones to avoid the costs associated with tuning big models. The discovery that simple ensemble techniques can enhance performance highlights the potential for more sophisticated algorithms to further improve self-improvement strategies.

B Related Works

Synthetic Data and Self-Training. Training LLMs with a mixture of "real" data (generated by human) and synthetic data has been the standard protocol nowadays given the limited number of human data and extensive amount of data required as we scale up the LLM training. Initial studies generated synthetic data from more powerful models (Gunasekar et al., 2023; Li et al., 2023b; Team

et al., 2023; Sun et al., 2023; Taori et al., 2023; Zhu et al., 2023; Wei et al., 2024), while recent approaches involve models training on their own outputs (Achiam et al., 2023; Adler et al., 2024; Dubey et al., 2024; Yang et al., 2024; Hui et al., 2024).

On the theoretical front, extensive research has explored the phenomenon of model collapse during self-training and strategies to counter this degenerate behavior (Hataya et al., 2023; Martínez et al., 2023; Bertrand et al., 2023; Briesch et al., 2023; Taori & Hashimoto, 2023; Alemohammad et al., 2023; Dohmatob et al., 2024; Gillman et al., 2024).

LLM Self-improvement. One of the most effective strategies to prevent model collapse during self-training is the use of a reliable verifier (Gillman et al., 2024). In the absence of additional resources like labeled data or an external oracle, models can utilize their own verification capabilities. This is particularly effective if the model is more proficient at verification than generation. Numerous studies have proposed variations of self-improvement algorithms based on this principle, resulting in significant practical achievements (Zelikman et al., 2022; Wang et al., 2022b; Huang et al., 2022; Singh et al., 2023; Chen et al., 2023; Madaan et al., 2024; Xu et al., 2024; Yuan et al., 2024; Liang et al., 2024; Wang et al., 2024b; Shinn et al., 2024; Zelikman et al., 2024). Previous research, however, often relied on additional data to enhance verification, used surrogate metrics for improvement, or limited their focus to a small number of models. In this work, instead of proposing any new algorithm, we aim to rigorously analyze the self-improvement phenomenon in a controlled, comprehensive manner.

Improving Test-time Inference with Additional Computation. Recent research has demonstrated that the performance of models can be enhanced by allocating more computational resources to inference (Welleck et al., 2024) . This typically leverages the observation that LLMs can make diverse generations, and with a small probability it can generate high-quality responses (Li et al., 2022; Brown et al., 2024; Bansal et al., 2024). Thus with oracle verifier, or with training a high-quality reward model, model performance can be improved by simply making multiple generations and selecting the best ones according to the oracle or the reward model (Cobbe et al., 2021). There are also works on training process-based reward models (Lightman et al., 2023) to improve the model's reasoning results (Luo et al., 2024; Wang et al., 2024a; Zhang et al., 2024a).

Concurrently there are also works on the test-time scaling law which investigates the computational trade-off between the model size (which determines the number of generations given a computation budget) and final accuracy combined with reward model or oracle (Wu et al., 2024; Snell et al., 2024). The results provide the compute-optimal solution for test-time inferencing with a fixed compute budget and a fixed verifier. We believe a better understanding of self-improvement can also lead to a test-time scaling law without an external verifier.

LLM-as-a-Judge. LLM-as-a-judge refers to using an LLM to verify the generation of some other (or the same) LLM (Chiang et al., 2023; Zheng et al., 2023; Bubeck et al., 2023; Chiang & Lee, 2023; Zhou et al., 2024). Recently the same idea has also been applied to train a generative reward model (Ankner et al., 2024; Zhang et al., 2024b). Having a model that can verify its own generation is one of the key components of self-improvement, and in this work, we perform a fine-grained study on various types of LLM verification mechanisms.

Reranking Algorithms. The self-improvement framework we study in this paper relies on reweighting the generation distribution. Prior to self-improvement, the reranking algorithm has already been widely applied in various NLP applications (Collins & Koo, 2005; Huang & Chiang, 2007; Stiennon et al., 2020; Cobbe et al., 2021; Krishna et al., 2022; Lightman et al., 2023).

C Verification Mechanisms

In this section, we provide a more complete description of the verification mechanism we use throughout the paper.

- **Multiple Choice (MC)**: Multiple choice verification asks the LM to label responses as "Correct" and "Incorrect". Let $\mathsf{prom}_{\mathsf{mc}}$ be a verification prompt and denote $\widehat{u}_f^{\mathsf{mc}}(x, y)$ a utility derived from the verifier generating a single token t^+, t^-, representing the word "Correct" and "Incorrect" respectively. The score uses the logits from these tokens to find the probability of "Correct"

conditioned on the next token being "Correct" or "Inorrect":

$$\widehat{u}_f^{\mathsf{mc}}(x, y) := \frac{f(t^+ \mid x, y, \mathsf{prom}_{\mathsf{mc}})}{f(t^+ \mid x, y, \mathsf{prom}_{\mathsf{mc}}) + f(t^- \mid x, y, \mathsf{prom}_{\mathsf{mc}})} .$$

- **Chain of Thought (CoT)**: CoT verification asks the LM to score responses and to provide the CoT. Denote by $\mathcal{S} \subset \mathbb{R}$ the set of verification scores and by $\mathsf{prom}_{\mathcal{S}}$ a verification prompt. We can define a utility

$$\widehat{u}_f^{\mathcal{S}}(x, y) := \mathbb{E}_{s,z \sim f(\cdot \mid x, y, \mathsf{prom}_{\mathcal{S}})}[s(z)],$$

  where z is the verification CoT, and $s(z) \in \mathcal{S}$ is the score extracted from the CoT. In our experiments we consider two versions, CoT-Score with $\mathcal{S} = [10]$ and CoT-Binary with $\mathcal{S} = \{0, 1\}$.

- **Tournament (To)**: The tournament verification does not directly fit the utility framework described in Section 2. Rather, this verification procedure involves comparisons of a batch of generations to provide a modified distribution[4]. Given a comparison prompt $\mathsf{prom}_{\mathsf{com}}$, we perform a tournament-style elimination over a batch of 2^r generations by comparing disjoint pairs in each round until a single generation remains. Let $\mathcal{Y}^{(0)} = y_1, y_2, \ldots, y_{2^r}$ be the initial set of generations. At round k, the set $\mathcal{Y}^{(k)}$ contains 2^{r-k} remaining generations. These are split into disjoint pairs $(y_i, y_j) \in \mathcal{Y}^{(k)}$. Each pair is compared using the prompt $\mathsf{prom}_{\mathsf{com}}$, and the verifier's output $s \in A, B$ indicates the preferred generation:

$$y_{\mathsf{win}} = \begin{cases} y_i & \text{if } f(\cdot \mid y_i, y_j, \mathsf{prom}_{\mathsf{com}}) = A, \\ y_j & \text{if } f(\cdot \mid y_i, y_j, \mathsf{prom}_{\mathsf{com}}) = B. \end{cases}$$

  where y_{win} is the winner of the pairwise comparison and advances to the next round. After each round k, the set of winners $\mathcal{Y}^{(k+1)}$ contains half the number of generations. This process is repeated until $k = r$, leaving only one generation, the lone element in $\mathcal{Y}^r$.

D Additional Results

D.1 Additional Results for Section 4.1

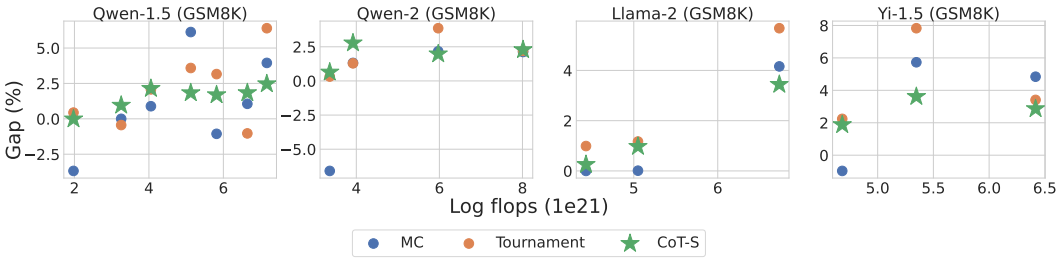


Figure 7: Scaling phenomenon of the gap with respect to the pretrain flops. The x-axis is the log of pretrain flops, and the y-axis is the relative gap. MC denotes Multiple Choice verification with quantile threshold 0.8 (top 0.8), CoT-S denotes CoT-Score verification with global threshold 8, and To denotes Tournament verification with 5 rounds.

