

CoSineVerifier: Tool-Augmented Answer Verification for Computation-Oriented Scientific Questions

Ruixiang Feng^{2*†}, Zhenwei An^{1*}, Yuntao Wen², Ran Le¹, Yiming Jia¹, Chen Yang¹,
 Zongchao Chen¹, Lisi Chen², Shen Gao^{2‡}, Shuo Shang^{2‡}, Yang Song^{1‡}, Tao Zhang¹

¹Nanbeige Lab, BOSS Zhipin

²University of Electronic Science and Technology of China, Chengdu, China
 {songyang,leran,anzhenwei}@kanzhun.com, {shengao,chenlisi}@uestc.edu.cn,
 {fengrx02,jedi.shang}@gmail.com

Abstract

Answer verification methods are widely employed in language model training pipelines spanning data curation, evaluation, and reinforcement learning with verifiable rewards (RLVR). While prior work focus on developing unified verifiers applicable across multiple reasoning scenarios, significant challenges remain in computation-oriented scientific domains, such as algebraic equivalence checking and physical constant substitution. In this paper, we introduce CoSineVerifier, a tool-augmented verifier that leverages external executors to perform precise computations and symbolic simplifications. CoSineVerifier enables robust verification that goes beyond simple semantic matching. We propose a novel two-stage pipeline, which begin with cold-start fine-tuning and followed by multi-turn reinforcement learning with tool integration. Extensive experiments conducted on STEM subjects, general QA, and long-form reasoning tasks demonstrates strong generalization of CoSineVerifier. The results shows that the CoSineVerifier achieves state-of-the-art performance on VerifyBench-Hard and SCI-Bench. And we also employ our CoSineVerifier in RLVR as a reward model, the results show that it consistently outperforms both rubric-based and model-based verifiers on AIME’24 and AIME’25, demonstrating strong potential to enhance reasoning capabilities of LLM. Our model is released at <https://huggingface.co/Nanbeige/CoSineVerifier-Tool-4B>.

1 Introduction

The evolution of large language models (LLMs) training—from relying solely on human-annotated data to enabling models to self-evolve—has imposed increasingly stringent demands on how we evaluate the quality of LLM outputs. In this context, answer verification, which compares a model output against a reference, has become a critical component throughout data curation, evaluation, and model training pipelines. In the phase of data curation, answer verification methods are extensively employed to guarantee the consistency, fidelity, and accuracy of synthetic corpora by many works [1, 2, 3, 4]. At the evaluation stage, these methods provide a robust alternative to assess correctness beyond exact match (EM), enabling semantic-level comparison and format-robust scoring [5, 6, 7, 8]. For model training, Reinforcement Learning with Verifiable Rewards (RLVR) relies on verifiers to produce reward signals, imposing stringent requirements on verifier accuracy, calibration, and latency/throughput [9, 10, 11].

*Equal contribution.

†Work done during internship at Nanbeige Lab.

‡Corresponding author.

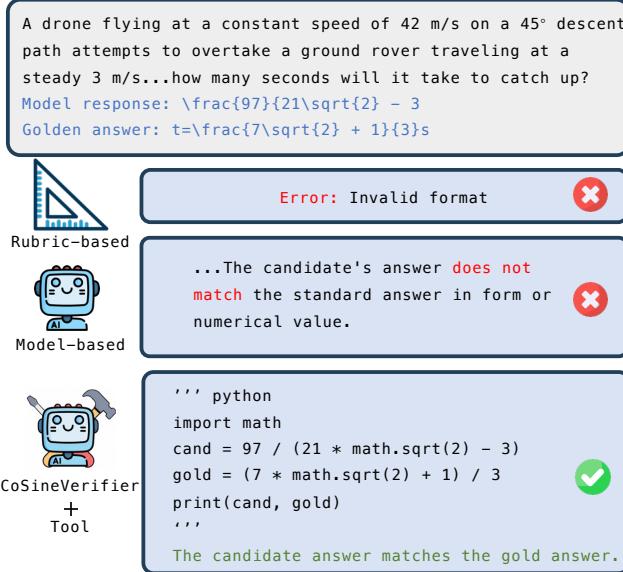


Figure 1: Comparison between CoSineVerifier and existing methods, which demonstrates our CoSineVerifier can enable accurate judgment on calculation-intensive scientific scenarios

Existing verification methods primarily rely on direct semantic comparisons between model outputs and reference answers. Rubric-based verifiers [12], which depend on structured outputs (e.g., Math-Verify evaluating equivalence in Markdown or step-by-step formats), can fail even on simple computation-heavy cases such as comparing $2*2$ and 4 . Alternatively, many studies [13, 14, 15] employ LLM-based verifiers that assess answers through chain-of-thought (CoT) reasoning or implicit reasoning encoded in model parameters. However, these verifiers are still prone to hallucination and inconsistencies in numerical or symbolic reasoning. For instance, when comparing matrices with hundreds of entries or substituting physical constants with specific values, LLM-based verifiers often produce incorrect judgments. The lack of robust computation skills prevents reasoning models from receiving accurate rewards in RLVR tasks within complex scientific domains, thereby limiting their overall potential.

We present CoSineVerifier, a tool-augmented LLM-based verifier designed for checking answers to computation-heavy scientific questions. As shown in Figure 1, reference answers and model outputs often express the same numerical value using different formulas. To enable robust verification, a reliable verifier must be capable of performing arithmetic transformations before making a judgment. In this paper, we propose to equip the verifier with external tools such as a Python interpreter. Therefore, our proposed CoSineVerifier can perform precise computations on both formulas to derive aligned numerical results, thereby ensuring accurate verification.

However, foundation models are not well-tuned on precise tool-augmented verification, which may issue inaccurate and redundant tool calls during the process. To address this challenge, we propose a novel verifier training methodology. We begin by collecting a diverse set of questions spanning multiple domains, including mathematics, natural sciences, and logical reasoning. We first conduct a comprehensive analysis of how common verification methods fail on these questions, identifying typical failure modes in such settings. As illustrated in Figure 2, we identify two major challenges: (i) Calculation inaccuracy represents the largest proportion of verification failures, highlighting the limitations of existing verifiers in computation-intensive contexts. (ii) Error types are diverse and follow a long-tail distribution, making it difficult for previous data augmentation approaches to achieve comprehensive coverage. Based on these insights, we design an automatic method for constructing verifier training data that synthesizes challenging computation-heavy instances. This approach increases both the density of queries prone to verification errors and the proportion of cases that require computational tool-augmented reasoning.

After constructing this diverse and challenging training dataset, we propose a two-stage training strategy to equip the verifier with effective tool-use capabilities in computation-heavy scenarios: (i) cold-start supervised fine-tuning, followed by (ii) reinforcement learning with tool-call encourage-

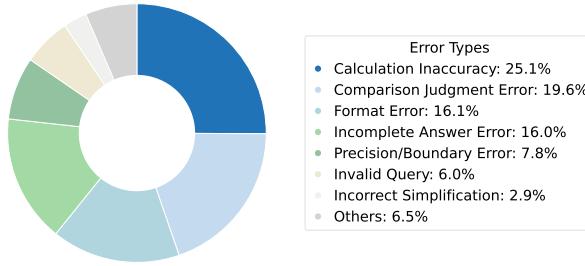


Figure 2: Error distribution on existing verification methods. More Details can be found in the Appendix A.

ment. The rationale for this approach is that while the cold-start phase enables the model to invoke tools, such invocations remain unreliable—only 76% of tool-invoked verifications are initially correct. To address this issue, the second stage employs agentic reinforcement learning with rewards designed to explicitly encourage precise and effective tool usage, which boosts tool-invoked verification accuracy from 76% to 90%.

