Demystifying Long Chain-of-Thought Reasoning

Shiming Yang [*1 2] **Yuxuan Tong** [*3] **Xinyao Niu** [1] **Graham Neubig** [4] **Xiang Yue** [*4]

Abstract

Scaling inference compute has become a key driver of advanced reasoning in large language models (LLMs). A proven approach for scaling inference compute is to generate long chains-of-thought (CoTs), enabling models to engage in structured reasoning strategies such as backtracking and error correction. Reinforcement learning (RL) has emerged as a crucial method for developing these capabilities, yet the conditions under which long CoTs emerge remain unclear, and RL training requires careful design choices. In this study, we systematically investigate the underlying *mechanics of long CoT reasoning*—examining the factors that enable models to generate extended reasoning trajectories. Through extensive supervised fine-tuning (SFT) and RL experiments, we identify three key findings: 1) while SFT is not strictly necessary, it significantly simplifies training and improves efficiency; 2) reasoning capabilities tend to emerge with increased training compute but are not guaranteed, making reward shaping essential for stabilizing CoT length growth; and 3) scaling verifiable reward signals is critical for RL, and we find that leveraging noisy, web-extracted solutions with filtering mechanisms shows promising potential, particularly in out-of-distribution (OOD) reasoning tasks such as STEM problem-solving. These insights provide practical guidance for optimizing training strategies to enhance long CoT reasoning in LLMs. Our code is available at: https://github.com/eddycmu/demystify-long-cot.

[*]Project Lead [1]IN.AI [2]English Name: Edward Yeo [3]Tsinghua University. Work started when interning at CMU. [4]Carnegie Mellon University. Correspondence to: Xiang Yue <xyue2@andrew.cmu.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

1. Introduction

Large language models (LLMs) (Brown et al., 2020; Touvron et al., 2023; Chowdhery et al., 2023; Anthropic, 2023; OpenAI, 2023) have demonstrated remarkable reasoning in domains like mathematics (Cobbe et al., 2021) and programming (Chen et al., 2021). A key technique for enabling reasoning in LLMs is chain-of-thought (CoT) prompting (Wei et al., 2022), which guides models to generate intermediate reasoning steps before arriving at a final answer.

Despite these advancements, LLMs still struggle with highly complex reasoning tasks, such as mathematical competitions (Hendrycks et al., 2021), PhD-level scientific QA (Rein et al., 2024), and software engineering (Jimenez et al., 2024), even with CoT. Recently, OpenAI's o1 models (OpenAI, 2024) have demonstrated significant breakthroughs on these tasks. A key distinguishing feature of these models is their ability to scale up inference compute while refining CoT strategies—such as recognizing and correcting mistakes, breaking down difficult steps, and iterating on alternative approaches—leading to substantially longer and more structured reasoning processes.

Several efforts have attempted to replicate the performance of o1 models by training LLMs to generate long CoTs (Qwen Team, 2024b; DeepSeek-AI, 2025; Kimi Team, 2025; Pan et al., 2025; Zeng et al., 2025). Most of these approaches rely on verifiable rewards, such as accuracy based on ground-truth answers, to stabilize reinforcement learning (RL) at scale. However, a comprehensive understanding of how models learn and generate long CoTs remains limited. In this work, we systematically investigate the underlying mechanics of long CoT generation. Specifically, we explore:

1) *Supervised fine-tuning (SFT) for long CoTs* – the most direct way to enable long CoT reasoning. We analyze its scaling behavior and impact on RL, finding that long CoT SFT allows models to reach higher performance and also facilitates easier further RL improvements than short CoT.

2) *Challenges in RL-driven CoT scaling* – we observe that RL does not always stably extend CoT length and complexity. To address this, we introduce a cosine length-scaling reward with a repetition penalty, which stabilizes CoT growth while encouraging emergent reasoning behaviors such as branching and backtracking.