[4]This batch-style distribution weighting also applies to strategies like top-k wherein we take the highest k utility generations for a particular question.

Table 4: Gap on GSM-8K for all models. For each verification, "top n" denotes taking the threshold as the n quantile of the proxity utility for each prompt, and $\tau = n$ denotes taking the threshold as n for all prompts. All numbers denote the percentage.

Name	Size	Accuracy	MC				CoT			To
			top 0.7	top 0.8	$\tau = 0.7$	$\tau = 0.8$	Bin	S ($\tau = 8$)	S ($\tau = 9$)	round 5
Qwen-1.5	0.5B	11.31	-0.62	-1.42	-2.27	-3.68	-0.02	-0.01	0.12	0.44
	1.8B	32.81	3.06	2.70	-0.01	0.00	1.58	0.95	1.02	-0.44
	4B	50.21	4.85	4.63	0.48	0.89	2.36	2.14	2.27	2.08
	7B	53.17	7.18	7.42	3.68	6.13	4.07	1.84	0.70	3.59
	14B	63.87	-2.04	-5.61	2.36	-1.06	1.79	1.69	2.00	3.16
	32B	70.25	-0.22	-1.98	1.72	1.06	3.07	1.84	1.90	-1.03
	72B	74.55	4.84	4.75	2.00	3.95	3.55	2.47	2.91	6.40
Qwen-2	0.5B	26.19	4.59	3.39	3.81	-6.59	0.21	0.64	0.72	0.32
	1.5B	48.82	6.09	5.27	5.02	1.32	1.26	2.78	2.80	1.29
	7B	76.42	4.69	4.66	3.90	2.17	2.36	1.97	2.08	3.86
	72B	81.69	2.39	2.38	0.89	2.12	2.51	2.08	2.45	2.20
Llama-2	7B	11.64	2.33	2.20	0.10	0.00	0.16	0.25	0.23	0.99
	13B	21.57	3.45	3.18	-0.13	0.01	1.13	0.97	1.01	1.17
	70B	48.42	5.14	4.91	4.77	4.16	3.98	3.44	3.45	5.68
Llama-3	8B	45.66	5.34	5.33	-0.50	0.44	2.67	2.10	2.13	4.08
	70B	74.19	5.06	4.52	4.68	1.35	3.89	2.59	2.72	2.00
Llama-3.1	8B	49.31	4.68	4.57	-2.25	-0.24	3.37	2.09	2.05	4.78
	70B	71.71	6.88	6.77	6.00	0.93	3.29	2.71	2.88	-0.40
Yi-1.5	6B	55.53	4.40	4.27	2.98	-0.97	2.01	1.88	1.95	2.24
	9B	61.04	7.74	7.50	5.61	5.73	2.32	3.61	3.72	7.83
	34B	73.71	6.29	6.23	3.34	4.84	2.49	2.86	2.95	3.41

D.2 Additional Results for Section 4.3

Model	Qwen-1.5						
	0.5B	1.8B	4B	7B	14B	32B	72B
Generation Accuracy	6.20	11.40	17.16	21.20	26.83	35.79	39.97
MC (top 0.2)	-0.62	-0.02	-1.20	-1.15	-1.99	-0.43	-0.75
MC ($\tau = 0.8$)	-0.51	-0.32	-0.72	-1.07	-1.21	-0.10	0.32

Model	Qwen-2			
	0.5B	1.5B	7B	72B
Generation Accuracy	6.51	13.87	29.09	41.45
MC (top 0.2)	-0.06	0.04	0.79	0.28
MC ($\tau = 0.8$)	-0.05	0.02	-0.05	-0.05

Model	Llama-2		
	7B	13B	70B
Generation Accuracy	25.52	41.00	29.09
MC (top 0.2)	-0.96	0.76	0.30
MC ($\tau = 0.8$)	-0.81	-2.31	-0.44

Model	Llama-3	
	8B	70B
Generation Accuracy	30.40	45.59
MC (top 0.2)	0.27	0.32
MC ($\tau = 0.8$)	-0.23	-0.41

Table 5: Relative gaps on GSM-8K for all models. For each verification, "top n" denotes taking the threshold as the n quantile of the proxity utility for each prompt, and $\tau = n$ denotes taking the threshold as n for all prompts. All numbers denote the percentage

Name	Size	Accuracy	MC				CoT			To
			top 0.7	top 0.8	$\tau = 0.7$	$\tau = 0.8$	Bin	S ($\tau = 8$)	S ($\tau = 9$)	round 5
Qwen-1.5	0.5B	11.31	-0.70	-1.60	-2.56	-4.15	-0.03	-0.13	0.04	0.59
	1.8B	32.81	4.55	4.01	-0.01	-0.01	4.45	3.10	3.15	-0.39
	4B	50.21	9.75	9.31	0.96	1.80	8.22	6.92	8.12	5.24
	7B	53.17	15.34	15.84	7.87	13.08	14.06	5.50	-0.35	10.26
	14B	63.87	-5.64	-15.54	6.52	-2.94	7.77	8.00	9.02	12.19
	32B	70.25	-0.75	-6.67	5.78	3.57	15.31	8.40	8.49	-4.55
	72B	74.55	19.01	18.65	7.87	15.52	21.82	14.96	16.97	32.06
Qwen-2	0.5B	26.19	6.22	4.59	5.16	-8.93	0.44	1.58	1.79	1.31
	1.5B	48.82	11.90	10.30	9.82	2.58	4.45	9.96	10.11	5.26
	7B	76.42	19.89	19.74	16.53	9.18	16.46	13.73	14.34	17.17
	72B	81.69	13.07	13.00	4.87	11.57	20.16	14.34	19.92	17.84
Llama-2	7B	11.64	2.64	2.49	0.11	0.00	0.25	0.39	0.37	1.91
	13B	21.57	4.40	4.05	-0.16	0.02	2.45	1.79	1.82	2.88
	70B	48.42	9.97	9.52	9.25	8.07	13.77	12.22	12.07	18.57
Llama-3	8B	45.66	9.62	9.60	-0.90	0.80	8.51	6.16	6.70	12.55
	70B	74.19	18.08	16.13	16.71	4.84	19.09	13.24	13.74	11.28
Llama-3.1	8B	49.31	8.97	8.77	-4.31	-0.47	11.74	7.02	6.89	14.97
	70B	71.71	22.83	22.47	19.91	3.09	16.68	12.65	13.59	-1.03
Yi-1.5	6B	55.53	9.89	9.60	6.70	-2.17	9.10	6.60	6.69	10.64
	9B	61.04	19.86	19.25	14.40	14.70	10.09	14.29	14.35	24.78
	34B	73.71	23.94	23.68	12.70	18.40	15.02	16.69	16.96	8.89

Model	Llama-3.1	
	8B	70B
Generation Accuracy	27.75	45.13
MC (top 0.2)	0.42	0.44
MC ($\tau = 0.8$)	-0.59	-0.37

Model	Yi-1.5		
	6B	9B	34B
Generation Accuracy	22.82	25.94	35.31
MC (top 0.2)	0.09	0.21	0.24
MC ($\tau = 0.8$)	-0.07	0.61	0.30

Table 6: Gap on Natural Question for all models. With non-trivial generation accuracy, all gaps are near 0, indicating that the task is non-improvable.