To summarize, our contributions are threefold:

- We propose **CoSineVerifier**, a tool-augmented answer verification model. By invoking external tools (e.g., Python interpreter), it performs precise computation and algebraic simplification to assess numeric equivalence between scientific formulas.
- We develop a two-stage training framework consisting of cold-start fine-tuning followed by a reinforcement learning stage that encourages effective tool use, substantially increasing the correctness of tool-invoked verifications on computation-heavy questions.
- Experimental results demonstrate the effectiveness of CoSineVerifier in terms of both verification performance and downstream RLVR applications. CoSineVerifier achieves state-of-the-art results on VerifyBench and Sci-Bench. As a fast and accurate reward model in RLVR, it also consistently outperforms rubric-based and CoT-only verifiers on both AIME’24 and AIME’25.

2 Related Works

2.1 Answer Verification

Answer verification is the automated process of deciding whether a candidate response is factually correct to the provided reference answer. Significant progress has been made in the development of LLM-output verification[16, 17], which can broadly be divided into two main classes: rule-based verification and model-based verification. Early rule-based methods like Math-Verify[12] extract the final answer via heuristics and compare it against the ground truth using predefined rules. However, these approaches struggle with unstructured model outputs and tend to incur a high false-negative rate[18]. To address these shortcomings, model-based verifiers exploit the flexibility of LLMs to interpret and evaluate unpredictable output. Xverify[13] and CompasVerifier[14] are designed to provide efficient and robust answer verification by carefully constructing training examples. While effective, they are constrained by the verifier’s inherent reasoning and calculation competency—particularly in computation-intensive math and unit-sensitive STEM problems.

In this paper, we concentrate on the model-verifiacton reward setting , notably in calculate-intensive and science domains where existing verifiers usually fail. To overcome these limitations, we propose CosineVerifier, which endows the model with external tool capabilities (e.g., a Python interpreter and a unit-conversion toolkit) to reliably validate diverse LLM outputs while delivering efficient, robust and trustworthy verdicts.

2.2 Tool Integrated Reasoning

A growing line of work investigates improving LLM reasoning by augmenting models with external tools, yielding consistent gains in accuracy, calibration, and sample efficiency across math, program, and multi-turn QA tasks[19, 20, 21, 22, 23, 24]. ReTool[25], ToRL[26] and SimpleTIR[27] train models to interleave real-time code execution with language reasoning and to learn when/how to call tools via reward signals, yielding strong gains on math-reasoning benchmarks. VTool-R1[28] integrates Python-based visual editing tools (box, mask, highlight) into the RFT loop so the model learns to interleave intermediate visual steps with textual chain-of-thought. Extending beyond code, Search-R1[29] casts web search as an action space and trains query generation/revision via outcome rewards for multi-turn retrieval-augmented reasoning. Taken together, these approaches recast reasoning over a unified tool-action space, spanning code, vision, and search—where outcome-level rewards teach models when to plan, execute, and reflect, yielding stronger generalization and reliability beyond base models.

To the best of our knowledge, CosineVerifier is the first outcome answer verifier that augments LLM-based judgments with executable tools, enabling complex unit-consistent calculations, tolerance-aware numeric checks and robust string format equivalence for general verification process.

3 Preliminaries

Answer verification We consider the task of outcome reward verification as follows: given a set of questions $Q = \{q_1, q_2, \dots, q_n\}$, their targeted model-generated answers $A = \{a_1, a_2, \dots, a_n\}$ and their corresponding reference answers $A_{ref} = \{a_{ref}^1, a_{ref}^2, \dots, a_{ref}^n\}$, the goal is to evaluate the correctness of the model-generated answers A , which can be formulated as:

$$R = V_\theta(Q, A, A_{ref}) \quad (1)$$

where $R = \{r_1, r_2, \dots, r_n\}$ with $r \in \{0, 1\}$ is the set of binary verification results and V_θ is the verification model parameterized by θ .

4 Methodology: CoSineVerifier

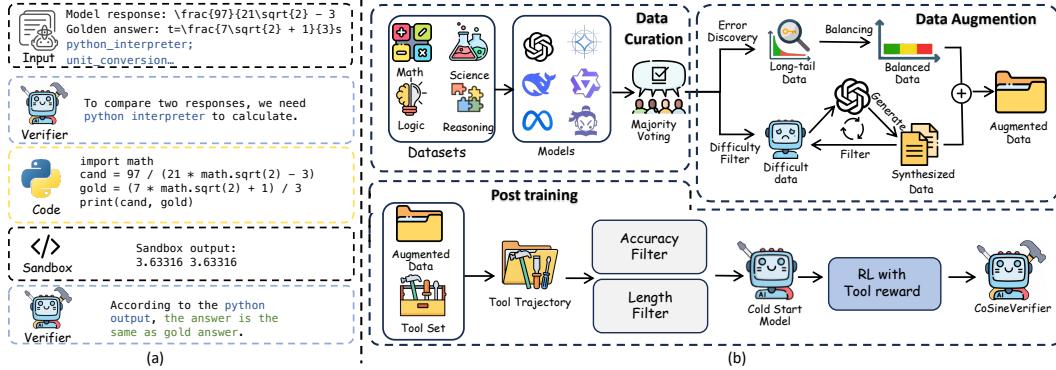


Figure 3: (a) Workflow of tool-augmented verification. (b) The overall training framework of CoSineVerifier.

In this section, we outline the workflow and training procedure of CoSineVerifier, which is endowed with agentic tool-use capabilities for accurate outcome verification. Figure 3 shows the overall framework of CoSineVerifier. Section 4.1 describes how our CoSineVerifier interleaves reasoning traces with dynamic tool execution to reach a final verdict. Section 4.2 details our large-scale, multi-domain data collection and annotation pipeline, Section 4.3 then describes our data augmentation strategy on long-tail and difficult data. Finally, Section 4.4 presents the training recipe of CoSineVerifier, including cold-start supervised fine-tuning and reinforcement learning.

4.1 Tool-augmented Verification

To facilitate the reliable outcome verification, we equip our verification model with external tools that can dynamically execute computations and norlizae units, enabling precise judgments on calculation-intensive and science tasks. Specifically, as shown in Figure 3 (a), when facing a verification problem, our CoSineVerifier-4B-Tool first decides whether to use an external tool. If external tool call is needed, we parse the tool arguments(e.g. python code) and send them to sandbox environments. After the sandbox environment completes the execution and returns the output, CoSineVerifier-4B-Tool gives its final judgement based on the sandbox output. Full case study can be found in Appendix D.

4.2 Data Curation

Data Collection Answer verification in real-world applications is challenging for two reasons: (i) the diversity of questions encountered in practice, and (ii) the variability of model outputs (format, length, and reasoning style). To address these challenges, we construct a large-scale candidate training data, comprising 1.14 million samples. These samples were collected from 15 datasets using 14 large language models (LLMs), spanning mathematics, science, logical reasoning, and commonsense knowledge domains. This provides broad coverage for training robust verification models. In addition, to strengthen CosineVerifier on practical verification tasks, we deliberately go beyond static benchmarks (in contrast to prior collections such as CompassVerifier[14] and Xverify[13]). A substantial portion of our data is sampled from competition-level math and science questions, where verification often fails in practice. During the construction of of our data, we also ensure cross-domain coverage, and to further increase robustness, we also prepend different prompt prefixes(e.g. Let’s think step by step) on a subset of training data, promoting generalization to diverse prompting styles.

Data Annotation After constructing a large and diverse training data, we introduce an iterative annotation pipeline that produces reliable verification labels. We first employ a set of models (Compass-Verifier-32B, qwen3-4b-2507, and Qwen3-32B), with each model providing three independent annotations for a total of nine per sample. We then retain 63,714 samples that show any disagreement among these nine annotations. For samples that showcase high disagreement rate(defined as more than three out of nine annotations differing), we further send them to a more powerful model(GPT-o3) with majority voting, ensuring high label fidelity on harder questions.