3) *Scaling up verifiable signals for long CoT RL* – Verifiable reward signals are essential for stabilizing long CoT RL. However, scaling up them remains challenging due to the limited availability of high-quality, verifiable data. To address this, we explore the use of data containing noisy, web-extracted solutions (Yue et al., 2024). While these "silver" supervision signals introduce uncertainty, we find that, with appropriate mixture in SFT and filtration in RL, they show promise, especially in out-of-distribution (OOD) reasoning scenarios such as STEM problem solving.

2. Problem Formulation

In this section, we define the notation, followed by an overview of SFT and RL methods for eliciting long CoTs.

Research Aim

Our primary aim is to investigate the *mechanics* of *long chain-of-thought reasoning* in LLMs. By systematically analyzing the factors that influence long CoT reasoning, we distill key insights from rigorous ablations and offer practical takeaways for enhancing and stabilizing long CoT performance.

2.1. Notation

Let $x = (x_1, x_2, \ldots, x_n)$ be a query, and let $y = (y_1, y_2, \ldots, y_m)$ be the corresponding output sequence. We consider a LLM parameterized by θ, which defines a conditional distribution over output tokens: $\pi_\theta(y_t \mid x, y_{1:t-1})$.

We denote by $\text{CoT}(y) \subseteq y$ the tokens in the generated output that constitute the *chain-of-thought*, which is often a reasoning trace or explanatory sequence. The final "answer" can be a separate set of tokens or simply the last part of y.

In this work, we use the term *long chain-of-thought* (*long CoT*) to describe an extended sequence of reasoning tokens that not only exhibits a larger-than-usual token length but also demonstrates more sophisticated behaviors such as:

1) Branching and Backtracking: The model systematically explores multiple paths (branching) and reverts to earlier points if a particular path proves wrong (backtracking).

2) Error Correction: The model detects inconsistencies or mistakes in its intermediate steps and takes corrective actions to restore coherence and accuracy.

2.2. Supervised Fine-Tuning (SFT)

A common practice is to initialize the policy π_θ via SFT (Lamb et al., 2016) on a dataset $\mathcal{D}_{\text{SFT}} = \{(x_i, y_i)\}_{i=1}^N$, where y_i can be normal or long CoT reasoning tokens.

2.3. Reinforcement Learning (RL)

After optional SFT initialization, we frame the generation of a long CoT as a Markov Decision-making Process (MDP) and optimize its reward with reinforcement learning.

Reward Function. We define a scalar reward r_t designed to encourage correct and verifiable reasoning. We only consider the outcome-based reward for the final answer produced, and do not consider process-based reward for the intermediate steps. We denote the term $r_{\text{answer}}(y)$ to capture the correctness of the final solution.

Policy Update. Policy gradient methods such as Proximal Policy Optimization (PPO) (Schulman et al., 2017) and REINFORCE (Sutton & Barto, 2018) are employed to iteratively update θ. The resulting updates push the policy to generate tokens that yield higher rewards, potentially favoring longer or more complex reasoning traces.

2.4. Training Setup

We adopt `Llama-3.1-8B` (Meta, 2024) and `Qwen2.5-7B-Math` (Qwen Team, 2024a) as the base models. For both SFT and RL, we use the 7,500-sample prompt set of MATH (Hendrycks et al., 2021) training split by default, with which verifiable ground truth answers are provided. For SFT when ground truth answers are available, we synthesize responses by rejection sampling (Zelikman et al., 2022; Dong et al., 2023; Yuan et al., 2023; Gulcehre et al., 2023; Singh et al., 2023; Tong et al., 2024). Specifically, we first sample a fixed number N of candidate responses per prompt and then filter by only retaining ones with final answers consistent with the corresponding ground truth answers. We also discuss data like WebInstruct (Yue et al., 2024) that is more diverse but without gold supervision signals like ground truth answers in §5. We train the models with the OpenRLHF framework (Hu et al., 2024).

2.5. Evaluation

We focus on four representative reasoning benchmarks: MATH-500, AIME 2024, TheoremQA (Chen et al., 2023), and MMLU-Pro-1k (Wang et al., 2024a). Given that our training data is primarily in the mathematical domain, these benchmarks provide a comprehensive framework for both in-domain and out-of-domain evaluations. By default, we evaluate the models using a temperature of $t = 0.7$, a top-p value of 0.95, and a maximum output length of 16,384 tokens. Please refer to Appendix E.1 for further details on the evaluation setup.