D.3 Additional Results for Section 4.4

Model	Qwen-1.5						
	0.5B	1.8B	4B	7B	14B	32B	72B
Generation Accuracy	0.43	1.00	0.88	0.95	1.57	2.67	2.02
Gap	0.02	-0.03	-0.15	-0.64	0.22	0.07	1.23
Relative Gap	-0.10	-2.80	-1.39	-3.06	0.67	-1.25	1.14

Model	Qwen-2					
	0.5B	1.5B	7B	72B	7B-Instruct	72B-Instruct
Generation Accuracy	0.66	0.62	2.09	8.82	2.16	8.15
Gap	-0.09	0.04	-0.07	16.99	0.13	22.97
Relative Gap	-0.15	-0.61	-0.01	20.81	0.20	26.40

Model	Llama-2		
	7B	13B	70B
Generation Accuracy	0.82	0.89	0.86
Gap	-0.13	-0.63	-0.86
Relative Gap	0.45	-2.02	-3.57

Model	Llama-3	
	8B	70B
Generation Accuracy	1.39	1.63
Gap	-1.10	-0.84
Relative Gap	-15.4	-36.12

Model	Llama-3.1	
	8B	70B
Generation Accuracy	1.11	1.68
Gap	-0.19	5.5
Relative Gap	-4.52	6.87

Model	Yi-1.5		
	6B	9B	34B
Generation Accuracy	0.59	1.29	4.48
Gap	-0.60	0.22	-1.75
Relative Gap	-0.94	0.43	-0.77

Table 7: Generation accuracy, gap and relative gap on Sudoku for all models.

D.4 Additional Results for Appendix A.1

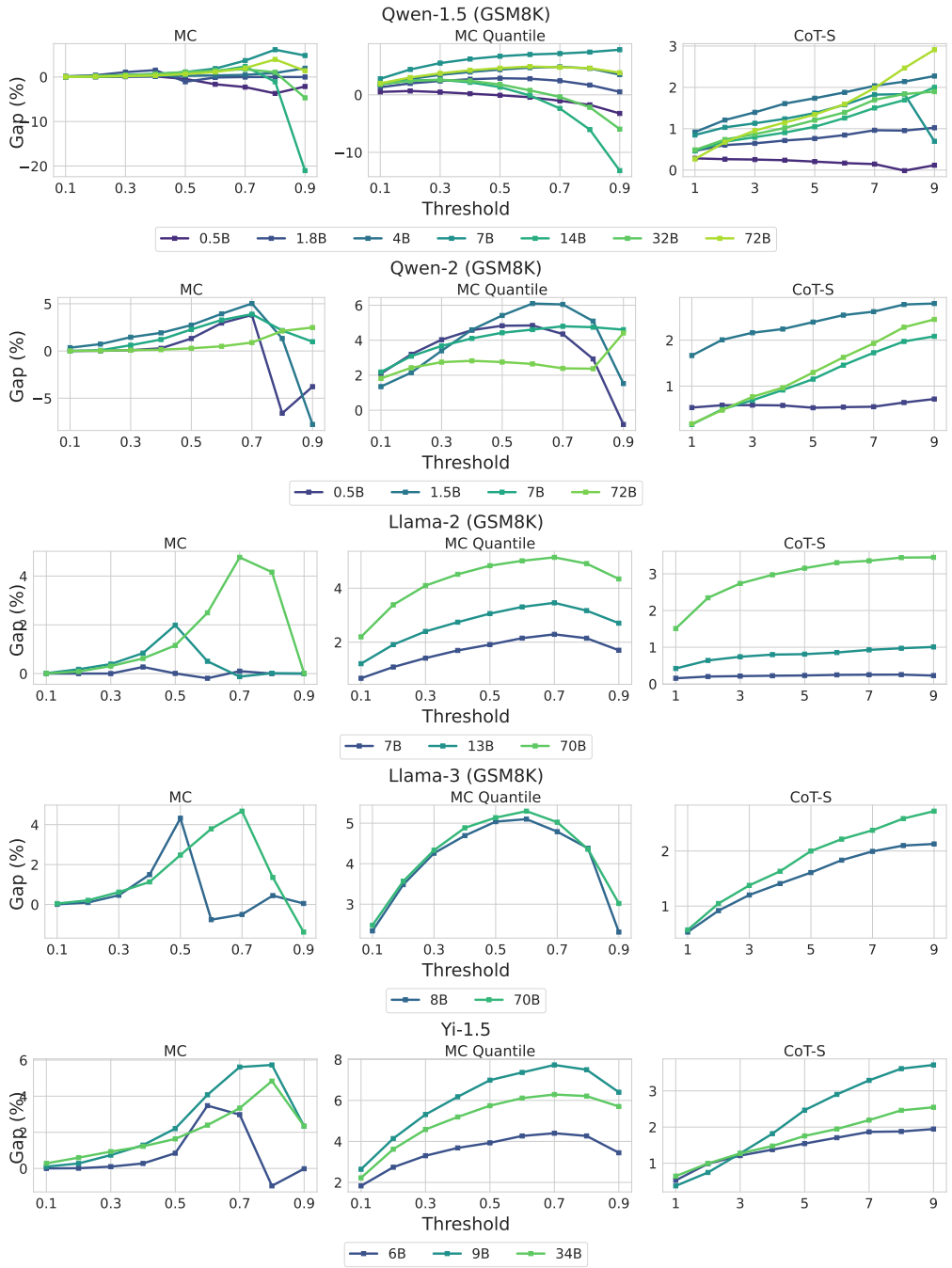
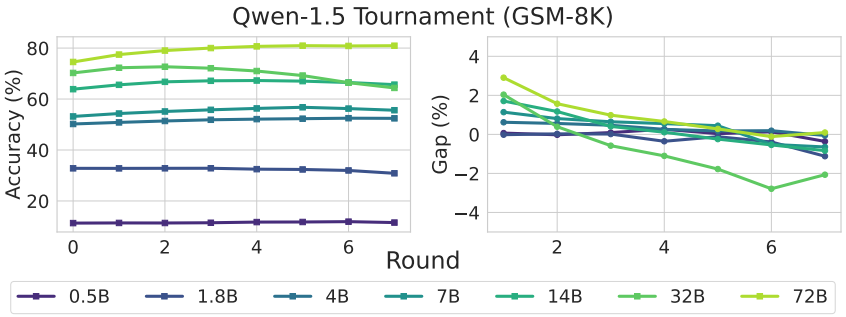
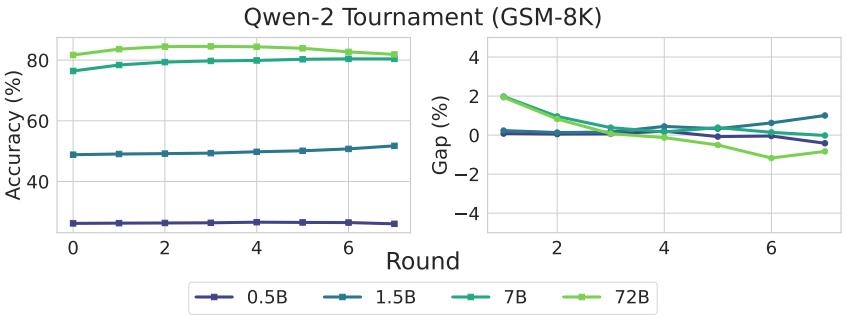
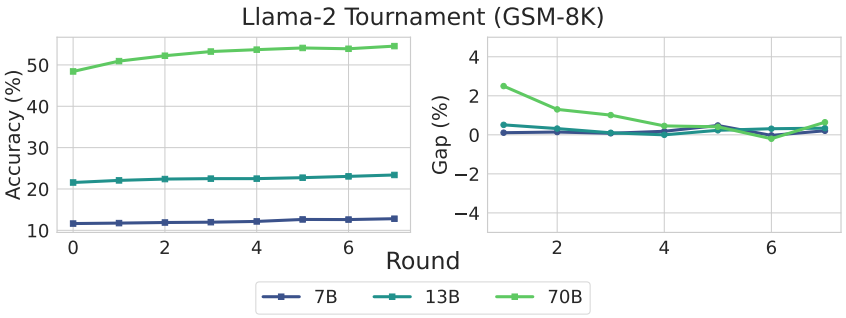
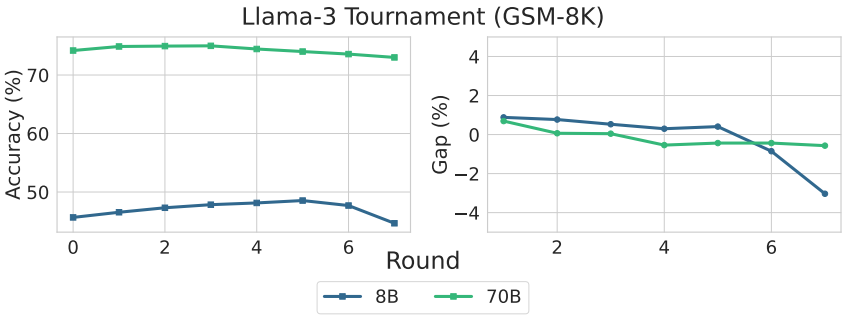


Figure 8: Change in the gap as we vary the threshold for each verification method. We only present the results for MC with global threshold, quantile threshold and CoT-Score, because CoT-Binary's gap does not change as we change the threshold.