4.3 Data augmentation

Long-tail data augmentation To understand when the verifier performs poorly, we conducted a discrepancy analysis over 63,374 disagreement cases. We first prompted Qwen3-32B to produce an error category for each sample and then leverage GPT-o3 to summarize these categories into 15 error types. As shown in Figure 2, all error types reveal a pronounced skew: the top five categories—dominated by Calculation Inaccuracy and Exact Match Failure, account for over 85% of errors, while the remaining ten categories are sparsely represented. This distribution makes low-frequency errors harder to learn. To counter this imbalance, we use gpt-o3 to synthesize targeted long-tail verification examples under constrained generation settings, yielding 10k long-tail synthesize instances. We further incorporate these samples into training for both the labeling verifier and the tool-augmented verifier, improving coverage of rare error modes. All training data statistic can be found in Appendix C.

Difficulty-oriented data augmentation After cold-start, our model acquired foundational tool-use and verification skills. To further amplify its performance, we focused on curating a high-difficulty post-training dataset using an iterative self-instruct framework. In each round, GPT-o3 synthesized new questions conditioned on the current hard cases. Our cold-start model then attempted these questions. We only retrained incorrect instances, and merged these instances with the reference set for the next iteration. After four such iterations, the procedure produced a reinforcement-learning corpus of 9,456 examples. A detailed algorithm is depicted in 1.

Algorithm 1 Iterative Difficult Data Synthesis

Input: Initial hard set \mathcal{D}_0 ; questioner Q ; solver S ; rounds T .
Output: Reinforcement-learning corpus \mathcal{R} .

```

1  $\mathcal{R} \leftarrow \{\}$   $\mathcal{D} \leftarrow \mathcal{D}_0$ 
2 for  $t \leftarrow 1$  to  $T$  do
3    $\mathcal{Q}_t \leftarrow Q(\text{conditioned on } \mathcal{D})$  // Conditioned generation by Questioner
4    $\mathcal{F}_t \leftarrow \{x \in \mathcal{Q}_t \mid S(x) \text{ is incorrect}\}$  // Accumulate failures as next-round context
5    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{F}_t$   $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{F}_t$ 
6 end
7 return  $\mathcal{R}$ 
```

4.4 Post Training

4.4.1 Cold-Start

To ensure verification accuracy and robustness, we equip CosineVerifier with two executable tools: a Python interpreter and a unit-conversion utility. During the preliminary experiment of CoSineVerifier, we find that directly apply reinforcement learning on base model yields performance degradation, due to the base model’s limited tool-calling capabilities. We therefore implement a cold-start stage to equip the model with foundational tool-use proficiency. Specifically, we first collect real-world tool-use traces from our training data and retain only those cases that (i) produce a correct final answer and (ii) contain explicit tool-call trajectories. To boost the efficiency of CoSineVerifier, we discard model internal `<think>...</think>` content, and further filter out responses longer than 200 tokens to keep the cold-start corpus compact and efficient. This procedure yields 29,339 tool-usage examples used to bootstrap training.

During training CoSineVerifier-4B-Tool, we mask the loss on tool-execution outputs (e.g., interpreter traces and unit-conversion results), which prevents the model from merely imitating tool dumps and improves optimization stability. We also train CoSineVerifier-Label series models, which are train on the full 63,374 disagreement data together with 10k model-generated long-tail instances, and providing verdict as binary classification with targets *Correct* and *Incorrect*.

4.4.2 Reinforcement Learning

Overall framework For CoSineVerifier, we also undergone reinforcement learning to incentivize more reasoning ability. Specifically, we adopt the DAPO [30] algorithm, which is an improved variant of GRPO[31]. Given a prompt–answer pair (q, a) , we draw G rollouts $\{s_i\}_{i=1}^G$ from the behavior policy $\pi_{\theta_{\text{old}}}$. With access to the grounded answer a , each rollout receives a scalar reward $R_i = R(s_i, a)$ and we then optimize the current policy π_θ using the clipped policy-gradient objective below:

$$\begin{aligned} \mathcal{J}_{\text{DAPO}}(\theta) = & \mathbb{E}_{(q,a) \sim \mathcal{D}, \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|q)} \\ & \left[\frac{1}{\sum_{i=1}^G |o_i|} \sum_{i=1}^G \sum_{t=1}^{|o_i|} \min \left(r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip} \left(r_{i,t}(\theta), 1 - \varepsilon_{\text{low}}, 1 + \varepsilon_{\text{high}} \right) \hat{A}_{i,t} \right) \right] \quad (2) \\ \text{s.t. } & 0 < \left| \{o_i \mid \text{is_equivalent}(a, o_i)\} \right| < G, \end{aligned}$$

where

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}, \quad \hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_i\}_{i=1}^G)}{\text{std}(\{R_i\}_{i=1}^G)}. \quad (3)$$

Reward design We decompose the CoSineVerifier’s outcome-based reward into two terms. The first is an answer-correctness component:

$$R_{\text{ans}} = \mathbb{I}\{\hat{y} = y\},$$

which assigns 1 when the model’s prediction \hat{y} matches the reference y , and 0 otherwise.

To promote tool use during training, we introduce a tool-use encourage reward. Let u denote the number of tool invocations in the trajectory. We award a bonus when the model is correct and uses an

Model	VerifyBench	VerifyBench(Hard)	SCI-Bench	Avg. Tokens
CoT Verifier				
Closed-source Models				
o3[32]	<u>96.1</u>	88.7	87.5	206.7
GPT-4o[33]	96.0	84.6	86.0	192.4
Gemini2.5-Flash[34]	96.0	86.0	85.9	193.0
Open-source Models				
GPT-oss-20B[35]	92.2	84.7	85.0	221.0
LLaMA3.3-70B-Instruct[36]	94.8	77.2	84.8	347.3
Qwen3-4B[37]	92.6	80.3	82.0	1156.7
Qwen3-8B[37]	93.7	83.6	83.9	926.6
Qwen3-32B[37]	94.7	85.2	83.5	798.8
Qwen3-4B-Instruct-2507[37]	94.7	84.1	82.4	869.7
Qwen3-235B-A22B-2507[37]	94.4	87.7	82.6	1885.3
CompassVerifier-7B (CoT)[14]	93.5	82.6	84.2	234.7
CompassVerifier-32B (CoT)[14]	95.9	86.5	85.5	213.0
CoSineVerifier-Tool-4B	96.6	91.9	89.7	95.3
Labeling Verifier				
XVerify-8B-I[13]	92.5	83.3	78.1	1.0
CompassVerifier-7B[14]	93.5	85.2	85.7	1.0
CompassVerifier-32B[14]	96.3	88.9	85.3	1.0
CoSineVerifier-4B	<u>95.7</u>	85.4	<u>85.9</u>	1.0
CoSineVerifier-32B	<u>95.7</u>	90.0	86.4	1.0

Table 1: Main results on three verify benchmarks. We separate CoT verifier and Labeling verifier for fair comparison, where CoT verifier output a reasoning chain before achieving final answer while labeling verifier directly gives it’s verdict. **Bold** highlights the overall best accuracy, while underline highlights the second best accuracy. We report mean@3 accuracy and average token used per question.

external tool, and impose a penalty when the model is incorrect without relying on function calling:

$$R_t = \begin{cases} 0.5, & \text{if } \mathbb{I}\{\hat{y} = y\} = 1 \text{ and } u > 0, \\ -0.5, & \text{if } \mathbb{I}\{\hat{y} = y\} = 0 \text{ and } u = 0, \\ 0, & \text{otherwise.} \end{cases}$$

The final reward is the sum of the correctness and tool-encourage terms:

$$R = R_{\text{ans}} + R_t.$$

This shaping explicitly encourages effective tool use trajectories that leverage tools to reach correct answers and discourages failures that eschew tools.