3. Impact of SFT for long CoT

In this section, we compare long CoT and short CoT as different SFT data patterns and different RL initializations.

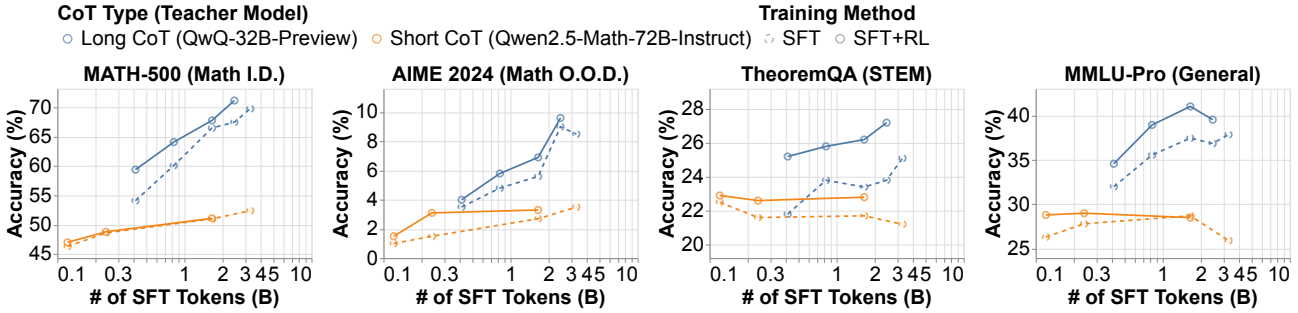


Figure 1. Scaling curves of post-training `Llama-3.1-8B` with long CoT and short CoTs. SFT with long CoTs can scale up to a higher upper limit and has more potential to further improve with RL.

3.1. SFT Scaling

To compare long CoT with short CoT, the first step is to equip the model with the corresponding behavior. The most straightforward approach is to fine-tune the base model on CoT data. Since short CoT is common, curating SFT data for it is relatively simple via rejection sampling from existing models. However, how to obtain high-quality long CoT data remains an open question. We begin by distilling from the open-weight `QwQ-32B-Preview` (Qwen Team, 2024b) because it is cheaper and produce better performance, which will be further discussed in §3.3.

Setup. To curate the SFT data, for long CoT, we distill from `QwQ-32B-Preview`; for short CoT, we distill from `Qwen2.5-Math-72B-Instruct` (Qwen Team, 2024a), which is a short CoT model at the SOTA level in math reasoning. Specifically, for long CoT, we perform rejection sampling with $N \in \{32, 64, 128, 192, 256\}$ candidate samples, while for short CoT, we use $N \in \{32, 64, 128, 256\}$. Here, we control SFT datasets with smaller N's as subsets of ones with larger N's. In each case, the number of SFT tokens is proportional to N. For fairness, we use the same base model `Llama-3.1-8B` (Meta, 2024). Please refer to Appendix E.3 for more details about the SFT setup.

Result. Dashed lines in Figure 1 show that as we scale up the SFT tokens, long CoT SFT keeps improving models' accuracies, while short CoT SFT saturates early at a lower accuracy level. For example, on MATH-500, long CoT SFT scales up to over 70% accuracy and still has not saturated yet, while short CoT converges under 55% accuracy and increasing SFT tokens from around 0.25B to around 1.5B only improves the accuracy by about 3% absolutely.

Takeaway 3.1 for SFT Scaling Upper Limit

SFT with long CoT can scale up to a higher performance upper limit than short CoT. (Figure 1)

3.2. SFT Initialization for RL

Since RL is reported to have a higher upper limit than SFT, we compare long CoT and short CoT as different SFT initialization approaches for RL.