Qwen-1.5 Tournament (GSM-8K)
Accuracy (%)
Gap (%)
Round
0.5B
1.8B
4B
7B
14B
32B
72B


Qwen-2 Tournament (GSM-8K)
Accuracy (%)
Gap (%)
Round
0.5B
1.5B
7B
72B


Llama-2 Tournament (GSM-8K)
Accuracy (%)
Gap (%)
Round
7B
13B
70B


Llama-3 Tournament (GSM-8K)
Accuracy (%)
Gap (%)
Round
8B
70B

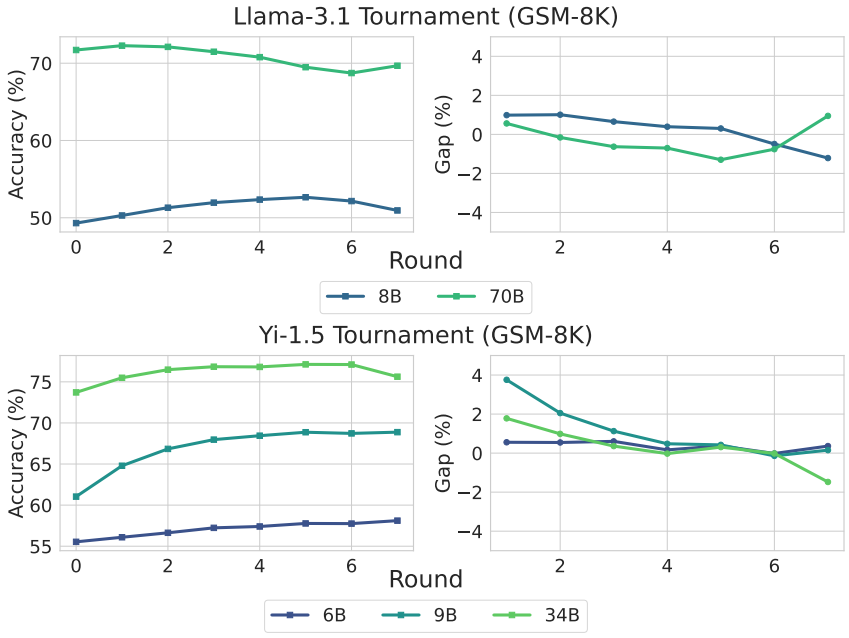
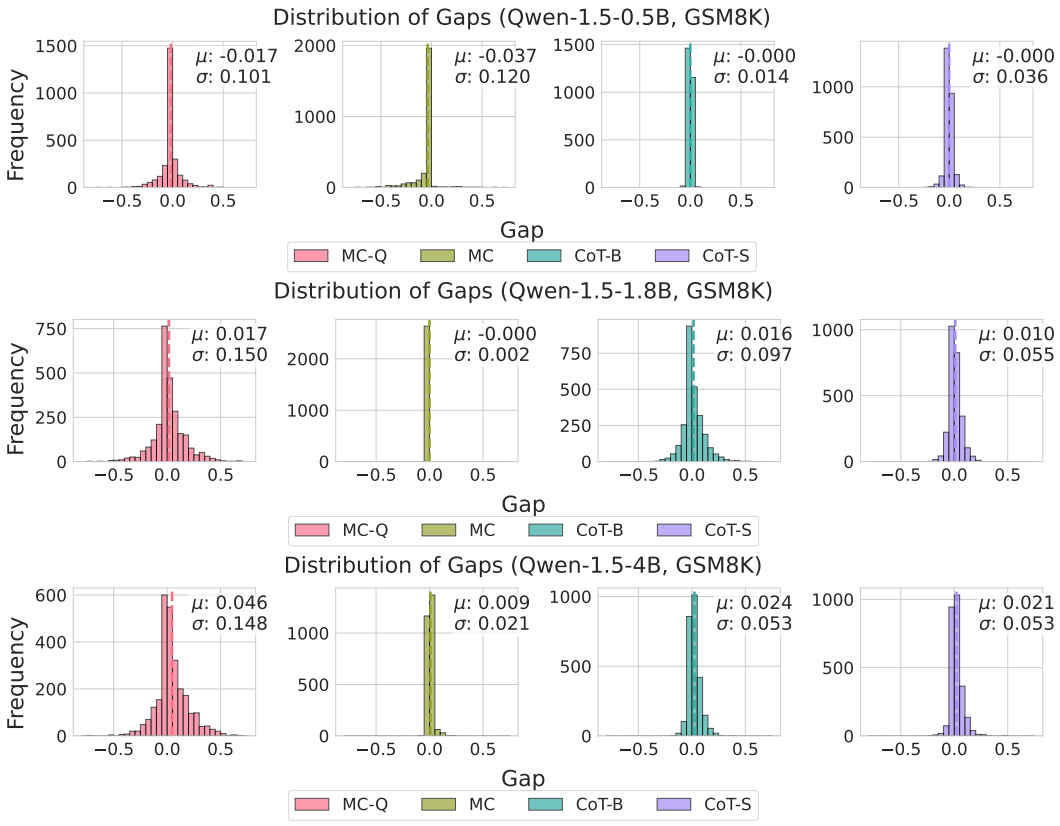
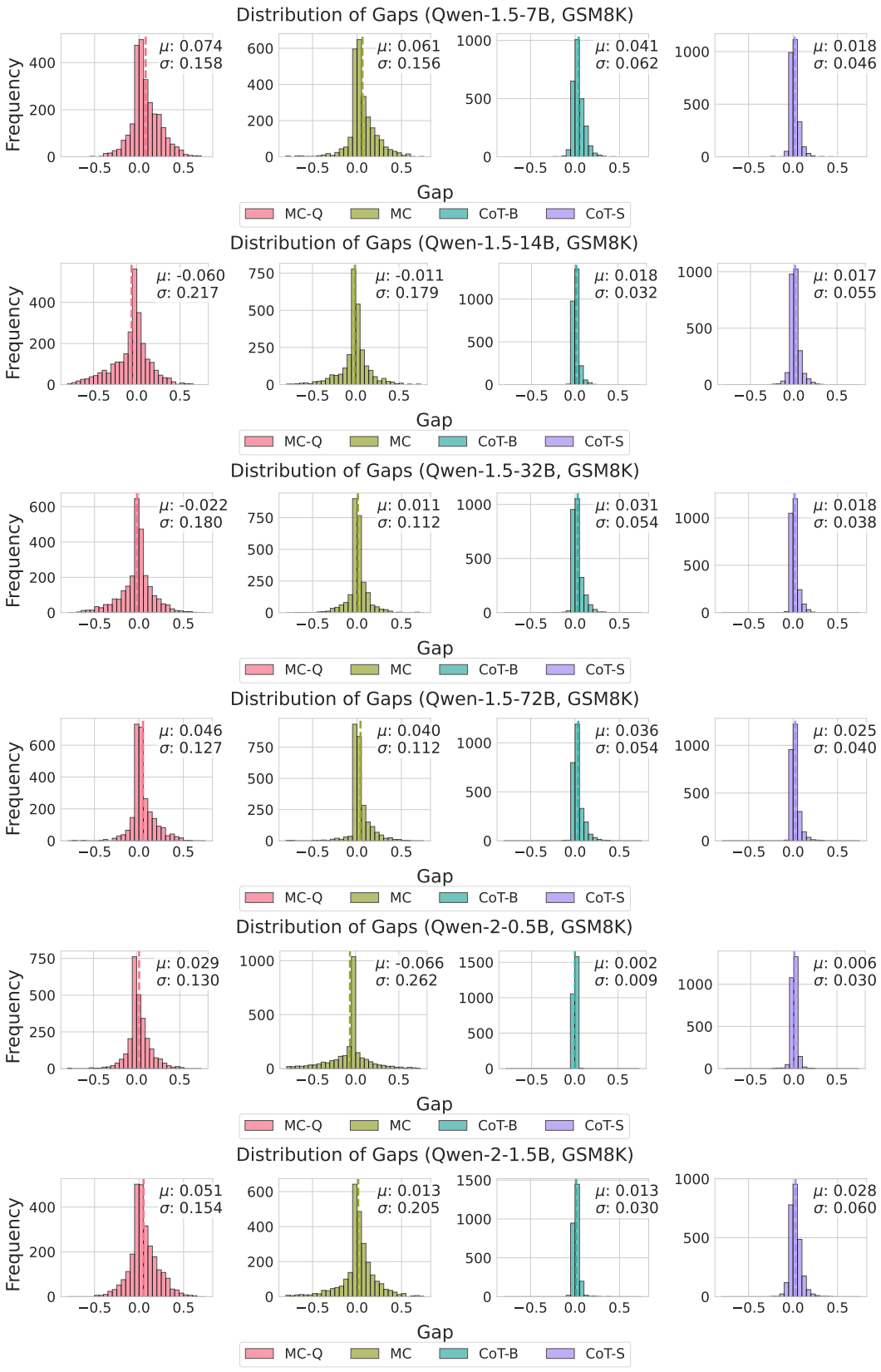
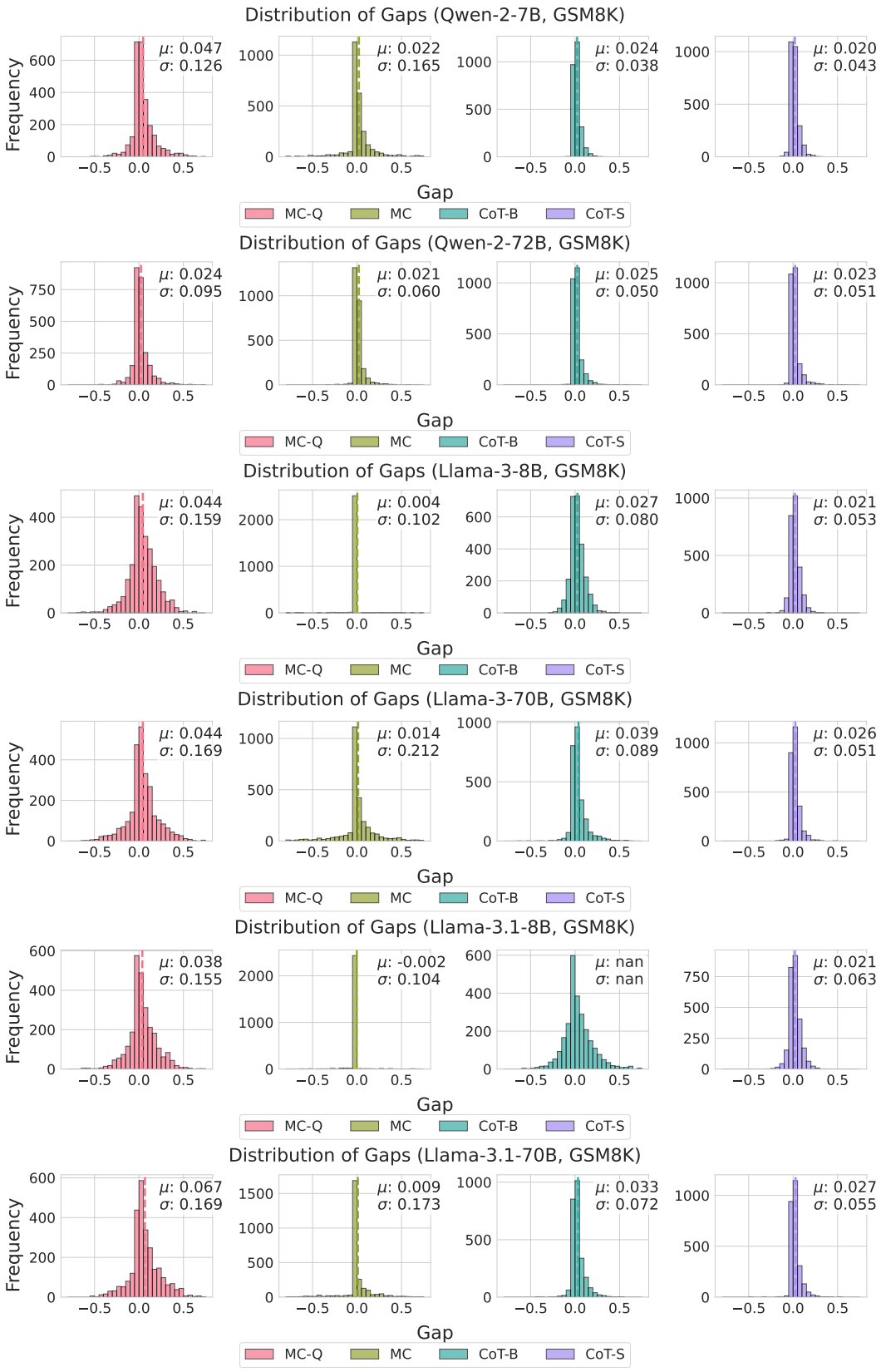


