Auto-Verify: A Plug-and-Play Step-wise Verification and Self-Correction Pipeline in Mathematical Reasoning

Jiayi Sheng [*1] **Jinrui Zhang** [*2] **Xuran Chen** [2] **Chenyang Deng** [2] **Bin Dong** [3] **Wotao Yin** [4]

Abstract

Recent advances in large language models (LLMs) have significantly enhanced their performance across various tasks. However, they continue to struggle with limited accuracy in mathematical reasoning due to hallucination, which includes intrinsic errors (e.g., calculation errors and unit inconsistencies) and extrinsic contradictions (e.g., logical contradictions and background inconsistencies) that frequently emerge during the reasoning process. To address hallucinations, most existing approaches focus on training reward models (ORM or PRM) and employing reinforcement learning. However, retraining such models is computationally intensive, time-consuming, and requires extensive annotations.

To overcome these challenges, we propose Auto-Verify, a plug-and-play, step-wise "verify-then-self-correct" pipeline that can be applied to any math reasoning model at inference time without additional training. This framework aims to fully harness the model's step-by-step self-refinement capability, guided by a pretrained verifier that evaluates and explains detected errors to identify and mitigate hallucination at each reasoning step. Once trained offline, the proposed pipeline enables accurate self-refinement to correct mistakes. This approach is broadly applicable, offering a practical solution to improve mathematical reasoning accuracy.

[*]Equal contribution [1]University of California, Berkeley [2]Alibaba DAMO Academy [3]Center for Machine Learning Research, Peking University [4]Decision Intelligence Lab, DAMO Academy, Alibaba Group US. Correspondence to: Jiayi Sheng <lorntz273@berkeley.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

1. Introduction

While Large Language Models (LLMs) demonstrate remarkable reasoning capabilities, they remain prone to hallucination errors (Huang et al., 2023), particularly in multi-step, complex, and symbolic reasoning tasks such as mathematical problem-solving. These hallucinations—instances where models generate plausible but incorrect or unfounded information—pose a significant challenge to the reliability of LLMs. Even minor errors in one step of a reasoning chain can propagate and significantly distort the final outcome.

Recent advancements, such as Monte Carlo Tree Search (MCTS) (Xie et al., 2024; Zhang et al., 2024a), step-level beam search (Chen et al., 2024b), and Chain of Thought (CoT) (Wei et al., 2022), have improved the overall reasoning performance of LLMs by encouraging more deliberate and structured thinking. However, these methods do not eliminate hallucinations, as they do not inherently guarantee the correctness of each step and cannot entirely prevent the model from generating incorrect or unfounded information.

The persistence of hallucinations undermines the reliability of LLMs, even as their reasoning capabilities improve. This issue is particularly critical in applications where accuracy is paramount, such as mathematical reasoning, scientific research, and decision-support systems. Without addressing hallucinations, the potential of LLMs to deliver trustworthy and consistent results remains limited. Therefore, developing specific methods to mitigate hallucinations across all models is crucial.

To mitigate hallucinations in mathematical reasoning, recent works have focused on training a numerical verifier (reward model) (Wang et al., 2024c; Yu et al., 2024) that outputs a numerical score indicating the probability of the content being verified as correct then fine-tuning the model with various strategies. The numerical verifier is essentially a reward model, which can be categorized into two types: Outcome reward model(ORM)(Cobbe et al., 2021; Yu et al., 2024) and Process reward model(PRM)(Wang et al., 2024a; Luo et al., 2024) trained by data labeled by other Language Models or a reliable methods which can evaluate each reasoning step reliably such as Monte Carlo Tree Search(MCTS)(Zhang et al., 2024a; Chen et al., 2024a). After training a precise

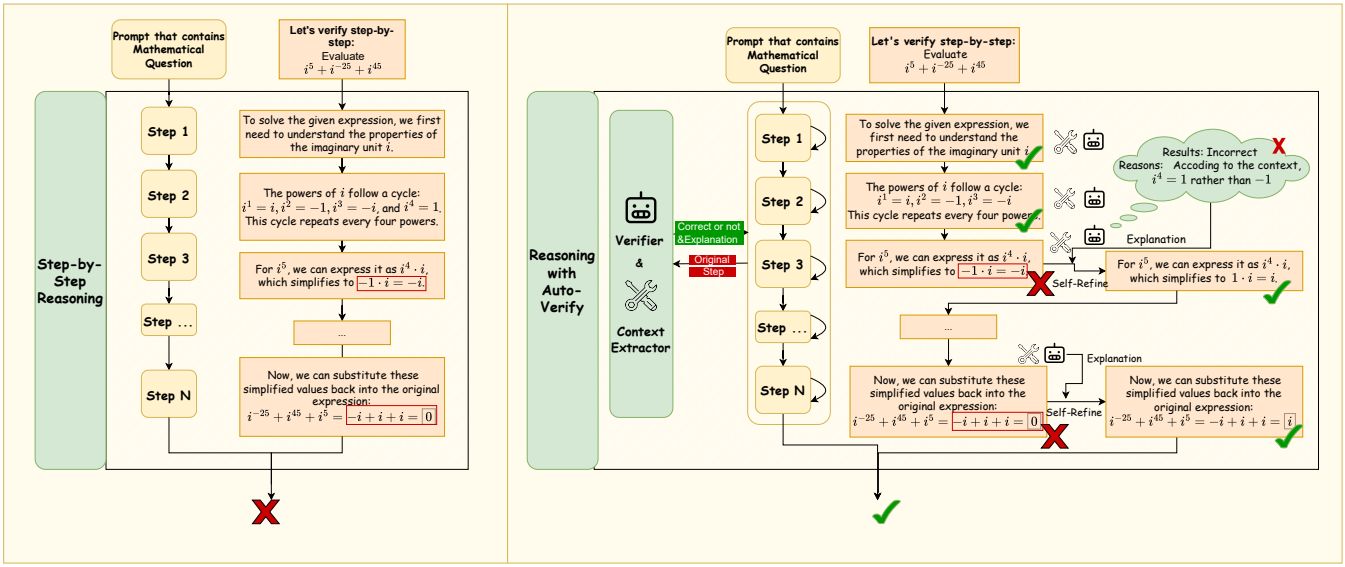

Figure 1. **Overview of Auto-Verify pipeline.** The flowchart on the left side of this diagram illustrates the step-by-step reasoning process by which a typical math reasoning model solves mathematical problems. The flowchart on the right side in a demonstration on how LLMs solve math problems step by step by Auto-Verify. A refine-and-correct iteration is performed at each step with the help of context extractor, ensuring that each step is free from any Hallucination.