5 Experiments

5.1 Experiment Setups

Datasets and Baselines We evaluate the CoSineVerifier family on three benchmarks: VerifyBench, VerifyBench-Hard[38], and Sci-Bench [39], which are challenging verification benchmarks spanning math, science, commonsense knowledge and general reasoning domains with 2000, 1000 and 2500 samples. Our baselines mainly include two kinds of models: labeling verifier and COT verifiers, with difference on whether contain reasoning chains for verification and we report accuracy as mean@3 and efficiency as average output tokens per verdict on these benchmarks.

Implementation details Our CoSinVerifier-4B-Tool, CosineVerifier-4B and CoSinVerifier-32B are trained from Qwen3-4B-Instruct-2507, Qwen3-4B, and Qwen3-32B[37]. We use OpenRLHF[40] for supervised fine-tuning and verl[41] framework for reinforcement learning. All models are trained for one epoch on 32xA800 80G GPUS during supervised finetuning with 1e-5 learning rate, and 60 step for reinforcement learning with 1e-6 learning rate. All training details are listed in Appendix C

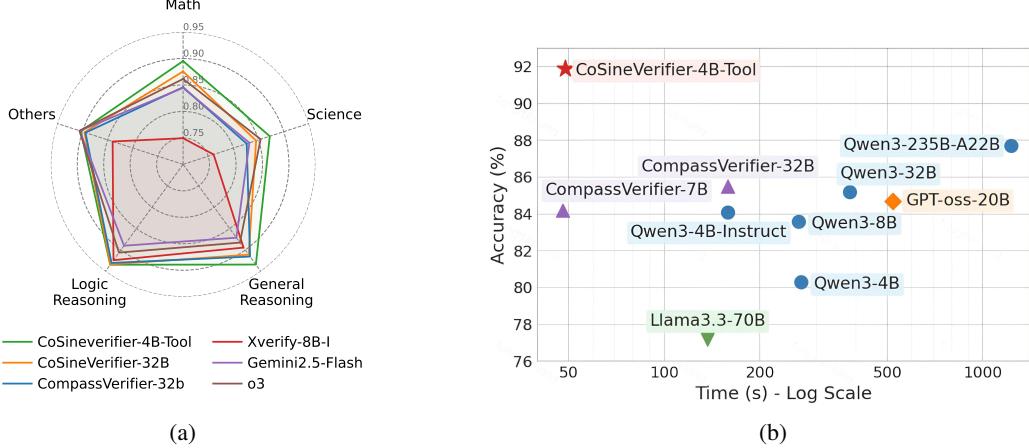


Figure 4: Performance Analysis. (a) Analysis on accuracy across different domains. (b) Inference efficiency analysis of CoSineVerifier-4B-Tool.

5.2 Main Results

CoSineVerifier series achieve strong verification accuracy compared to baselines As shown in Table 1, our CoSineVerifier series outperforms all baselines across three verify benchmarks. Specifically, for CoT Verifier, our CoSineVerifier-Tool-4B surpasses baselines by 0.5%-14.7% with noticeably less tokens, especially on harder benchmarks like VerifyBench(hard) and Sci-Bench, where complex calculation and long-context string comparing is needed to verify the LLM-generated answer. In addition, for labeling verifier, CoSineVerifier-4B and CoSineVerifier-32B also surpass other labeling verifiers by 0.7%-1.1%, demonstrating stronger verification skills under single-token scenarios.

Small models with external tools excel large models Figure 4 demonstrates the performance comparison of our CoSineVerifier against various model across different domains. Among three benchmarks, even close-source SOTA models like o3 still struggle with reliable verification on compute-intensive and scientific reasoning tasks. In contrast, both CoSineVerifier-4B-Tool and CoSineVerifier-32B demonstrate exceptional proficiency in the Math and Science domains while achieve competitive or even superior performance on the remaining domains. Despite being much more smaller on parameter size, the superior accuracy exhibited by CoSineVerifier-4B-Tool validates the efficacy of integrating external calculation tools to enhance the model’s verification capabilities. These results demonstrate the feasibility of tool-augmented, small-scale models to exceed the performance of large SOTA models in complex verification tasks.

CoSineVerifier-4B-Tool shows superior efficiency while maintaining high accuracy We further evaluate the efficiency of CoSineVerifier-4B-Tool against several baselines in Figure 4. To ensure a fair comparison, all models were benchmarked on 4xNVIDIA A800 GPUs. We report the mean inference latency computed over 1000 randomly sampled prompts. While CoSineVerifier-4B-Tool achieves high accuracy, incorporating external tools does not incur significant inference latency. We attribute this efficiency to two primary factors: (1) our efficiecy-driven cold-start data construction methodology, which explicitly optimizes for short and precise tool-calling trajectories, resulting in 2.4x-19x times less token usage compared to other models. (2) the computational efficiency afforded by the model’s compact 4B parameter count, which minimizes the overhead of the backbone model during inference.

5.3 Ablation Study

Two stage training is crucial for CoSineVerifier-4B-Tool As shown in Figure 5(a), we conduct ablation studies on each module’s contribution to our two-stage training framework. We first observe that after removing cold start stage for CoSineVerifier-4B-Tool, the model’s performance drop sharply. We then discover this is because model without cold-start fails to conduct accurate function calling

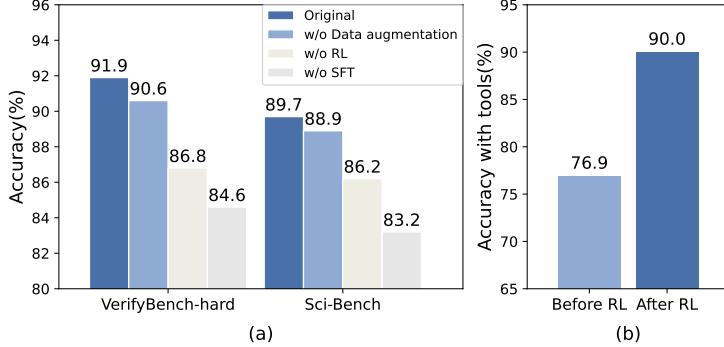


Figure 5: (a) Ablation study of CoSineVerifier-4B-Tool. (b) Average verification accuracy with tools before RL and after RL

and often becomes trapped in tool-use loops, harming performance while significantly increase inference latency. In addition, model trained without reinforcement learning(RL) also demonstrate weaker verification performance. To investigate this, we also analyze the accuracy on samples that have tool trajectories before and after RL stage. As shown in Figure 5(b), the accuracy on samples with tools improved markedly from 76.9% to 90.0%. This result indicates that RL teaches the model how to leverage tools for accurate verification, rather than merely to initiate a tool call. In conclusion, each stage of our two-stage framework is crucial for robust verifier training. Together, they equip the model to determine precisely when and how to use tool calls for verification.

Data augmentation can further boost performance As shown in Figure 5(a), removing data augmentation also yields inferior performance. We attribute this to the fact that verification tasks often contain rare but difficult samples, such as truncated responses, multiple self-reflections, and meaningless repetitions. These samples are relatively more difficult to verify and are underrepresented in ablation training data, preventing the model from learning to handle them as effectively as common verification samples. Incorporating our data augmentation methods can exposing the model to a more balanced and challenging training distribution, which ensure the our model have high verification accuracy in rare and difficult circumstances.

6 Application in RLVR

The primary objective of our CoSineVerifier series is to establish a reliable and efficient outcome reward verifier tailored for real-world data processing and reinforcement learning. To avoid potential biases inherent in existing benchmarks and validate the efficacy of CoSineVerifier as a reward model with RL training, we further evaluate answer-verification methods in RLVR settings. Specifically, utilizing an on-policy GRPO algorithm, we train Qwen3-4B-Instruct-2507 on competition-math problems with 42K training data drawn from DAPO-Math-17k[30], OpenR1-Math-220k[42], and DeepScaleR-Preview[43]. We compare our approach against various verifiers, including both rule-based and model-based methods, under identical training configurations. We report mean@32 accuracy on challenging math benchmarks AIME 2024 and AIME 2025 and comprehensive training details are provided in Appendix C.