Figure 9: Tournament

D.5 Additional Results for Appendix A.2




Distribution of Gaps (Qwen-1.5-7B, GSM8K)
μ: 0.074 σ: 0.158
μ: 0.061 σ: 0.156
μ: 0.041 σ: 0.062
μ: 0.018 σ: 0.046
Distribution of Gaps (Qwen-1.5-14B, GSM8K)
μ: -0.060 σ: 0.217
μ: -0.011 σ: 0.179
μ: 0.018 σ: 0.032
μ: 0.017 σ: 0.055
Distribution of Gaps (Qwen-1.5-32B, GSM8K)
μ: -0.022 σ: 0.180
μ: 0.011 σ: 0.112
μ: 0.031 σ: 0.054
μ: 0.018 σ: 0.038
Distribution of Gaps (Qwen-1.5-72B, GSM8K)
μ: 0.046 σ: 0.127
μ: 0.040 σ: 0.112
μ: 0.036 σ: 0.054
μ: 0.025 σ: 0.040
Distribution of Gaps (Qwen-2-0.5B, GSM8K)
μ: 0.029 σ: 0.130
μ: -0.066 σ: 0.262
μ: 0.002 σ: 0.009
μ: 0.006 σ: 0.030
Distribution of Gaps (Qwen-2-1.5B, GSM8K)
μ: 0.051 σ: 0.154
μ: 0.013 σ: 0.205
μ: 0.013 σ: 0.030
μ: 0.028 σ: 0.060
Frequency
Gap
MC-Q
MC
CoT-B
CoT-S