model, reinforcement learning(RL) such as Proximal Policy Optimization (PPO)(Schulman et al., 2017) are applied to fine-tune the model with the pre-trained reward model. These methods can effectively improve the accuracy of math reasoning and avoid the hallucination to some extent. However, Hallucination is diverse and complex, encompassing many types, such as intrinsic errors (e.g., calculation errors and unit inconsistencies) and extrinsic contradictions (e.g., logical contradictions and background inconsistencies), which cannot be easily measured using simple numerical values, as relying solely on numbers loses much of the inherent characteristics of Hallucination. This is one of the major challenges faced by existing methods. Hence, it's essential to train a verifier that could give natural language feedback to help LLMs have a deeper understanding of their error and improve themselves.

As for refinement of LLMs based on feedback, recent research can be split into 3 groups: Training-time Correction(Dubois et al., 2024; Gulcehre et al., 2023), Generation-Time Correction(Hao et al., 2023; Miao et al., 2023) and Post-hoc Correction(Pan et al., 2023a; Chen et al., 2023). While Training-time Correction requires a large amount of labeled training data which requires extensive and labor-intensive manual labeling and Post-hoc Correction is time consuming as self-refine iteration happens at the end of the reasoning process but not each reasoning step, Generation-Time Correction is particularly well-suited for math reasoning tasks as it is a multi-step and time-consuming task. Recent research on Generation-Time Correction aims at self-thinking at each step and make a sensible decision that is possible to get to the answer. While this method allows the model to have a global perspective, enabling it to arrive at more accurate answers in subsequent reasoning, its self-correction is not sufficiently precise and detailed. Specifically, it does not focus on mitigating minor errors and hallucinations within individual steps.

Based on the aforementioned limitations of recent studies, we propose Auto-Verify, a plug-and-play step-wise verify then self-refinement pipeline that can be applied to any math reasoning model without additional training. This framework aims at fully unleashing model's step-by-step self-refine ability guided by a pretrained verifier which evaluates and explains any identified errors in natrual language to detect and mitigate fined hallucinations at each reasoning step. The pipeline of Auto-Verify is shown in Figure 1. Different from other methods, we are trying to mitigate all fined and complex hallucinations for all models with a step wise self-correction strategy based on a pretrained natrual language verifier.

As for training the verifier, in order to enable the LLMs to provide explanations for the corresponding errors, we firstly categorize Hallucination into two main types, then insert corresponding hallucinations into the correct reasoning steps and generate the explanation of why the processed step is

incorrect to build a training dataset. Since the hallucinations are inserted by the LLMs, the explanation will be precise and detailed. The verifier after trained offline, offers the advantage of enabling accurate, step-by-step error detection and explanation for any models without the need for additional joint training, making it computationally efficient and widely applicable in improving math reasoning accuracy.

Although LLMs have shown impressive ability in a variety of tasks, they are still suffering from generating response with long context(Wang et al., 2024b). Since the math reasoning process always contains multiple steps resulting in long inputs for the verifier, the limited context window constrains the performance of the verifier on detecting possible hallucinations. When we as a human are trying to judge whether one step is correct or not, we would not review the whole previous steps. Instead, we would filter the context which is highly related to the step and confirm whether the step can be derived by the related context. For example, imagine there is a question involving multiple independent equations to be solved. When we are verifying one step in solving one of the equation, it's unnecessary to review the steps in dealing with other equations obviously. The process filtering useful information from context is named **context extraction**. Inspired by various context compression and extraction techniques in Retrieval-augmented generation(RAG)(Zhang et al., 2024b; Gutiérrez et al., 2024; Cohen-Wang et al., 2024), we propose a step-wise math reasoning context extraction method to extract context elements that strongly contribute to the generated content. Instead of asking LLMs for extracting context directly, we use the likelihood(conditioned probability of generated text) to estimate how important the context is by removing a portion of the context and observing whether the likelihood changes significantly or not.

After acquiring precise detection and explanation of hallucination of each reasoning step, we ask the model to refine each reasoning step based on natural language feedback after generating it immediately, saving times and increasing reasoning accuracy at the same time. This self-correction is distinct from others who only refine based on a scaler feedback at each step, enabling detect more subtle mistakes during reasoning. In addition, this pipeline is the basis of other methods to mitigate flaws in reasoning, which means every state-of-the-art model or newly proposed methods can utilize this plug-and-play pipeline to further improve their performance.

Our contributions are summarized as follows:

- We proposed a plug-and-play step-wise verify then self-refinement pipeline that can be applied to any math reasoning model without additional training. This framework aims at fully unleashing model's step-by-step self-refine ability guided by a pretrained verifier which evaluates and explains any identified subtle errors to detect and mitigate hallucinations at each reasoning step for any model.
- We trained a verifier which is able to detect hallucination and provide explanation. By inserting finely categorized Hallucinations hallucination, a high-quality dataset containing problem-solving steps with Hallucinations and corresponding explanations is generated, enabling the verifier to achieve excellent performance.
- We propose a context extraction methods for mathematical reasoning, which shortens the context and filters out important context that contributes to the to be verified step, mitigating the performance loss in error detection caused by the context window limitations of the verifier.