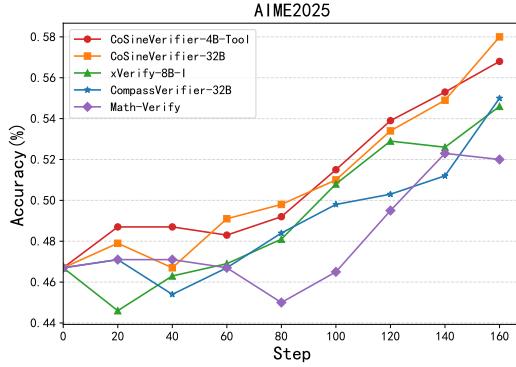


Figure 7: Accuracy of AIME 2025 during training process

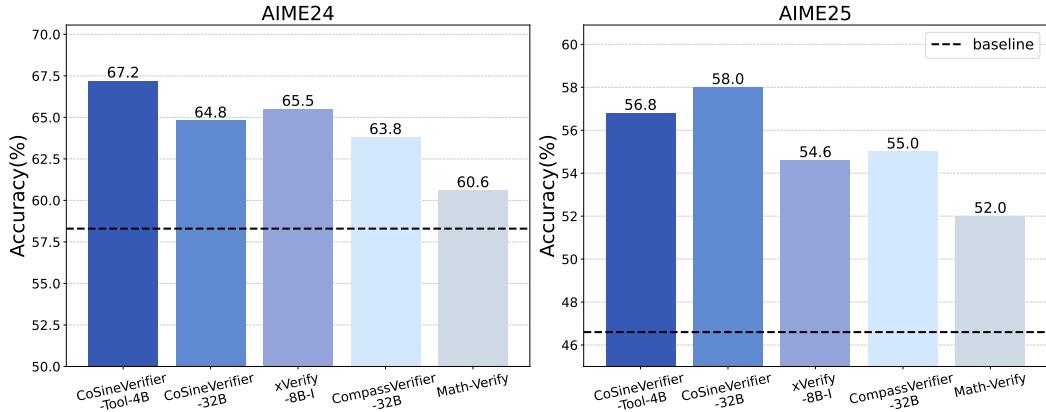


Figure 6: Results of using CoSineVerifier as a reward model on AIME24 and AIME25 benchmark

RL training result As illustrated in Figure 6, training with our CoSineVerifier enables the base model to achieve stronger reasoning capabilities, yielding improvements of 1.7%–6.6% on AIME 2024 and 1.8%–6.0% on AIME 2025. Moreover, a clear gap exists between rule-based and model-based verifiers. This highlights that static methods alone are inadequate for mapping diverse LLM outputs to reference answers. We believe a precise and reliable reward signal is beneficial for stable RL training, especially when the model is already well-tuned and requires more challenging RL training data to surpass its reasoning ceiling.

From the perspective of training process We also dive into the accuracy change trend on AIME 2025 during RL training process. We aim to answer the following question: what is the impact of using different verifiers on downstream task performance under identical RL training settings? As shown in Figure 7, our CoSineVerifier consistently yields a clear lead over all baselines, demonstrating that base model can benefit significantly from optimizing verifiers during RL training. In contrast, weaker verifier like Math-Verify perform worse than CoSineVerifier series, demonstrating a direct correlation between a verifier’s capability and base model’s reasoning performance during reinforcement learning. As recent research increasingly utilizes challenging datasets that elicit highly complex model responses and reference answers[44, 45], this trend naturally motivates us to develop a more reliable verification system that ensures robust and accurate reward signaling to boost the performance of base models.

7 Conclusion

In this work, we introduce the CoSineVerifier series, a family of answer verifiers designed for computation-oriented scientific questions. Our core contribution, CoSineVerifier-Tool-4B, demon-

strates that a compact 4B model, when augmented with external tools like a Python interpreter, can significantly outperform much larger models on challenging verification benchmarks. This tool-integrated approach enables the precise evaluation of algebraic equivalences and physical constant alignments, addressing a key weakness of existing LLM-based verifiers. Furthermore, when applied as a reward model in RLVR tasks, CoSineVerifier series yields clear performance improvements on AIME24 and AIME25, highlighting the importance of accurate, tool-enhanced reward signals for training more capable models.

References

- [1] Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025.
- [2] Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Ilia Kulikov, Kyunghyun Cho, Dong Wang, Yuandong Tian, Jason E Weston, et al. Naturalreasoning: Reasoning in the wild with 2.8 m challenging questions. *arXiv preprint arXiv:2502.13124*, 2025.
- [3] Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.
- [4] Alon Albala, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, et al. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models. *arXiv preprint arXiv:2502.17387*, 2025.
- [5] Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. S²r: Teaching llms to self-verify and self-correct via reinforcement learning. *arXiv preprint arXiv:2502.12853*, 2025.
- [6] Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*, 2024.
- [7] Samuel J. Paech. Eq-bench: An emotional intelligence benchmark for large language models. *arXiv preprint arXiv:2312.06281*, 2023.
- [8] Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, et al. Writingbench: A comprehensive benchmark for generative writing. *arXiv preprint arXiv:2503.05244*, 2025.
- [9] Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, et al. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*, 2025.
- [10] Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*, 2025.
- [11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [12] Hynek Kydlíček. Math-Verify: Math Verification Library.
- [13] Ding Chen, Qingchen Yu, Pengyuan Wang, Wentao Zhang, Bo Tang, Feiyu Xiong, Xinchi Li, Minchuan Yang, and Zhiyu Li. xverify: Efficient answer verifier for reasoning model evaluations. *arXiv preprint arXiv:2504.10481*, 2025.
- [14] Shudong Liu, Hongwei Liu, Junnan Liu, Linchen Xiao, Songyang Gao, Chengqi Lyu, Yuzhe Gu, Wenwei Zhang, Derek F Wong, Songyang Zhang, et al. Compassverifier: A unified and robust verifier for llms evaluation and outcome reward. *arXiv preprint arXiv:2508.03686*, 2025.

- [15] Nuo Chen, Zhiyuan Hu, Qingyun Zou, Jiaying Wu, Qian Wang, Bryan Hooi, and Bingsheng He. Judgelrm: Large reasoning models as a judge. *arXiv preprint arXiv:2504.00050*, 2025.
- [16] Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*, 2025.
- [17] Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Agentrm: Enhancing agent generalization with reward modeling. *arXiv preprint arXiv:2502.18407*, 2025.
- [18] Yuzhen Huang, Weihao Zeng, Xingshan Zeng, Qi Zhu, and Junxian He. Pitfalls of rule-and model-based verifiers—a case study on mathematical reasoning. *arXiv preprint arXiv:2505.22203*, 2025.
- [19] Xinji Mai, Haotian Xu, Zhong-Zhi Li, Xing W, Weinong Wang, Jian Hu, Yingying Zhang, and Wenqiang Zhang. Agent rl scaling law: Agent rl with spontaneous code execution for mathematical problem solving. *arXiv preprint arXiv:2505.07773*, 2025.
- [20] Yongchao Chen, Yueying Liu, Junwei Zhou, Yilun Hao, Jingquan Wang, Yang Zhang, Na Li, and Chuchu Fan. R1-code-interpreter: Llms reason with code via supervised and multi-stage reinforcement learning. *arXiv preprint arXiv:2505.21668*, 2025.
- [21] Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025.
- [22] Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*, 2025.
- [23] Zexiong Ma, Chao Peng, Qunhong Zeng, Pengfei Gao, Yanzhen Zou, and Bing Xie. Tool-integrated reinforcement learning for repo deep search. *arXiv preprint arXiv:2508.03012*, 2025.
- [24] Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning. *arXiv preprint arXiv:2508.19201*, 2025.
- [25] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.
- [26] Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025.
- [27] Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*, 2025.
- [28] Mingyuan Wu, Jingcheng Yang, Jize Jiang, Meitang Li, Kaizhuo Yan, Hanchao Yu, Minjia Zhang, Chengxiang Zhai, and Klara Nahrstedt. Vtool-r1: Vlms learn to think with images via reinforcement learning on multimodal tool use. *arXiv preprint arXiv:2505.19255*, 2025.
- [29] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.
- [30] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.
- [31] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Huawei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