Distribution of Gaps (Qwen-2-7B, GSM8K)
μ: 0.047
σ: 0.126
μ: 0.022
σ: 0.165
μ: 0.024
σ: 0.038
μ: 0.020
σ: 0.043
Frequency
Gap
MC-Q
MC
CoT-B
CoT-S
Distribution of Gaps (Qwen-2-72B, GSM8K)
μ: 0.024
σ: 0.095
μ: 0.021
σ: 0.060
μ: 0.025
σ: 0.050
μ: 0.023
σ: 0.051
Frequency
Gap
MC-Q
MC
CoT-B
CoT-S
Distribution of Gaps (Llama-3-8B, GSM8K)
μ: 0.044
σ: 0.159
μ: 0.004
σ: 0.102
μ: 0.027
σ: 0.080
μ: 0.021
σ: 0.053
Frequency
Gap
MC-Q
MC
CoT-B
CoT-S
Distribution of Gaps (Llama-3-70B, GSM8K)
μ: 0.044
σ: 0.169
μ: 0.014
σ: 0.212
μ: 0.039
σ: 0.089
μ: 0.026
σ: 0.051
Frequency
Gap
MC-Q
MC
CoT-B
CoT-S
Distribution of Gaps (Llama-3.1-8B, GSM8K)
μ: 0.038
σ: 0.155
μ: -0.002
σ: 0.104
μ: nan
σ: nan
μ: 0.021
σ: 0.063
Frequency
Gap
MC-Q
MC
CoT-B
CoT-S
Distribution of Gaps (Llama-3.1-70B, GSM8K)
μ: 0.067
σ: 0.169
μ: 0.009
σ: 0.173
μ: 0.033
σ: 0.072
μ: 0.027
σ: 0.055
Frequency
Gap
MC-Q
MC
CoT-B
CoT-S

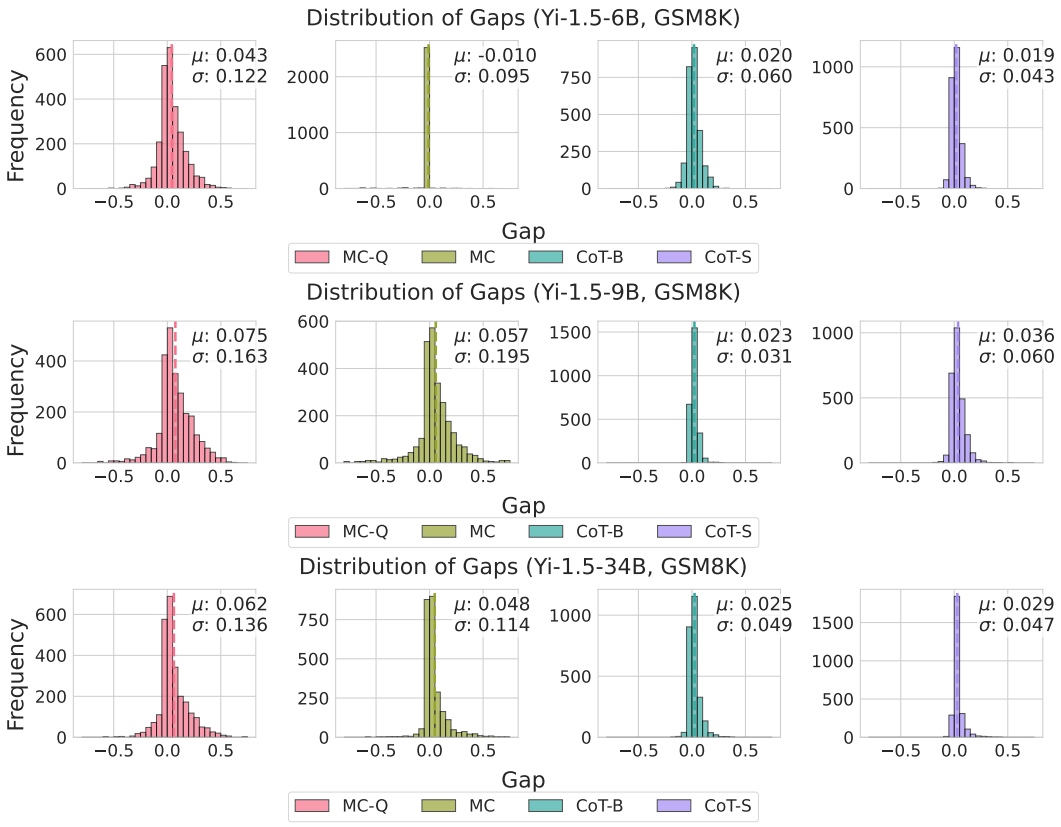
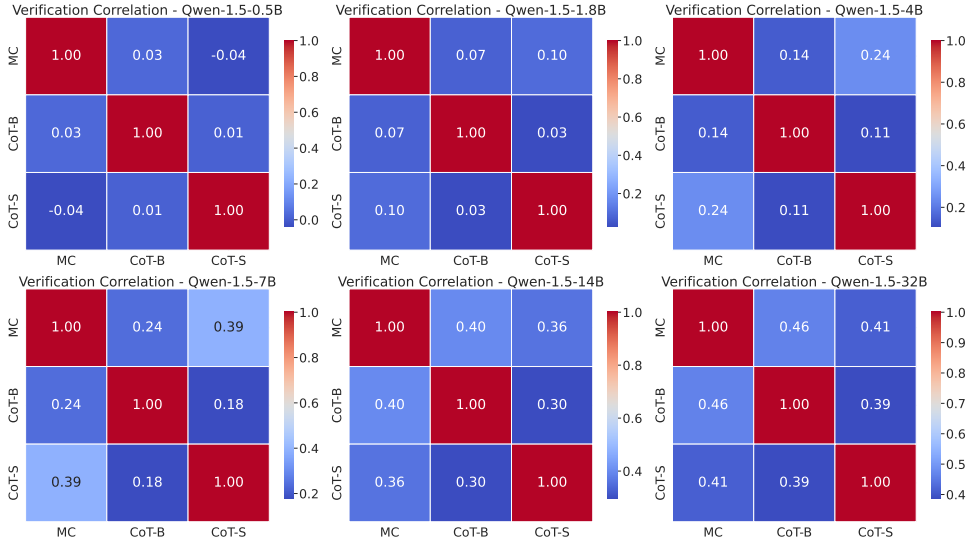


Figure 10: The empirical distribution of gaps of each verification method of each model on GSM8K. We cluster gaps in bins of intervals with width 0.005. We label the mean (μ) and standard deviation (σ) of each distribution.