2. Context Extraction

2.1. Preliminaries

When LLMs are solving math problems step by step, every step is derived from previous steps and the question which refers to the **context**. To formalize the math reasoning process, we denote $\mathcal{Q}$ as the to be solved math question, and $s_1, s_2, \ldots, s_n$ to be the steps, where n is the total number of steps required to solve the question. Then, define P_{LLM} to be an autoregressive model which defines a probability distribution over the next token given a sequence of preceding tokens. Hence, each step $s_k (i = k, \ldots, n)$ is sampled conditioned on the previous steps and questions:

$$s_k \sim P_{LLM}(\cdot|\mathcal{Q}, s_1, \ldots, s_{k-1})$$

When analyzing from a sentence perspective, assume that $s_{(i,j)},\ j = 1, 2, \ldots, n_i$ is the j^{th} sentence of step i, then every sentence is generated based on previous sentences:

$$s_{(k,j)} \sim P_{LLM}(\cdot|\mathcal{Q}, s_{(k,j-1)}, \ldots, s_{(k,1)}, s_{(k-1,n_{k-1})}, \ldots, s_{(1,1)})$$

Context Attribution is to find out which sentence of the context contribute to the output of the model and assigning them a numerical value that indicates how important they are to the final output. Context Extraction is to filter out all sentenses whose score is over a given threshold based on the Context Attribution results.

2.2. Method

Inspired by ContextCite(Cohen-Wang et al., 2024) which introduces context attribution in retrieval augmented generation(RAG), the contribution can be quantified by the contribution of each sentences to the probability of generated output. Even though step by step math reasoning generate each step autonomously based on the context which is

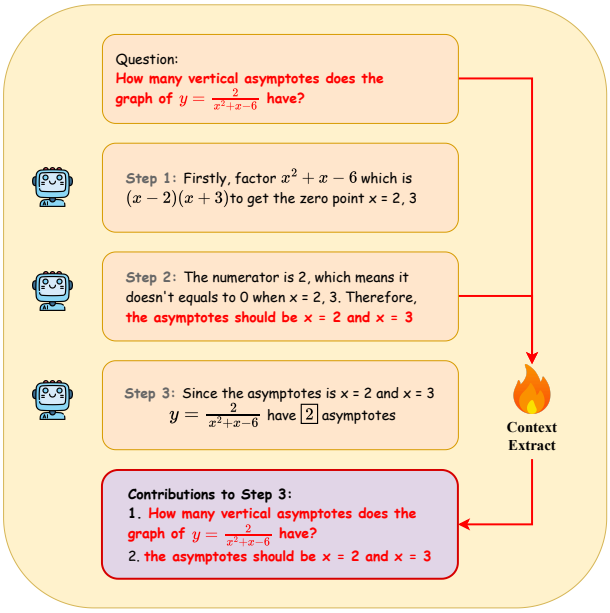

Figure 2. **An example of context extraction**. The sentences of the question and previous steps which contribute to the generation of step 3 are extracted.

different from RAG which requires queries to generate outputs, measuring context contribution through the probability of generated output is still applicable in math reasoining settings. In order to better describe the probability and the relationship between different sentences, we first encode all sentences before the j^{th} sentence of step i as a d-dimensional vector denoted as **sentences vector** V, where d is the total number of all sentences before the j^{th} sentence of step i. Each element of the vector corresponds to a sentence, with a value of 1 if the sentence appears and 0 if it does not. The reason why we define the vector in this form is that when analysing the contribution of each sentences, the content of the sentences is trivial, but whether the appearance of each sentence would have great impact on the probability of the output is important.

we define the probability as a function of the sentences vector, denoted as $f : \{0,1\}^d \to \mathbb{R}$:

$$f(V) := P_{LLM}(\cdot|\mathcal{Q}, \textit{mask}(V, S_{(i,j)}))$$

where $S_{(i,j)} = \{s_{(k,j-1)}, \ldots, s_{(k,1)}, s_{(k-1,n_{k-1})}, \ldots, s_{(1,1)}\}$, V is the sentences vector of $S_{(i,j)}$, and *mask* is a function that output sentences where their corresponding element of V is 1.

To find the contribution of each sentence, the best way is to find a linear surrogate function of f by linear regression which could demonstrate contribution of each sentences by the weight. Based on Contextcite(Cohen-Wang et al., 2024) , using logit function σ^{-1} ($\sigma^{-1}(p) = log\frac{p}{1-p}$) to map $[0,1]$ to $(-\infty, +\infty)$ could have excellent regression result. Therefore, our job is to randomly sample vector v from $\{0,1\}^d$ and calculate $f(V)$ to have a set of training dataV_i, $f(V_i)$, then execute regression to obtain weight. The regression task could be written in the following form:

$$\sigma^{-1}(\hat{f}(V)) = W^T \cdot V + b, \ where \ \ \sigma^{-1}(\hat{f}(V_i)) = W^T \cdot V_i + b$$

where $W \in \mathbb{R}^d, b \in \mathbb{R}$ are the weight vector and bias respectively. Then, W is the attribution score refering to the contribution to the output for each sentences.

We summarize the context attribution process in Algorithm 1

Algorithm 1 Context Attribution for k^{th} step

Input: question $\mathcal{Q}$, Regulation coefficient λ, number of regression training data m, number of sentences d, sentences for each previous steps $\{\{s_{(i,j)}\}_{j=1}^{n_i}\}_{i=1}^{k-1}$
Output: attribution score for each sentences $\omega_{i,j}$
{Generate step k: s_k}
$s_k \sim P_{LLM}(\cdot|\mathcal{Q}, s_{(k-1,n_{k-1})}, \ldots, s_{(1,1)})$

$f(V) := P_{LLM}(\cdot|\mathcal{Q}, \textit{mask}(V, S_{(i,j)}))$
$g(V) := \sigma^{-1}(P_{LLM}(\cdot|\mathcal{Q}, \textit{mask}(V, S_{(i,j)})))$

{Collect LASSO training data}
for $l = 1$ **to** m **do**
  Random sample $V_l \in \mathbb{R}^d$ from $\{0,1\}^d$
  $y_l \leftarrow g(V_l)$
end for
$\omega_{i,j}, b \leftarrow LASSO(\{y_l, V_l\}_{l=1}^m, \lambda)$
return $\omega_{i,j}$

3. Verifier

Based on Section 1, we aims at training a verifier which can detect hallcination and explain it precisely. To achieve the goal, we will catagorize hallucination first and generate synthetic dataset later, which are explained in details as follows.