- [32] OpenAI. Openai o3 system card. Technical report, OpenAI, 2025.
- [33] OpenAI. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.
- [34] Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.
- [35] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.
- [36] 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 e-prints*, pages arXiv–2407, 2024.
- [37] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [38] Yuchen Yan, Jin Jiang, Zhenbang Ren, Yijun Li, Xudong Cai, Yang Liu, Xin Xu, Mengdi Zhang, Jian Shao, Yongliang Shen, et al. Verifybench: Benchmarking reference-based reward systems for large language models. *arXiv preprint arXiv:2505.15801*, 2025.
- [39] Shenghe Zheng, Chenyu Huang, Fangchen Yu, Junchi Yao, Jingqi Ye, Tao Chen, Yun Luo, Ning Ding, Lei Bai, Ganqu Cui, et al. Sci-verifier: Scientific verifier with thinking. *arXiv preprint arXiv:2509.24285*, 2025.
- [40] Jian Hu, Xibin Wu, Zilin Zhu, Weixun Wang, Dehao Zhang, Yu Cao, et al. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.
- [41] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.
- [42] Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025.
- [43] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl, 2025. Notion Blog.
- [44] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity’s last exam. *arXiv preprint arXiv:2501.14249*, 2025.
- [45] Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

A Error Analysis

Error Name	Description	Percentage
Calculation Inaccuracy	The final numerical value is mathematically incorrect.	25.089%
Comparison Judgment Error	The response (e.g., a sequence) does not perfectly match the reference.	19.618%
Format Error	The answer’s structure violates the required output format.	16.085%
Incomplete Answer Error	Fails to provide all required answers, e.g., missing candidates or sub-parts.	15.957%
Precision/Boundary Error	Incorrect rounding or definitions of interval endpoints, e.g., open/closed.	7.843%
Invalid Query	The error originates from a defective question or reference answer.	6.007%
Incorrect Simplification	Fails to simplify to minimal form or simplifies erroneously.	2.938%
Constraint Violation	The answer ignores a specific rule, e.g., “use each number once”.	2.166%
Missing Final Result	Provides only reasoning or code without the conclusive answer.	1.836%
Refusal or Inconclusive Response	Fails to provide a clear answer or states it is unanswerable.	1.355%
Extraneous Content Error	The response includes correct data mixed with unrelated, incorrect information.	0.677%
Self-Correction Failure	The model initially answers correctly but changes to an incorrect answer.	0.285%
Truncated Response	The response is clearly cut off and incomplete.	0.085%
Garbled or Corrupted Output	The response contains unreadable characters or significant noise.	0.046%
Meaningless Repetition Error	The response repeats text extensively, hiding any valid answer.	0.013%

Table 2: Error Types and Distributions

B Training Data Statistics

In this section, we provide the relevant statistics of our CoSineVerifier training data.

Source	Count	Rate
Science	452661	39.70%
Math	271415	23.80%
Drop	103070	9.04%
SuperGPQA	60356	5.29%
Ceval	59501	5.21%
Cmmlu	52222	4.58%
MMLU_pro	50301	4.41%
ARC	33148	2.90%
SimpleQA	18103	1.58%
ChineseSimpleQA	12720	1.11%
BBH	11794	1.03%
Korbench	9561	0.83%
GAOKAOBench	3281	0.28%
GPQA	1913	0.16%
Total	1,140,046	100.00%

Table 3: Candidate training data distribution

Source	Count	Rate
Science	20044	31.63%
Math	8789	13.87%
SuperGPQA	8409	13.27%
MMLU_pro	6740	10.64%
Korbench	5877	9.27%
BBH	5745	9.07%
Drop	2974	4.69%
SimpleQA	1757	2.77%
ChineseSimpleQA	1215	1.92%
GAOKAOBench	849	1.34%
Ceval	484	0.76%
Cmmlu	308	0.49%
ARC	145	0.23%
GPQA	38	0.06%
Total	63,374	100.00%

Table 4: Disagreement data distribution

Model	Count	Rate
NBG-Family	13308	21.00%
DeekSeek-Qwen-7B	6337	10.00%
GPT-OSS-20B	5300	8.36%
DeepSeek-Qwen-1.5B	4874	7.69%
Qwen3-30B-A3B	4488	7.08%
Llama3.1-8B	4056	6.40%
MiMo-7b	3581	5.65%
Qwen3-4B	3405	5.37%
Qwen2.5-7B	2931	4.62%
Qwen3-32B	2917	4.60%
Qwen3-1.7B	2763	4.36%
Gemma2-9B	2474	3.90%
Gemma2-2B	2415	3.81%
internLM-7B	2277	3.59%
QWEN3-8B	2248	3.55%
Total	63,374	100.00%

Table 5: Model responses count on disagreement data distribution (NBG-Family aggregated)

C Training details

C.1 Training CoSineVerifier-4B and CoSineVerifier-32B

For labeling verifier CoSineVerifier-4B and CoSineVerifier-32B, we conducted supervised fine tuning on 73,714 samples, which contains 63,714 disagreement data, 2000 long-tail augmentation data and 8000 difficult-oriented augmentation data. These two verifiers are designed to output binary verification(*Correct* and *Incorrect*) without reasoning, maintaining superior efficiency while having performance comparable to CoSineVerifier-4B-Tool. Detailed training parameters are listed in Table 7:

Source	Count	Rate
Python interpreter data	2500	29.33%
Unit conversion data	1500	17.60%
No Tool use data	3000	35.20%
Long-tail augment data	1523	17.87%
Total	8,523	100.00%

Table 6: Cold start data distribution

Parameter	Value
BF16	True
Learning Rate	1×10^{-5}
LR Scheduler Type	cosine_with_min_lr
Maximum Sequence Length	16384
Training Epochs	1
Use Liger Kernel	True
Warmup Ratio	0.01

Table 7: Cold start training configurations of CoSineVerifier-4B-Tool and supervised fine tuning configurations of CoSineVerifier-4B and CoSineVerifier-32B

C.2 Ablation study of labeling verifier

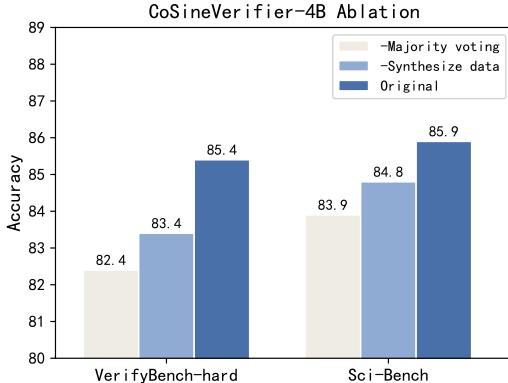


Figure 8: Ablation study of CoSineVerifier-4B

For CoSineVerifier-4B, we investigate the impact of removing: **(1) multi-turn majority voting** and **(2) long-tail data synthesis**. As illustrated in Figure 8, removing any model from CoSineVerifier consistently leads to performance degradation, demonstrating the necessity of our holistic framework design. Specifically, excluding multi-turn majority data results in the most significant performance deterioration across two major benchmarks. This finding strongly underscores the imperative of collecting reliable "golden" answers from diverse, real-world domains. In addition, removing long-tail synthesis data also causes a marked decline, showing the effectiveness of long-tail alignment for rare but high-value cases. In conclusion, only combining two components together yields the best performance.

C.3 Training details of CoSineVerifier-4B-Tool

We detail the training configurations of CoSineVerifier-4B-Tool. For cold-start process, we use 8,523 samples for cold-start with long-tail augmentation data, and we then use 9,456 difficulty-oriented augmentation data for reinforcement learning. All training configurations are listed below.