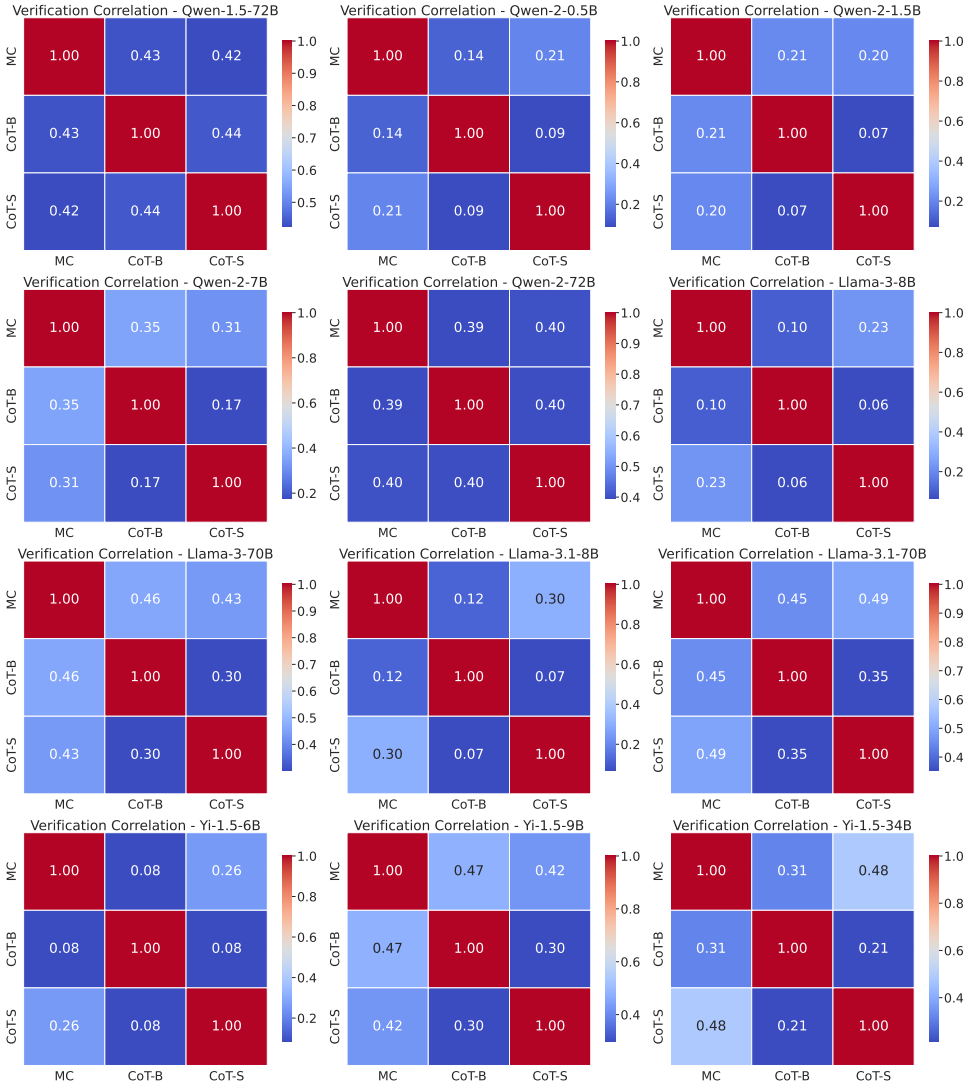
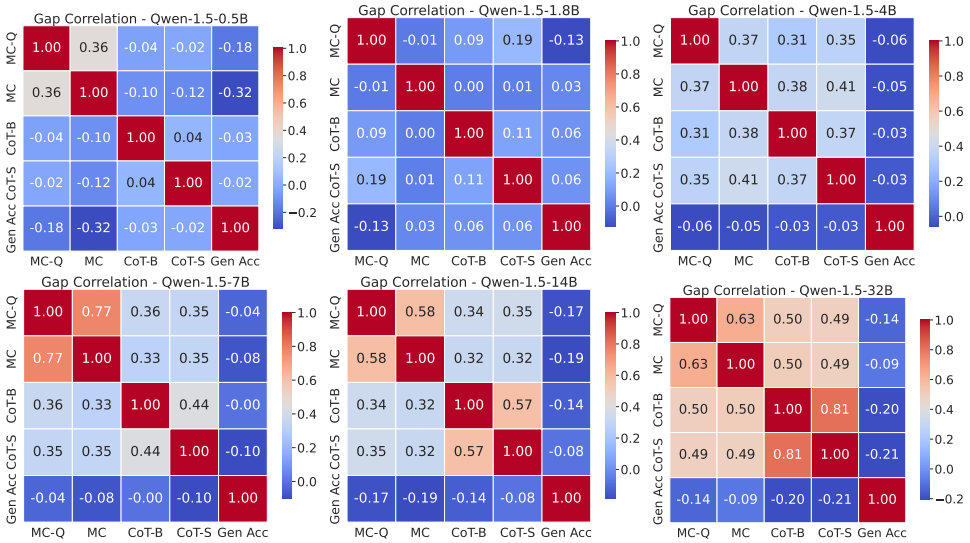


Figure 11: The correlation plot of the output of each verification $\widehat{u}$.



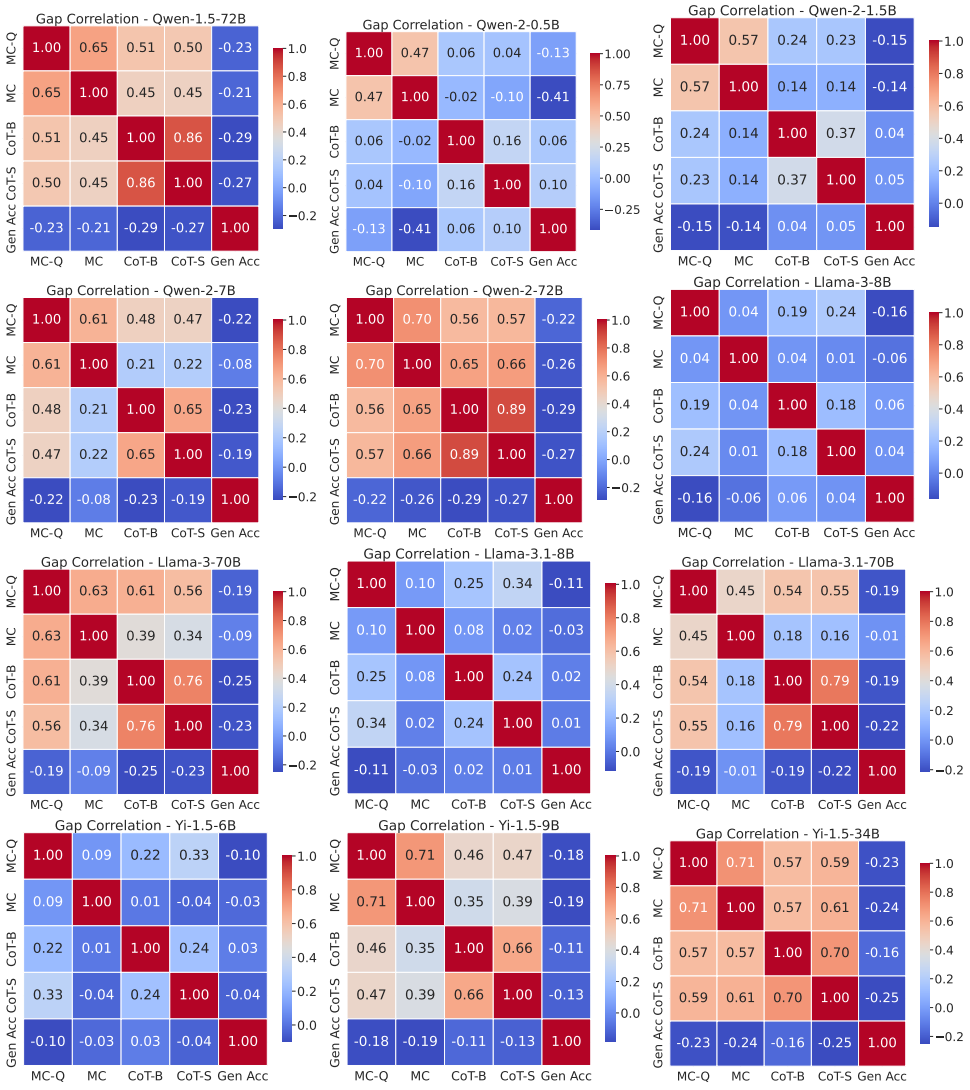


Figure 12: The correlation plot of the gap from each verification and generation accuracy.

D.6 Additional Results for Appendix A.3

Table 8: Relative gaps on GSM-8K for all models. For each verification, "top n" denotes taking the threshold as the n quantile of the proxity utility for each prompt, and $\tau = n$ denotes taking the threshold as n for all prompts. All numbers denote the percentage

Name	Size	MC	CoT-B	CoT-S	MC+CoT-B	MC+CoT-S	CoT-B+CoT-S	All
Qwen-1.5	0.5B	-1.62	-0.07	-0.01	-1.65	-1.36	-0.08	-1.42
	1.8B	2.36	1.04	0.95	2.16	2.22	1.18	2.14
	4B	4.55	2.26	2.14	4.81	4.77	3.50	4.90
	7B	7.46	3.98	1.84	9.02	7.88	4.88	9.12
	14B	-6.09	1.79	1.69	-5.73	-5.85	2.45	-5.60
	32B	-2.30	3.07	1.84	-1.87	-2.21	3.62	-2.00
	72B	4.61	3.43	2.47	5.70	5.25	4.46	6.07
Qwen-2	0.5B	3.01	0.18	0.64	3.01	3.13	0.72	3.11
	1.5B	5.04	0.86	2.78	5.08	7.68	3.15	7.61
	7B	4.68	2.31	1.97	5.25	4.96	3.40	5.46
	72B	2.44	2.50	2.28	3.21	3.13	3.35	3.65
Llama-2	7B	2.17	0.13	0.25	2.21	2.42	0.37	2.44
	13B	3.19	0.91	0.97	3.47	3.38	1.76	3.21
	70B	4.78	3.73	3.44	5.52	5.61	5.79	4.90
Llama-3	8B	4.98	2.59	2.10	5.88	5.81	4.09	5.83
	70B	4.37	3.88	2.59	4.91	4.71	4.45	4.98
Llama-3.1	8B	4.34	3.50	2.09	4.47	4.57	3.27	2.96
	70B	6.72	3.29	2.71	7.08	7.03	3.94	7.21
Yi-1.5	6B	4.27	2.01	1.88	4.83	4.72	3.16	4.95
	9B	7.50	2.32	3.61	7.79	7.87	4.80	8.11
	34B	6.23	2.49	2.86	6.32	6.35	3.76	6.41