Setup. We initialize RL using SFT checkpoints from §3.1, and train for four epochs, sampling four responses per prompt. Our approach employs PPO (Schulman et al., 2017) with a rule-based verifier from the MATH dataset, using its training split as our RL prompt set. We adopt our cosine length scaling reward with the repetition penalty, which will be detailed in §4. Please refer to Appendix E.4 for more details on the RL setups.

Result. The gap between solid and dashed lines in Figure 1 shows that models initialized with long CoT SFT can usually be further significantly improved by RL, while models initialized with short CoT SFT see little gains from RL. For example, on MATH-500, RL can improve long CoT SFT models by over 3% absolutely, while short CoT SFT models have almost the same accuracies before and after RL.

Takeaway 3.2 for SFT Initialization for RL

SFT with long CoTs makes further RL improvement easier, while short CoTs do not. (Figure 1)

3.3. Sources of Long CoT SFT Data

To curate long CoT data, we compare two synthesis approaches: 1) **Construct** long CoT trajectories by prompting short CoT models to generate primitive actions and sequentially join them. 2) **Distill** long CoT trajectories from existing long CoT models that exhibit emergent long CoT patterns.

Setup. To construct long CoT trajectories, we developed an Action Prompting framework (Appendix E.8) which defined the following primitive actions: `clarify`, `decompose`, `solution_step`, `reflection`, and `answer`. We employed multi-step prompting with a short CoT model (e.g.,

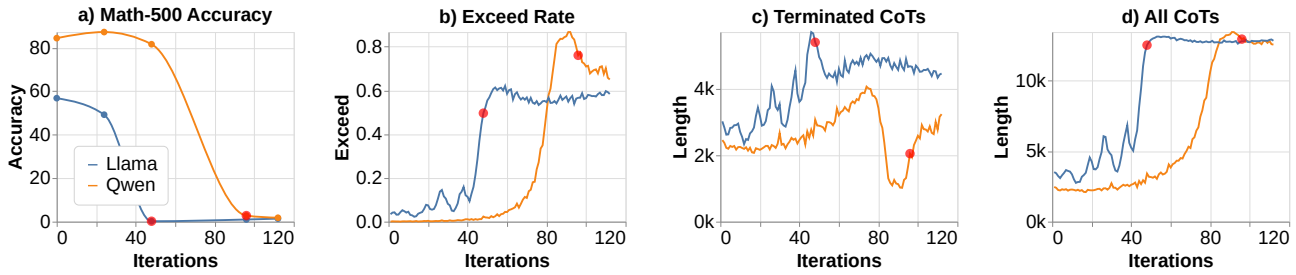


Figure 2. Both fine-tuned `Llama3.1-8B` and `Qwen2.5-Math-7B` models trained under RL with the Classic Reward manifested emergent CoT length scaling past the context window size, resulting in deterioration of Math-500 accuracy. The red points on the charts correspond to the iteration where the accuracy dropped to near zero.

`Qwen2.5-72B-Instruct`) to sequence these actions, while a stronger model, `o1-mini-0912`, generates reflection steps incorporating self-correction. For distilling long CoT trajectories, we use `QwQ-32-Preview` as the teacher model. In both approaches, we adopt the MATH training set as the prompt set and apply rejection sampling. To ensure fairness, we use the same base model (`Llama-3.1-8B`), maintain approximately 200k SFT samples, and use the same RL setup as in §3.2.

Result. Table 1 shows that distilled data with an emergent long CoT pattern generalizes better than the constructed pattern, and can be further significantly improved with RL, while the constructed pattern cannot. Models trained with the emergent long CoT pattern achieve significantly higher accuracies on OOD benchmarks AIME 2024 and MMLU-Pro-1k, improving by 15-50% relatively. Besides, on the OOD benchmark TheoremQA, RL on the long CoT SFT model significantly improve its accuracy by around 20% relatively, while the short CoT model's performance does not change. This is also why we conduct most of our experiments based on distilled long CoT trajectories.

Takeaway 3.3 for Long CoT Cold Start

SFT initialization matters: high-quality, emergent long CoT pattern leads to significantly better generalization and RL gains. (Table 1)

Table 1. Emergent long CoT patterns outperform constructed ones. All the models here are fine-tuned from the base model `Llama-3.1-8B` with the MATH training prompt set.