3.1. Hallucination Classification

Our key observation is that hallucinations arising in step-by-step mathematical solutions generated by large language models (LLMs) typically fall into two categories. The first category, termed **intrinsic errors**, encompasses inaccuracies that are self-contained within a given reasoning step. For example, numerical miscalculations (e.g., "30+20 = 60'') or the application of incorrect mathematical formulas (e.g., stating the area of a square as $4 \times a$' instead of a^2. The

second category, designated as **extrinsic contradictions**, occurs where the reasoning presented in a particular step is inconsistent with information established in preceding steps or the problem statement itself. This may involve the erroneous modification of previously derived intermediate results or the unsubstantiated rejection of prior conclusions.

3.2. Synthetic Dataset Generation

We present an automated methodology for generating the two identified types of hallucinations as shown in Figure 3. Our approach employs verification rationales, each consisting of positive and negative examples. These examples contain the original math problem, the current reasoning step under consideration, a compressed context for efficient training, and, importantly, an explanation detailing why the current step is either correct or incorrect. This explanatory component serves a dual purpose: enabling the model to automatically assess step correctness and providing feedback to solution providers for self-refine process. Given the limited availability of richly annotated step-level datasets (Golovneva et al., 2023) and the resource-intensive nature of manual stepwise data acquisition, we overcome this challenge by generating synthetic data. Positive instances are sourced from the correct chain-of-thought (CoT) reasoning steps within the GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) datasets. Negative instances are synthetically created by injecting hallucinations into these steps using the Llama3.3-70B model (Dubey et al., 2024), ensuring an equal representation of positive and negative examples. The synthetic data generation process is structured into four distinct phases as below.

Step 1: Collecting chain-of-thought reasoning steps The existing solutions in the MATH (Hendrycks et al., 2021) and GSM8k (Cobbe et al., 2021) datasets, while serving as valuable resources, often present a challenge in terms of granularity. Specifically, the level of detail provided does not typically extend to an explicit, step-by-step exposition of the underlying mathematical reasoning. These solutions lack fine-grained articulation, which is necessary for a comprehensive understanding of the problem-solving process. To address this, we leverage the Llama3.3-70B model (Dubey et al., 2024) to capture and collect the implicit reasoning within these solutions. This produced the golden chain-of-thought solutions, characterized by their detailed, step-level breakdown of the mathematical derivations. We chose Llama3.3-70B due to its long context length, which enables handling complex, multi-step reasoning tasks. Additionally, it offers a strong balance of cost efficiency, high accurancy and relatively fast processing speed.

Step 2: Select location to insert a hallucination Our observation of hallucination types specifies unique requirements for each category, and not every chain-of-thought reasoning step is suitable to insert every type of hallucination. For instance, if a math reasoning step solely includes numerical computation, it can be difficult to introduce extrinsic contradictions. Therefore, we define custom rules for the Llama3.3-70B model(Dubey et al., 2024), instructing it to determine whether a given step contains the necessary elements for injecting a certain hallucination. For example, the model will judge whether a particular step involves referring content from previous steps. The details of judging rules are in appendix.

Step 3: Generate next reasoning step with hallucinations To regulate the distribution of hallucinations across individual reasoning steps, we implement a sampling method to select one reasoning step per solution as the candidate for hallucination injection. Based on the results of Step 2, which determined the suitability of each step for specific hallucination types, we employ a few-shot learning technique. For each hallucination type, we provided three demonstrations, each consisting of a hallucinated example, an explanation of how it is injected, and associated constraints. The model was subsequently provided with the math problem and the preceding reasoning steps. We extract the generated next step with target hallucination from the model's output.

Step 4: Extract context and explanation To mitigate the context windows limitation of the verifier, we prompt Llama3.3-70b(Dubey et al., 2024) to summarize the sentences that contribute to the generated step of each data. After summerizing context, each data in training dataset is shorter, enabling verifier to detect hallucination more precisely. Moreover, we asked Llama3.3-70b(Dubey et al., 2024) to give explanations about why this step is incorrect or correct, which form the most significant part of our dataset.

4. Step-wise Self-Refine

Given the question and previous steps, Step-wise Self-Refine generates the next step, provides feedback on the generated step, and refines the step according to the verifier feedback. Step-wise Self-Refine iterates between feedback and refinement until a desired condition is met. SELF-REFINE relies on a suitable language model, and three prompts (for initial generation, feedback, and refinement prompts). The whole Self-Refine can be divided into 4 parts:

4.1. Steps Generation

Given an math question Q, previous steps$\{S_i\}_{i=1}^{k-1}$, prompt p_{gen}, and math reasoning model f_{LLM}, the reasoning model generates the k^{th} step S_k:

$$S_k \sim f_{LLM}(\cdot|p_{\text{gen}}, S_{k-1}, \ldots, S_1, Q)$$

Here, the p_{gen} is a task-specific prompt for next step genera-

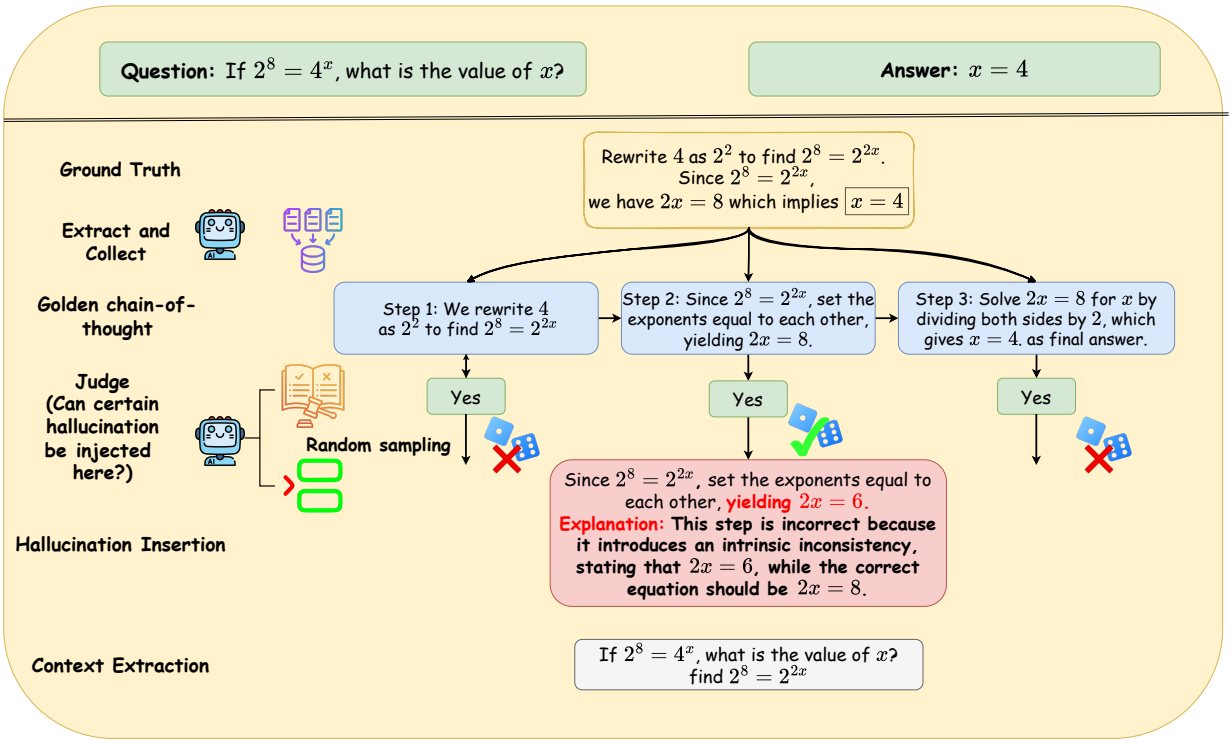

Figure 3. **Demonstration of Synthetic Dataset Generation.**

tion.

4.2. Feedback (using a Separate Feedback Model)

Next, a verifier $f_{verifier}$, which has been supervised fine-tuned (SFT), to provide feedback(whether the problem is correct or not and its explanation) E on the generated step, given a task-specific verification prompt p_{fb}, previous steps $\{S_i\}_{i=1}^{k-1}$ and the question Q for generating feedback:

$$E \sim f_{verifier}(\cdot|p_{\text{fb}}, S_{k-1}, \ldots, S_1, Q)$$

Where $S_k \subset p_{\text{fb}}$, which means the k^{th} step is a part of the verification prompt. Intuitively, the feedback may address multiple aspects of the output. To be specifically, a set of hallucination like logic error or inconsistent data can be detected.

4.3. Refine (Using Feedback)

Next, SELF-REFINE uses feedback E to refine the generated step:

$$S_k \sim f_{LLM}(\cdot|p_{\text{refine}}, S_{k-1}, \ldots, S_1, Q)$$

where $E, S_k \subset p_{\text{refine}}$, meaning p_{refine} contains information of E, S_k. Given the generated step and the generated feedback, the model regenerates the step to correct the mistakes guided by the feedback given by the feedback.

4.4. Iterating SELF-REFINE

Step-wise Self-Refine alternates between feedback and refining steps until a stopping condition is met. The stopping condition either stops at a specified timestep t or when the refined step is verified to be correct.

To inform the model about the previous iterations, we retain the history of previous feedback and outputs by appending them to the prompt. Intuitively, this allows the model to learn from past mistakes and avoid repeating them. The j^{th} iteration for k^{th} step S_k^j is generated by:

$$S_k^j \sim f_{LLM}(\cdot|p_{\text{refine}}^{(j)}, S_{k-1}, \ldots, S_1, Q)$$

Where $p_{\text{refine}}^{(j)}$ is the self-refine prompt containing the previous iteratively generated $j-1$ steps of step k along with the corresponding feedback provided by the verifier.

5. Auto-Verify

Auto-Verify is a plug-and-play, step-wise "verify-then-self-correct" pipeline as shown in Figure 1 that can be applied to any math reasoning model without additional training.

Table 1. **Verifier Hallucination Detection Accuracy on Benchmarks**. The Benchmark GSM8K and MATH contain only incorrect problems, and the data in these columns reflect the model's accuracy in identifying incorrect answers. The Correct GSM8K and MATH contain only correct problem-solving steps, and the data in these columns represent the model's accuracy in correctly recognizing correct answers.

Model	Size	Zeroshot	Benchmark GSM8K	Correct GSM8K	Benchmark MATH	Correct MATH
GPT-4o	-	√	90.75	90.73	83.12	84.11
GPT-4o-mini	-	√	91.5	81.75	81.8	77.42
GPT-3.5-Turbo	-	√	74.22	73.52	56.18	73.10
Gemini-1.5-Flash	-	√	91.51	91.58	78.58	81.74
Gemini-1.5-Pro	-	√	92.13	94.96	62	85.67
Claude-3.5-sonnet	-	√	84.68	93.85	80.78	90.6
Claude-3.5-haiku	-	√	84.83	93.25	75.3	89.36
Llama-3.1	8b	√	81.12	61.52	60.88	77.66
Llama-3.1	70b	√	88.20	96.96	74.2	91.68
Qwen-2.5	7b	√	72.14	73.90	37.6	76.72
Gemma-2	9b	√	59.42	93.25	27.16	74.96
Gemma-2	9b	×	62.21	93.45	42.48	82.78
Auto-Verifier	9b	√	**70.12**	**94.31**	**66.92**	**75.13**

Table 2. **Verifier Dropout on benchmarks**.