E Hyperparameters

Table 9: Hyperparameter for Iterative Self-improvement

Minibatch size	64
Learning rate	1e-6
Optimizer	AdamW
Gradient step	2000
Max Sequence Length	2048
Data Type	bf16

F Generation and Verification Prompts

Multiple Choice Verification Prompt (GSM8K / nq_open)

Judge the correctness of the following solution of the problem. Answer with either Correct or Incorrect. Problem: {problem}
Solution: {generation}
Judge:

Chain of Thoughts Binary Prompt (GSM8K)

Review the following math problem and the attempted solution and verify the correctness of the attempted solution, with a judgement of <correct> or <incorrect>. Your judgement should follow each criterion below: - The final ANSWER is after the phrase "The answer is ANSWER", and verify if the answer is correct with respect to the problem. If there is no such phrase, treat the answer as incorrect. - Each solution contains a derivation before the final answer, check the soundness of the derivation as well. - Your final judgement should reflect solely on the correctness of the final answer, but if there are issues in the derivation, please mention them in your justification.
Problem: {problem}
Attempted Solution: {generation}
After examining the problem and the attempted solution: - Briefly justify your judgement, up to 50 words. - Conclude with the judgement using the format: "Correctness: <correct> or <incorrect>".
Remember to assess from the math verifier perspective and be critical and verify carefully.
Judgement:

Chain of Thoughts Score Prompt (GSM8K)

Review the following math problem and the attempted solution and give a score from 1 to 10 to the attempted solution. The final ANSWER is after the phrase "Final Answer: The final answer is ANSWER". Give the answer a 1 if there is no such phrase or ANSWER is wrong, and give the answer a 10 if both the answer and the derivation are correct.
Problem: {problem}
Attempted Solution: {generation}
After examining the problem and the attepmted solution: - Briefly justify your score, up to 50 words. - Conclude with the score using the format: "Score: <score>".
Remember to assess from the math verifier perspective and be critical and verify carefully.
Judgement:

Tournament Prompt (GSM8K)

Review the following math problem and two attempted solutions. Your task is to determine the better solution between the two. Your judgement should follow each criterion below: - The final ANSWER is after the phrase "The answer is ANSWER", and you should always prefer correct answers over incorrect answer. - Always prefer solutions with the phrase "The answer is ANSWER" over ones without it. - If both answers are correct or incorrect, you should prefer the one with better reasonings. Problem: {problem}
Solution A: {generation1}
Solution B: {generation2}
After examining the problem and the attempted solutions: - Briefly justify your judgement, up to 50 words. - Conclude with the judgement using the format: "Preferred solution: <A> or <B>".
Remember to assess from the math verifier perspective and be critical and verify carefully.
Judgement:

Sudoku Generation Prompt

You are a Sudoku solver specialized in 4x4 puzzles. You will be given a string of 16 digits representing an initial 4x4 Sudoku puzzle, where 0 represents an empty cell. Your task is to solve the puzzle and provide the complete solution.
Rules for solving a 4x4 Sudoku:
Each row must contain the numbers 1-4 without repetition.
Each column must contain the numbers 1-4 without repetition.
Each 2x2 quadrant must contain the numbers 1-4 without repetition.
The solution must maintain all the initial non-zero numbers from the puzzle in their original position.
Input format:
ab cd
ef gh
ij kl
mn op
Where each letter represents a single digit (0-4), and 0 indicates an empty cell. Your tasks:
Analyze the given puzzle.
Fill in all empty cells (0s) with appropriate numbers following Sudoku rules.
Provide your solution in the same format as the input, maintaining the original structure.
Puzzle: {problem}
Answer:

Sudoku Generation Example

Puzzle:
03 21
00 30
04 00
21 00
Answer: I filled A1 with 4 to complete Row 1. In Quadrant 1, the missing numbers 1 and 2 were assigned to B1 and B2 based on Column constraints. B4 was set to 4 to complete Row 2. In Quadrant 3, C1 was filled with 3. Row 3 missing numbers 1 and 2 were placed in C3 and C4. D4 was assigned 3 to complete Column 4, and D3 was set to 4 to finish Row 4.
Solution:
43 21
12 34
34 12
21 43

Sudoku Verification Prompt

You are a Sudoku validator for 4x4 puzzles. You will be given two strings of 16 numbers each: the first represents the initial Sudoku puzzle, and the second represents the proposed solution. Your task is to determine if the solution is correct.
Rules for a valid 4x4 Sudoku solution:
Each row must contain the numbers 1-4 without repetition.
Each column must contain the numbers 1-4 without repetition.
Each 2x2 quadrant must contain the numbers 1-4 without repetition.
The solution must maintain all the initial numbers from the puzzle.
Input format:
Question: Input format:
ab cd
ef gh
ij kl
mn op
Where each letter represents a single digit (0-4), and 0 indicates an empty cell. Answer: [same format but there should be no 0]
Your task:
Check if the solution maintains all initial numbers from the puzzle.
Verify that each row, column, and 2x2 quadrant contains 1-4 without repetition.
Respond with "Correct" if the solution is valid, or "Incorrect" if it's not.
Question: {problem}
Answer: {generation}
Judgment:

Sudoku Verification Example

problem:
03 00
21 00
00 01
10 43
generation:
34 12
21 31
34 21
12 44
correctness: Checking if the solution maintains all initial numbers from the puzzle:
Position (Row 1, Column 2): Puzzle has 3, Solution has 3 — Correct
Position (Row 2, Column 1): Puzzle has 2, Solution has 2 — Correct
Position (Row 2, Column 2): Puzzle has 1, Solution has 1 — Correct
Position (Row 3, Column 4): Puzzle has 1, Solution has 1 — Correct
Position (Row 4, Column 1): Puzzle has 1, Solution has 1 — Correct
Position (Row 4, Column 3): Puzzle has 4, Solution has 4 — Correct
Position (Row 4, Column 4): Puzzle has 3, Solution has 3 — Correct
All initial numbers are maintained.
Checking the rows:
Row 1: Numbers are 4, 3, 1, 2 (No repetition) - Correct
Row 2: Numbers are 2, 1, 3, 1 (Number 1 repeats) - Invalid
Row 3: Numbers are 3, 4, 2, 1 (No repetition) - Correct
Row 4: Numbers are 1, 2, 4, 3 (No repetition) - Correct
Checking the columns:
Column 1: Numbers are 4, 2, 3, 1 (No repetition) - Correct
Column 2: Numbers are 3, 1, 4, 2 (No repetition) - Correct
Column 3: Numbers are 1, 3, 2, 4 (No repetition) - Correct
Column 4: Numbers are 2, 1, 1, 3 (Number 1 repeats) - Invalid

Checking the 2x2 quadrants:
Quadrant 1 (Rows 1-2, Columns 1-2): Numbers are 4, 3, 2, 1 (No repetition) - Correct
Quadrant 2 (Rows 1-2, Columns 3-4): Numbers are 1, 2, 3, 1 (Number 1 repeats) - Invalid
Quadrant 3 (Rows 3-4, Columns 1-2): Numbers are 3, 4, 1, 2 (No repetition) - Correct
Quadrant 4 (Rows 3-4, Columns 3-4): Numbers are 2, 1, 4, 3 (No repetition) - Correct
Conclusion: There are repetitions in Row 2, Column 4, and Quadrant 2, the solution violates the Sudoku rules.
Therefore, the response is: Incorrect