Training Method	Long CoT SFT Pattern	MATH 500	AIME 2024	Theo. QA	MMLU Pro-1k
SFT	Constructed	48.2	2.9	21.0	18.1
	Emergent	**54.1**	**3.5**	**21.8**	**32.0**
SFT+RL	Constructed	52.4	2.7	21.0	19.2
	Emergent	**59.4**	**4.0**	**25.2**	**34.6**

4. Impact of Reward Design on Long CoT

This section examines reward function design, with a focus on its influence on CoT length and model performance.

4.1. CoT Length Stability

Recent studies on long CoT (DeepSeek-AI, 2025; Kimi Team, 2025; Hou et al., 2025) suggest that models naturally improve in reasoning tasks with increased thinking time. Our experiments confirm that models fine-tuned on long CoT distilled from `QwQ-32B-Preview` tend to extend CoT length under RL training, albeit sometimes unstably. This instability, also noted by (Kimi Team, 2025; Hou et al., 2025), has been addressed using techniques such as length and repetition penalties to stabilize training.

Setup. We adopt the same RL setup as §3.2. We used a 16K context window size with two different models fine-tuned on long CoT data distilled from `QwQ-32B-Preview` using the MATH train split. The models were `Llama3.1-8B` and `Qwen2.5-Math-7B`. More details can be found in Appendix E.5.1.

Results. We observed that both models increased their CoT length as they scaled, eventually reaching the context window limit. This led to a decline in training accuracy due to CoTs exceeding the allowable window size. Additionally, different base models exhibited distinct scaling behaviors. The weaker `Llama-3.1-8B` model showed greater fluctuations in CoT length compared to `Qwen-2.5-Math-7B`, as illustrated in Figure 2.

We also found that the rate at which CoTs exceeded the context window was correlated with response length (Figure 2). This suggests that feedback from exceeding the limit applied downward pressure on the overall trajectory distribution rather than affecting only a few isolated CoTs. Notably, a model might be penalized even without an explicit length-exceedance penalty due to reward or advantage normalization, both of which are standard in RL frameworks.

Figure 3. The Classic and Cosine Reward functions. The Cosine Reward varies with generation length.

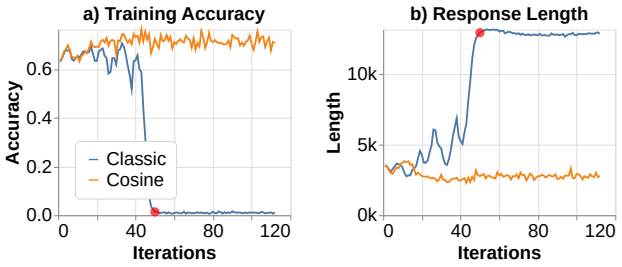


Figure 4. `Llama3.1-8B` trained with length shaping using the Cosine Reward exhibited more stable (a) training accuracy and (b) response length. This stability led to improved performance on downstream tasks (Figure 5). Red points on the charts indicate iterations where training accuracy dropped to near zero.

Takeaway 4.1 for CoT Length Stability

CoT length does not always scale up in a stable fashion. (Figure 2)

4.2. Active Scaling of CoT Length

We found that reward shaping can be used to stabilize emergent length scaling. We designed a reward function to use CoT length as an additional input and to observe a few ordering constraints. Firstly, correct CoTs receive higher rewards than wrong CoTs. Secondly, shorter correct CoTs receive higher rewards than longer correct CoTs, which incentivizes the model to use inference compute efficiently. Thirdly, shorter wrong CoTs should receive higher penalties than longer wrong CoTs. This encourages the model to extend its thinking time if it is less likely to get the correct answer.