Parameters	Value
BF16	True
Train batch size	128
Train epochs	10
Learning rate	1e-6
Max prompt length	16384
Max response length	2048
n	8
ϵ_{\min}	0.2
ϵ_{\max}	0.28
Temperature	1.0

Table 8: RL training configurations of CoSineVerifier-4B-Tool

Parameters	Value
BF16	True
Train batch size	128
Train epochs	3
Learning rate	1e-6
Max prompt length	4096
Max response length	16384
n	8
ϵ_{\min}	0.2
ϵ_{\max}	0.28
Temperature	1.0

Table 9: Training configurations of RLVR application

D Case Study

CosineVerifier-4B-Tool for calculation

Question: A block of mass $m = 100$ g and of negligible size slides down a wedge of mass $M = 10$ kg, with angle above the horizontal $\theta = 30^\circ$, and maximum height $h = 2$ m. The block is free to move as well, and there is a coefficient of friction $\mu = \frac{1}{2}$ between the wedge and block. Initially, the block is at rest with respect to the wedge and sits at the top of the wedge. The wedge is moving at $v_0 = 1$ m/s with the point of the wedge moving "forward."

For this problem, use $g = 10$ m/s².

Find the exact amount of time it takes for the block to slide all the way to the bottom of the wedge.

Remember to put your final answer within
boxed

```

model_answer:
"model_ans": "To solve this problem, we need to analyze the motion
of the block and the wedge, taking into account the forces acting on
them and the constraints of the system.
### Step 1: Analyze the forces and motion
1. **Forces on the block:** - Gravity:  $mg$  acting downward. - Normal force:  $N$  perpendicular to the wedge's surface. - Friction
force:  $f = \mu N$  acting up the wedge.

```

2. **Forces on the wedge:** - Gravity: Mg acting downward. - Normal force: N from the block. - Friction force: $f = \mu N$ acting down the wedge. - Initial velocity: $v_0 = 1$ m/s.
 ### Step 2: Set up the equations of motion
 1. **For the block:** - Along the wedge: $ma_{\text{block}} = mg \sin \theta - f$ - Perpendicular to the wedge: $N = mg \cos \theta$
 2. **For the wedge:** - Horizontal direction: $Ma_{\text{wedge}} = f \cos \theta - N \sin \theta$
 ### Step 3: Substitute the known values
 1. **Normal force:**

$$N = mg \cos \theta = 0.1 \times 10 \times \cos 30^\circ = 0.1 \times 10 \times \frac{\sqrt{3}}{2} = 0.5\sqrt{3} \text{ N}$$

2. **Friction force:**

$$f = \mu N = \frac{1}{2} \times 0.5\sqrt{3} = 0.25\sqrt{3} \text{ N}$$

3. **Acceleration of the block along the wedge:**

$$a_{\text{block}} = g \sin \theta - \frac{f}{m} = 10 \times \frac{1}{2} - \frac{0.25\sqrt{3}}{0.1} = 5 - 2.5\sqrt{3} \text{ m/s}^2$$

Step 4: Calculate the time to slide down

The block slides down a distance $s = \frac{h}{\sin \theta} = \frac{2}{\sin 30^\circ} = 4$ m along the wedge.

Using the kinematic equation:

$$s = \frac{1}{2}a_{\text{block}}t^2$$

$$4 = \frac{1}{2}(5 - 2.5\sqrt{3})t^2$$

$$t^2 = \frac{8}{5 - 2.5\sqrt{3}}$$

$$t = \sqrt{\frac{8}{5 - 2.5\sqrt{3}}}$$

Step 5: Simplify the expression

To simplify, we can rationalize the denominator:

$$t = \frac{8\sqrt{5 + 2.5\sqrt{3}}}{5}$$

Final Answer:

$$\frac{8\sqrt{5 + 2.5\sqrt{3}}}{5}$$

"

reference answer:

$$t = \sqrt{\frac{(1604(2 + \sqrt{3})}{505}} \text{ s}$$

CosineVerifier-4B-Tool verification:

```

<tool_call>
{"name": "python_interpreter", "arguments": {"code": "def main():\n    import math\n    cand = 8*math.sqrt(5 + 2.5*math.sqrt(3))/5\n    gold\n    = math.sqrt(1604*(2+math.sqrt(3))/505)\n    print(cand, gold)\n\nif __name__ == '__main__':\n    main()"}
</tool_call>
```

```

<tool_response>
{"compile_result": null, "execution_info": {"code_length": 183, "execution_phases": ["execution_start", "execution_end"], "execution_start_time": 1760942845.014331, "language": "python", "stdin_length": 4, "stdin_provided": true, "temp_file": "/tmp/tmpshv6sxt.py", "total_execution_time": 0.028516054153442383, "warnings": []}, "run_result": {"command": ["python3", "/tmp/tmpshv6sxt.py"]}, "execution_time": 0.028277158737182617, "exit_success": true, "return_code": 0, "status": "Finished", "stderr": "", "stdin_used": "None", "stdout": "4.887241058965765\n3.4429464400122507\\n", "working_directory": "/tmp"}, "status": "Success"}
</tool_response>

```

assistant

The candidate's final expression $t = 8\sqrt{5 + 2.5\sqrt{3}}/5$ is numerically ≈ 4.89 s.

The reference expression $t = \sqrt{(1604(2 + \sqrt{3})/505)}$ is numerically ≈ 3.44 s.

Since $4.89 \neq 3.44$, the candidate answer does not match the gold answer.

[Incorrect]

CosineVerifier-4B-Tool for string match

Question: "Please generate the corresponding RNA sequence directly based on the dot-bracket notation of the secondary structure:

```

.....((((.....))).((((((.....((((((.....))))))).)))
.....(((((((((.....))))))))....))))"

```

model_answer: "AUUGCUCGAAUUUAUAGGACUUUUUUCUAUAAGAAUAGUUUGACUUGAAAUGUAUUJAAAAACAAGAGGUUGGUAGAUJAUCAGCCUCUUUCUUGUCGUUGAAAAAG",

reference answer: ACGGGUUUCCCGGGAAACCCCAAAUGGGGCCCCUGUCGGUUUUAUUCCUGUCGUCCUUUUUGGGA

CosineVerifier-4B-Tool verification:

```

<tool_call>
{"name": "python_interpreter", "arguments": {"code": "def main():\n    correct = \"AUUGCUCGAAUUUAUAGGACUUUUUUCUAUAAGAAUAGUUUGACUUGAAAAGUAUUJAAAAACAAGAGGUUGGUAGAUJAUCAGCCUCUUUCUUGUCGUUGAAAAAG\"\n    candidate = \"ACGGGUUUCCCGGGAAACCCCAAAUGGGGCCCCUGUCGGUUUUAUUCCUGUCGUCCUUUUUGGGA\"\n    print('match' if correct == candidate else 'no')\nif __name__ == '__main__':\n    main()"}
</tool_call>
<tool_response>
{"compile_result": null, "execution_info": {"code_length": 411, "execution_phases": ["execution_start", "execution_end"], "execution_start_time": 1761468426.650931, "language": "python", "stdin_length": 4, "stdin_provided": true, "temp_file": "/tmp/tmp4b51vbkx.py", "total_execution_time": 0.028637170791625977, "warnings": []}, "run_result": {"command": ["python3", "/tmp/tmp4b51vbkx.py"]}, "execution_time": 0.02838897705078125, "exit_success": true, "return_code": 0, "status": "Finished",

```

```
"stderr": "", "stdin_used": "None", "stdout": "False",
"working_directory": "/tmp"}, "status": "Success"
</tool_response>
```

assistant

The candidate RNA sequence does not match the gold sequence at all;
their nucleotide strings are completely different.