Model	Size	Zeroshot	Benchmark GSM8K	Correct GSM8K	Benchmark MATH	Correct MATH
GPT-4o	-	√	2.27	0.2	3.38	0.1
GPT-4o-mini	-	√	2.27	0.3	3.32	0.16
GPT-3.5-Turbo	-	√	2.35	0.3	3.08	0.08
Gemini-1.5-Flash	-	√	2.12	0.98	5.96	2.45
Gemini-1.5-Pro	-	√	2.34	2.73	3.5	1.12
Claude-3.5-sonnet	-	√	2.65	2.7	0.44	1.24
Claude-3.5-haiku	-	√	4.01	1.4	0.86	0.26
Llama-3.1	8b	√	1.1	5.19	4.14	3.45
Llama-3.1	70b	√	1.01	1.01	1.77	0.78
Qwen-2.5	7b	√	15.61	10.65	35.79	18.9
Gemma-2	9b	√	23	3.08	48	20.19
Gemma-2	9b	×	13.1	1.07	16.11	13.4
Auto-Verifier	9b	√	**0.01**	**0.08**	**0.48**	**0.03**

Firstly, we let LLMs to think step by step. By setting stopping criteria, after each step, the model would stop generating to wait for verification. Then, a context extactor starts to extract the sentences that contribute to the generation of the generated step. After the context is summarized by the context extractor, the summarized context and the generated step will be delivered to the verifier who will provide the results of whether this step contains hallucination or not and its reasons. If the step is wrong, the reasons, summarized context and generated step will guide the reasonoing model to regenerate this step. After regeneration, the same verify-then-refine process will continue until the step is verified to be correct or the number of iterations has reached the threshold. Finally, the step is correct, and the reasoning process can proceed. The formal expression of Auto-Verify is shown in Algorithm 2.

6. Experiments

In this study, we aim to evaluate the performance of our verifier and the improvement in reasoning ability brought by Auto-Verify across different models.

6.1. Verifier

Training Data We build our training data from the **train** splits of both GSM8K and MATH. Each problem in these datasets already has a multi-step solution, but the original solution steps often need to be reorganized or expanded to reveal the complete chain-of-thought. After collecting the golden chain-of-thought from these datasets, we construct our training dataset into two distinct parts: **correct reasoning steps** and **incorrect reasoning steps**.

First, We gather 7.4k correct steps in GSM8K, preserving the original flow of valid reasoning from the golden chain-of-thought.Each step is paired with (a) the concise context

Table 3. **Improvements of models with Auto-Verify on benchmarks**. This table presents the proportion of originally incorrect problems that were corrected after applying Auto-Verify across different models. Additionally, the table includes results obtained without using the context extraction technique for comparison.

Model	Context Extraction	MATH500	GSM8K	Olympiadbench	Aime24
Llama-3-8b	×	↑ 15.38%	↑ 23.3%	↑ 1.5%	↑ 1.1%
Llama-3-8b (Ours)	√	↑ **15.38%**	↑ **24.4%**	↑ **1.5%**	**0%**
Llama-3.1-8b	×	↑ 14.0%	↑ 33.3%	↑ 3.2%	↑ 1.2%
Llama-3.1-8b (Ours)	√	↑ **18.6%**	↑ **38.1%**	↑ **6.0%**	↑ **1.2%**
Llama-3.1-70b	×	↑ 32.26%	-	-	-
Llama-3.1-70b (Ours)	√	↑ **34.47%**	-	-	-
Qwen-2.5-14b	×	↑ 18.5%	↑ 30.7%	↑ 3.0%	↑ 1.2%
Qwen-2.5-14b (Ours)	√	↑ **24%**	↑ **38.4%**	↑ **3.0%**	↑ **1.2%**
Gemma-2-9b	×	↑ 1.1%	↑ 3.24%	0%	0%
Gemma-2-9b (Ours)	√	↑ **3.5%**	↑ **2.7%**	**0%**	↑ **1.1%**

that gives just enough prior information for understanding that step and (b) a natural-language explanation of why it is valid. Similarly, we obtain 7.5k correct steps in MATH, each annotated with context and an explanation of correctness. These correct steps directly reflect the gold solutions in the datasets, ensuring that our verifier sees diverse, well-grounded reasoning examples during training.

Second, We synthesize erroneous steps by injecting hallucinations(3.2) into otherwise correct solution chains. Specifically, for each problem's chain-of-thought in MATH and GSM8K, we select a step where an error can be plausibly introduced. Such step is replaced with an altered version, one that contains either *Intrinsic errors* or *Extrinsic contradictions*. Each incorrect step is likewise accompanied by a concise context—filtered from earlier parts of the solution chain and a natural-language explanation clarifying why the step is wrong. These fine-grained annotations help the verifier not only detect errors but also provide feedback for self-refine process.

Test Dataset (Benchmarks) We construct test data using exactly the same procedure described above but starting from the **test** splits of GSM8K and MATH. This yields four categories (correct vs. incorrect steps, sourced from GSM8K vs. MATH), providing a comprehensive benchmark for evaluating step-level verification and explanation quality. Throughout our experiments, we measure:

1. **Verification Accuracy**: the fraction of steps where the verifier correctly classifies a step as "correct" or "incorrect."

2. **Dropout Rate**: the fraction of steps where the verifier either refuses or fails to provide a judgment.

Baselines Our Auto-Verifier is fine-tuned on **Gemma-2-9b**. We compare our approach with strong proprietary and open source models, including OpenAI's GPT variants, Gemini, Claude and Llama, Gemma, Qwen. By default, we report the results obtained based on the same zero-shot prompting, detailed in Appendix.

Results Despite having fewer parameters than many of the large-scale LLMs, our fine-tuned Gemma-2-9B verifier surpasses or closely matches larger proprietary models on both the GSM8K and MATH test sets. In particular, we outperform GPT4o on Benchmark of GSM8K, and GPT3.5 on MATH Benchmark. In addition, our Auto-Verifier demonstrated a notably low dropout rate across both experiments.

6.2. Auto-Verify Experiments

Benchmarks We aim to evaluate the performance of our Auto-Verify by measuring the accuracy of solving math problems among a variety of benchmarks. To achieve this goal, we select both in-domain and out-of-domain test sets, specifically MATH, GSM8K for in-domain benchmark and Olympiadbench(He et al., 2024), Aime24(MAA, 2024) for out-domain benchmarks.

Experimental Setup To better demonstrate the correction capability of our Auto-Verify, we first perform mathematical reasoning using various models such as Llama-3, Llama-3.1, Qwen-2.5, Gemma-2. Then, we conduct experiments with Auto-Verify to evaluate the proportion of originally incorrect problems that it can correctly resolve as shown in Table 3. We set the Context extraction training data size as 32.