We found it convenient to use a piecewise cosine function, which is easy to tune and smooth. We refer to this reward function as the *Cosine Reward*, visualized in Figure 3. This is a *sparse* reward, only awarded once at the end of the CoT based on the correctness of the answer. The formula of **CosFn** can be found in equation 1 in the appendix.

$$R(C, L_{\text{gen}}) = \begin{cases} \text{CosFn}(L_{\text{gen}}, L_{\max}, r_0^c, r_L^c), & \text{if } C = 1, \\ \text{CosFn}(L_{\text{gen}}, L_{\max}, r_0^w, r_L^w), & \text{if } C = 0, \\ r_e, & \text{if } L_{\text{gen}} = L_{\max}. \end{cases}$$

Hyperparameters:

r_0^c / r_0^w : Reward (correct/wrong) for $L_{\text{gen}} = 0$,
r_L^c / r_L^w : Reward (correct/wrong) for $L_{\text{gen}} = L_{\max}$,
r_e : Exceed length penalty,

Inputs:

C : Correctness (0 or 1),
L_{gen} : Generation length.

Setup. We ran experiments with the Classic Reward and the Cosine Reward. We used the `Llama3.1-8B` fine-tuned on long CoT data distilled from `QwQ-32B-Preview` using the MATH train split, as our starting point. For more details, see Appendix E.5.2.

Result. We found that the Cosine Reward significantly stabilized the length scaling behavior of the models under RL, thereby also stabilizing the training accuracy and improving RL efficiency (Figure 4). We also observed improvements in model performance on downstream tasks (Figure 5).

Takeaway 4.2 for Active Scaling of CoT Length

Reward shaping can be used to stabilize and control CoT length while improving accuracy. (Figure 4, 5)

4.3. Length Scaling at Different Model Sizes

We evaluate the effectiveness of active scaling of CoT length at different model sizes.

Setup. We ran experiments with the Classic Reward and the Cosine Reward on both `Qwen-2.5-32B` and `Qwen-2.5-1.5B`. To increase the diversity of scenarios, we used the base variant of the 32B model without any fine-tuning, but used the 1.5B model after fine-tuning on long CoT data distilled from `QwQ-32B-Preview` using the MATH train split. For more details on the training setup, see Appendix E.5.3.

Result. We observed that the Cosine Reward stabilized the length scaling under RL and improved the training accuracy (Figure 6). It resulted in better performance on downstream tasks at both model sizes (Table 2 and Figure 7).

Table 2. `Qwen-2.5-32B` trained under RL directly from the base model without SFT. The three reward types used are the *Classic Reward*, *Cosine Reward*, and the *Cosine Reward* with repetition penalty. The scores are shown for steps 40 and 80.

	Classic		Cosine		Cosine + Rep	
Task	40	80	40	80	40	80
AIME 2024	15.0	16.9	18.5	20.4	**19.2**	**20.6**
MATH-500	78.8	79.6	**81.1**	**81.9**	80.1	81.8
TheoremQA	35.4	36.6	36.8	38.0	**37.7**	**39.1**
MMLU-Pro-1k	45.6	47.1	**48.9**	**47.1**	47.3	45.5
Average	43.7	45.1	**46.3**	**46.9**	46.1	46.8

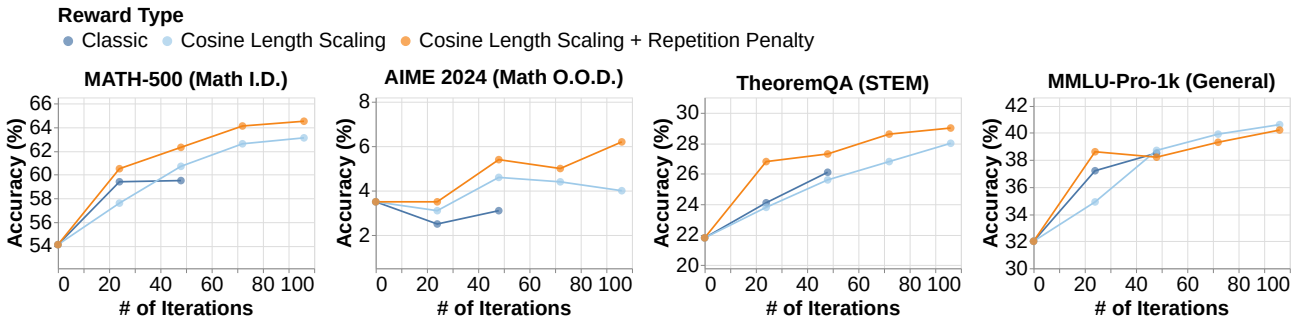


Figure 5. Performance of models trained with different reward functions on a variety of evaluation benchmarks.