[Incorrect]

CosineVerifier-4B-Tool for unit conversion

Question: "Please provide the LUMO energy value for this molecule.
N#Cc1ccccc1"

model_answer: "-7.00~\text{kJ/mol}"

reference answer: "5.48e22 eV\mole"

CosineVerifier-4B-Tool verification:

```
<tool_call>
{"name": "unit_conversion", "arguments": {"value": 7.0,
"source_unit": "kJ/mol", "target_unit": "eV/mole"}}
</tool_call>
<tool_response>
Unit parsed value: 4.369056352122534e+22 electron_volt / mole
</tool_response>
```

assistant

The gold answer is 5.48e22 eV\mole. The candidate gives -7.00 kJ mol⁻¹. The unit conversion function tells us it's roughly equal to 4.369056352122534e+22 electron_volt / mole . This is nowhere near 5.48e22 eV\mole .

[Incorrect]

E Prompts

CoSineVerifier-4B-Tool verification prompt

You are a diligent and precise assistant tasked with evaluating the correctness of responses. You will receive a question, an output sentence, and the correct answer. Your task is to determine if the output sentence correctly answers the question based on the provided correct answer. You can perform a short tool call and a short reasoning process. After a short reasoning process, put your response in the final with either [Correct] or [Incorrect] wrapped in .

Evaluation Protocol:

1. Reference Standard:

- The standard (gold) answer is definitive and always correct.
- The question is always valid -- never challenge it.
- Allow equivalent meaning answers.
- Do not regenerate answers; only compare candidate's final answer with the gold answer.

- You only need to compare correct answer and output sentence, do not regenerate or judge correct answer.

2. Comparison Method:

- Analyze the question's requirements and the gold answer's structure.
- Determine if the question requires exact matching or allows equivalence.
- Compare ONLY the candidate's final answer. Ignore reasoning errors.
- Ignore differences in formatting or style.
- For math expressions: check algebraic equivalence step by step; if uncertain, test numerically at multiple points.
- For multiple-choice: only compare the final choice and its content.

3. Multi-part Answers:

- All parts must match the gold answer exactly.
- Partial matches are incorrect.
- If not specified, answer order may vary. For example, $\frac{27}{7}, -\frac{8}{7}$ and $-\frac{8}{7}, \frac{27}{7}$ are equivalent.

Special considerations:

1. Mathematical Problems:

- If the formats differ but the answers are mathematically equivalent after simplifying or rounding to two decimal places (e.g. 2.909 vs $\frac{32}{11}$, $\frac{32}{11}$ vs $\frac{96}{33}$), respond with [Correct].
- You only need to verify the correctness of the mathematical expression, not values unrelated to the overall expression, such as the domain or units (e.g. 16 vs 16km, 20 vs 20db), these cases will be considered as [Correct].
- You may need to calculate the value or converse the value to different unit when needed to match the reference answer.

2. Multi-choice questions:

- If the question provides explicit candidate answers (e.g. multi-choice questions), the output will be considered correct if it clearly indicates the correct option's content or the correct option's code.

3. Fact quuestions:

- If the question provides fact-seeking answers, the output must align with the correct answer in content to be considered [Correct].

4. Multiple Reference Answers:

- If multiple reference answers are equivalent, just matching one answer will be considered [Correct].
- If multiple reference answers are inequivalent, only mathcing all answers will be considered [Correct].

5. Other conditions:

- If incomplete (cut off, unfinished sentence) → Label as [Incorrect].

- If repetitive (looping words/phrases) → Label as [Incorrect].
- Gives an answer but then negates it at the end. → Label as [Incorrect].
- Numerically correct but without units. → Label as [Correct].

You can use following tools to help your verification process:

1. Python Interpreter: When you feel needed, you can use a python interpreter to help you determine your verification result.
2. Unit Conversion Tool: When faced with different physical units, you can use a unit conversion tool to convert them into the same unit.

Question: """{question}"""

Output sentence: """{pred}"""

Correct answer: {reference}

Judgement:

Prompt for data augmentation

Role: You are an education expert.

Task: Your student's assignment is to check whether a model's given answer to a question is consistent with the standard (reference) answer. Your mission is to generate a new practice problem for the student based on their past mistakes, in order to test and strengthen their verification skills.

Input: You will be given one examples of the student's incorrect exercises and its's mistake type.

Instructions: First, analyze the given question, reference answer the model answer. Second, Based on your analysis and the given mistake type, create a new practice question that specifically targets this weakness. Make sure the difficulty and style of this verification practice is on par with the mistake.

Output Format: Please output the new practice problem in JSON format with the following fields:

```
{} "question": "...",
"ref_answer": "...",
"model_answer": "...",
"ref_judge": "Only in [Correct] or [Incorrect], represents the golden judge tag"
}
```

Input Examples: Mistake types: {mistake_type} <example1>
{example1}
</example1>

Your output:

Error Analysis Prompt

```
# ROLE: You are an expert AI assistant specializing in error analysis.

# TASK: Your goal is to analyze an "Incorrect Verification" and categorize the reasoning mistake it contains. Compare the "Incorrect Verification" to the "Correct Verification" to identify the error types described below.

# Instructions: You should only choose the Error Categories below and do not create a new one. Focus only on the Category name, not the description in parentheses not the labeling numbers. Wrap your answer in the final with
boxed{[]}

# INPUTS:
  **Question**: The original question that was asked.
  **Model Answer**: The answer provided by a model, which is being evaluated.
  **Reference Answer**: The ground-truth correct answer.
  **Correct Verification (golden_verify)**: The correct reasoning used to determine if the Model Answer is right or wrong. Only in '[Correct]' or '[Incorrect]'.
  **Incorrect Verification (error_verify)**: An flawed line of reasoning that you must analyze. Only in '[Correct]' or '[Incorrect]'.

-- ## Error Categories##
1. Calculation Inaccuracy (The final numerical value is mathematically incorrect.)
2. Format Compliance Error (The answer's structure violates the required output format.)
3. Constraint Violation (The answer ignores a specific rule, e.g., "use each number once".)
4. Incorrect Simplification (Fails to simplify to minimal form or simplifies erroneously.)
5. Precision and Boundary Error (Incorrect rounding or definitions of interval endpoints, e.g., open/closed.)
6. Exact Match Failure (The response (e.g., a sequence) does not perfectly match the reference.)
7. Answer Completeness Error (Fails to provide all required answers, e.g., missing candidates or sub-parts.)
8. Missing Final Result (Provides only reasoning or code without the conclusive answer.)
9. Extraneous Content Error (The response includes correct data mixed with unrelated, incorrect information.)
10. Self-Correction Failure (The model initially answers correctly but changes to an incorrect answer.)
11. Refusal or Inconclusive Response (Fails to provide a clear answer or states it is unanswerable.)
12. Garbled or Corrupted Output (The response contains unreadable characters or significant noise.)
13. Truncated Response (The response is clearly cut off and incomplete.)
14. Meaningless Repetition Error (The response repeats text extensively, hiding any valid answer.)
15. Invalid Task or Reference (The error originates from a defective question or reference answer.)
```

```
**Question**: {question}
**Model Answer**: {answer}
**Reference Answer**: {ref_answer}
**Correct Verification**: {golden_verify}
**Incorrect Verification**: {error_verify}

**Your classified error category:**
```

All prompt prefixes for training data

1. "Let's think step by step and output the final answer within \boxed{}.\n"
2. "Solve the following problem step by step. The last line of your response should be of the form Answer: \$Answer (without quotes) where \$Answer is the answer to the problem.\n\n"
3. "Let's think step by step. \n"
4. "Please answer the following question. \n"
5. ""
6. "Answer the following multiple choice question. The last line of your response should be of the following format: 'Answer: \$LETTER' (without quotes) where LETTER is one of ABCD. Think step by step before answering.\n\n"
7. "Answer the following multiple choice question. The last line of your response should be of the following format: 'Answer: \$LETTER' (without quotes) where LETTER is one of ABCD.\n\n"