Ablation Study To further investigate the role of context extraction, we conducted an ablation study. Based on the application of Auto-Verify mentioned in the previous section, we tested its performance after removing the context summary. The detailed results are shown in Table 3.

Results After applying Auto-Verify, the accuracy improvement on incorrect problems in the in-domain datasets, MATH500 and GSM8K, is highly significant. More-

over, it is evident that utilizing the context extraction method further enhances the model's accuracy on these datasets.However, on the out-of-domain datasets (OlympiadBench and AIME24), the model's accuracy still requires further improvement. These datasets contain inherently more challenging problems, highlighting the need for future work to focus on enhancing performance on out-of-domain problems.

6.3. Computational Time Cost

To evaluate the impact of Auto-Verify on reasoning time, we use GSM8K as the dataset. We conduct experiments with three different settings: The original model without Auto-Verify (**Classic**), Auto-Verify with at most one self-refine for each step (**Auto-Verify(2)**), Auto-Verify with at most three self-refine for each step(**Auto-Verify(3)**). Then, we measure the average time required to complete all problems in GSM8K for each setting. The results are presented in Table 4.

Table 4. **The average execution time for each problem on GSM8K**

Model	Classic	Auto-Verify (1)	Auto-Verify (3)
Llama-3-8b	9.6s	55.2s	58.8s
Llama-3.1-8b	9.6s	80s	102s
Qwen-2.5-7b	3.6s	11s	12s
Gemma-2-9b	8.2s	36s	40.2s

7. Related Work

7.1. Mathematical Searching and Decision Method

In Mathematical reasoning, even though a variety of searching and decision making methods such as Monte Carlo Tree Search(MCTS) (Xie et al., 2024; Zhang et al., 2024a), step level beam search(Chen et al., 2024b), Chain of Thought(COT)(Wei et al., 2022) have been proposed to enhance model's reasoning abilities by thinking or searching before generating the next step during reasoning process, they are still facing the hallucination that makes decisions and searching strategy unreliable resulting in wrong answer.

7.2. Outcome and Process Verifier

To mitigate the Hallucinations in Mathematical reasoning, recent works have focused on training a numerical verifier(reward model)(Wang et al., 2024c; Yu et al., 2024) that outputs a numerical score indicating the probability of the content being verified as correct then fine-tuning the model with various strategy .The numerical verifier is essentially a reward model, which can be categorized into two types: Outcome reward model(ORM)(Cobbe et al., 2021; Yu et al., 2024) and Process reward model(PRM)(Wang et al., 2024a; Luo et al., 2024) trained by data labeled by other Language Models or a reliable methods which can evaluate each reasoning step reliably such as Monte Carlo Tree Search(MCTS)(Zhang et al., 2024a; Chen et al., 2024a). After training a precise model, reinforcement learning(RL) such as Proximal Policy Optimization (PPO)(Schulman et al., 2017) are applied to fine-tune the model with the pre-trained reward model. These methods can effectively improve the accuracy of math reasoning and avoid the hallucination to some extent. However, Hallucination is diverse and complex, which cannot be easily measured using simple numerical values. In contrast, we train a verifier that could give natural language feedback to help LLMs have a deeper understanding of their error and improve themselves.

7.3. LLM Output Refinement

As for refinement of LLMs based on feedback, recent research can be split into 3 groups: Training-time Correction(Dubois et al., 2024; Gulcehre et al., 2023), Generation-Time Correction(Hao et al., 2023; Miao et al., 2023) and Post-hoc Correction(Pan et al., 2023a; Chen et al., 2023). While Recent research on Generation-Time Correction aims at self-thinking at each step and make a sensible decision that is possible to get to the answer.

7.4. Context Attribution

In mathematical reasoning and structured problem-solving, step-wise context attribution has been explored to refine context extraction dynamically. Least-to-Most Prompting (Zhou et al., 2022) and Self-Consistency Decoding (Wang et al., 2022) focus on breaking down complex reasoning into intermediate steps, selecting relevant context at each stage. However, these approaches often lack fine-grained error correction, making them less effective in mitigating hallucinations at the step level.

8. Conclusions

In this paper, we introduced Auto-Verify, a plug-and-play, step-wise verification and self-correction pipeline designed to enhance mathematical reasoning in large language models (LLMs). Our approach addresses the prevalent issue of hallucinations in multi-step reasoning by integrating a pretrained verifier that provides natural language feedback, enabling precise error detection and correction at each step. Unlike traditional reinforcement learning-based methods that require extensive training and computational resources, Auto-Verify operates efficiently without additional model retraining, making it widely applicable to various LLMs.

Through extensive experiments, we demonstrated that Auto-Verify significantly improves reasoning accuracy across mul-

tiple mathematical benchmarks, outperforming existing verification approaches. The incorporation of context extraction further enhances the verifier's performance by mitigating the limitations imposed by LLMs' context windows. Our results indicate that Auto-Verify not only refines individual reasoning steps but also serves as a foundational framework that can be integrated with other reasoning enhancement methods.

Despite its effectiveness, Auto-Verify still faces challenges in handling complex out-of-domain mathematical problems, suggesting opportunities for future research in further refining context extraction and verifier robustness. Overall, our work presents a scalable and practical solution to improving mathematical reasoning in LLMs, paving the way for more reliable AI-driven problem-solving.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Chen, G., Liao, M., Li, C., and Fan, K. Step-level value preference optimization for mathematical reasoning. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7889–7903, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.463. URL https://aclanthology.org/2024.findings-emnlp.463/.

Chen, G., Liao, M., Li, C., and Fan, K. Alphamath almost zero: Process supervision without process. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=VaXnxQ3UKo.

Chen, J. C.-Y., Saha, S., and Bansal, M. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. *arXiv preprint arXiv:2309.13007*, 2023.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Cohen-Wang, B., Shah, H., Georgiev, K., and Madry, A. Contextcite: Attributing model generation to context. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=7CMNSqsZJt.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dubois, Y., Li, C. X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P. S., and Hashimoto, T. B. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Golovneva, O., Chen, M. P., Poff, S., Corredor, M., Zettlemoyer, L., Fazel-Zarandi, M., and Celikyilmaz, A. ROSCOE: A suite of metrics for scoring step-by-step reasoning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=xYlJRpzZtsY.

Gulcehre, C., Paine, T. L., Srinivasan, S., Konyushkova, K., Weerts, L., Sharma, A., Siddhant, A., Ahern, A., Wang, M., Gu, C., et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Gutiérrez, B. J., Shu, Y., Gu, Y., Yasunaga, M., and Su, Y. Hipporag: Neurobiologically inspired long-term memory for large language models. *arXiv preprint arXiv:2405.14831*, 2024.

Hao, S., Gu, Y., Ma, H., Hong, J. J., Wang, Z., Wang, D. Z., and Hu, Z. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

He, C., Luo, R., Bai, Y., Hu, S., Thai, Z. L., Shen, J., Hu, J., Han, X., Huang, Y., Zhang, Y., et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 2023.

Luo, L., Liu, Y., Liu, R., Phatale, S., Lara, H., Li, Y., Shu, L., Zhu, Y., Meng, L., Sun, J., et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

MAA. American invitational mathematics examination. Online, 2024. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Miao, N., Teh, Y. W., and Rainforth, T. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436*, 2023.

Pan, L., Albalak, A., Wang, X., and Wang, W. Y. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*, 2023a.

Pan, L., Saxon, M., Xu, W., Nathani, D., Wang, X., and Wang, W. Y. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*, 2023b.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wang, X., Salmani, M., Omidi, P., Ren, X., Rezagholizadeh, M., and Eshaghi, A. Beyond the limits: A survey of techniques to extend the context length in large language models. *arXiv preprint arXiv:2402.02244*, 2024b.

Wang, Z., Li, Y., Wu, Y., Luo, L., Hou, L., Yu, H., and Shang, J. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. *arXiv preprint arXiv:2402.02658*, 2024c.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xie, Y., Goyal, A., Zheng, W., Kan, M.-Y., Lillicrap, T. P., Kawaguchi, K., and Shieh, M. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024.

Yu, F., Gao, A., and Wang, B. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 858–875, 2024.

Zhang, D., Wu, J., Lei, J., Che, T., Li, J., Xie, T., Huang, X., Zhang, S., Pavone, M., Li, Y., et al. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *arXiv preprint arXiv:2410.02884*, 2024a.

Zhang, Y., Sun, R., Chen, Y., Pfister, T., Zhang, R., and Arik, S. Ö. Chain of agents: Large language models collaborating on long-context tasks. *arXiv preprint arXiv:2406.02818*, 2024b.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A. Algorithm of Auto-Verify

Algorithm 2 Auto-Verify

Input: question Q, reasoning model f_r, verification model f_v, context extraction function $Context(\cdot)$, max iteration of self-refineN_{refine}, reasoning prompt p_m, self-refine prompt p_r, verifier prompt p_f, a function that extract answer from '$\backslash boxed\{\}$' answer($\cdot$)
Output: The answer of the question $final\_answer$.
k = 0
while True **do**
  k += 1
  if k == 1 **then**
    $S_k \sim f_r(\cdot|Q, p_m)$
  else
    $S_k \sim f_r(\cdot|S_{k-1}, \ldots, S_1, Q, p_m)$
  end if
  refine_list = [ ]
  for $l = 1$ **to** N_{refine} **do**
    context = Context($S_{k-1}, \ldots, S_1, Q$)
    correct, reasons $\sim f_v(\cdot|$context, $S_k, p_v)$
    if correct == True **then**
      break
    else
      step = $f_r(\cdot|S_k, reason, context, refine\_list, p_r)$
      refine_list.append((reasons, step))
      $S_k = step$
    end if
  end for
  if '$\backslash boxed\{\}$' in S_k **then**
    $final\_answer$ = answer(S_k)
    break
  end if
end while
return $final\_answer$

B. Verifier Prompt

Verifier Prompt

[Question]
{question}
[Reasoning Steps]
{correct reasoning steps}
[instruction]
You are given two tasks. Please limit your output in 612 tokens.
First, output "[Next Reasoning Step with Fabrication Hallucination]", then on a new line, Please generate the next brief reasoning step to incorrectly continue the reasoning process based on a question and a series of correct reasoning steps.The next reasoning step you generate does not necessarily result in an instant final answer. And you should follow the hallucination generation instruction below to generate the next reasoning step.
 # Guidelines for Creating a Hallucinated Reasoning Step with Extrinsic Error When you generate the next reasoning step, intentionally insert subtle logical mistakes by misreferencing or incorrectly replicating content from earlier steps. These mistakes should be noticeable upon careful review but not immediately obvious. Follow these guidelines:
- **Copying Errors:** When you copy a portion of a previous reasoning step, deliberately alter a key detail. For example, if an earlier step states "Item A costs 5 * 2 = 10," you might later write "Since item A costs 5 * 3 = 15…" This small change introduces an error without making the inconsistency glaringly obvious.
- **Creating Logical Leaps:** Make conclusions or logical jumps that do not directly follow from the earlier steps. These leaps should lead to an incorrect final answer, further contributing to the inconsistency. # Constraints - **Subtlety:** The introduced inconsistencies should be minor enough that they don't immediately stand out, yet clear upon closer inspection.
- **Coherence:** Aside from the intentional errors, the rest of the content should remain coherent and logically sound.
- **Contextual Fit:** Ensure that the incorrect details blend well with the surrounding text. The inconsistencies should fit naturally within the overall narrative, reducing the chance of immediate detection.
- After you generate the next hallucination reasoning step, please first output "[End of Step]", then output "[Explanation]", then on a new line, shortly explain why the next reasoning step is incorrect, end with outputting "[End of Explain]".
Second, output "[Context]", then on a new line, output some previous sentences that contribute most to the step you generate, that is, provide brief context of the next reasoning step you generate with Fabrication Hallucination.
Then, on a new line, output "[End of Context]" to indicate the end of generation.