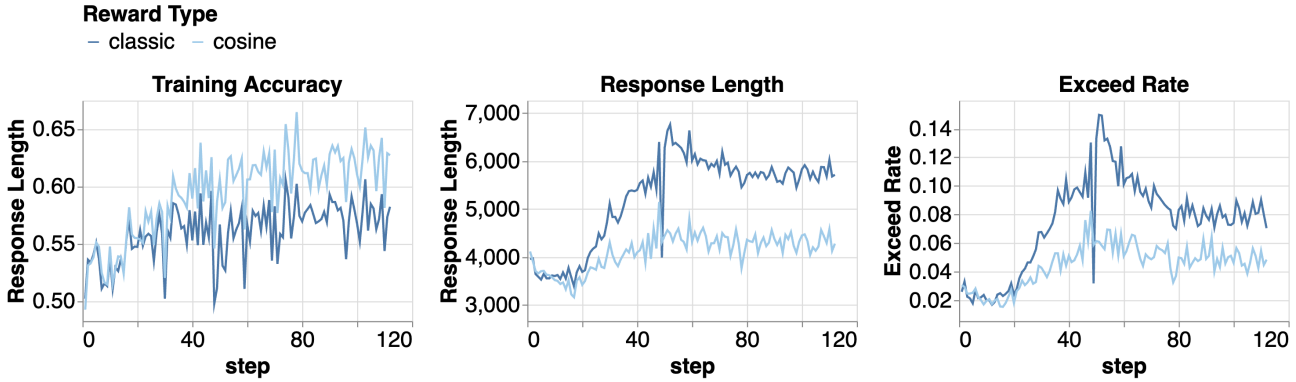


Figure 6. `Qwen-2.5-1.5B` trained with length shaping using the Cosine Reward resulted in greater stability in response length and better training accuracy.

Takeaway 4.3 for Length Scaling Different Model Sizes

Active length scaling is effective at different model sizes. (Table 2 and Figure 7)

Takeaway 4.4 for Cosine Reward Hyperparameters

The Cosine Reward can be tuned to incentivize different kinds of length scaling behaviors. (Figure 9)

4.4. Cosine Reward Hyperparameters

The Cosine Reward hyperparameters can be tuned to shape CoT length in different ways.

Setup. We set up RL experiments with the same model fine-tuned on long CoT distilled from `QwQ-32B-Preview`, but with different hyperparameters for the Cosine Reward function. We tweaked the correct and wrong rewards $r_0^c, r_L^c, r_0^w, r_L^w$ and observed their impact on the CoT lengths. For more details, see Appendix E.5.4.

Result. We see from Figure 9 in the Appendix that if the reward for a correct answer increases with CoT length ($r_0^c < r_L^c$), the CoT length increases explosively. We also see that the lower the correct reward relative to the wrong reward, the longer the CoT length. We interpret this as a kind of trained risk aversion, where the ratio of the correct and wrong rewards determines how confident the model has to be about an answer for it to derive a positive expected value from terminating its CoT with an answer.

4.5. Context Window Size

We know that longer contexts give a model more room to explore, and with more training samples, the model eventually learns to utilize more of the context window. This raises an interesting question – are more training samples necessary to learn to utilize a larger context window?

Setup. We set up 3 experiments using the same starting model fine-tuned on long CoT data distilled from `QwQ-32B-Preview` with the MATH train split. We also used the latter as our RL prompt set. Each ablation used the Cosine Reward and repetition penalty with a different context window size (4K, 8K, and 16K). For more details, see Appendix E.5.5.

Result. We found that the model with a context window size of 8K performed better than the model with 4K, as expected. However, we observed performance was better under 8K than 16K. Note that all three experiments used the

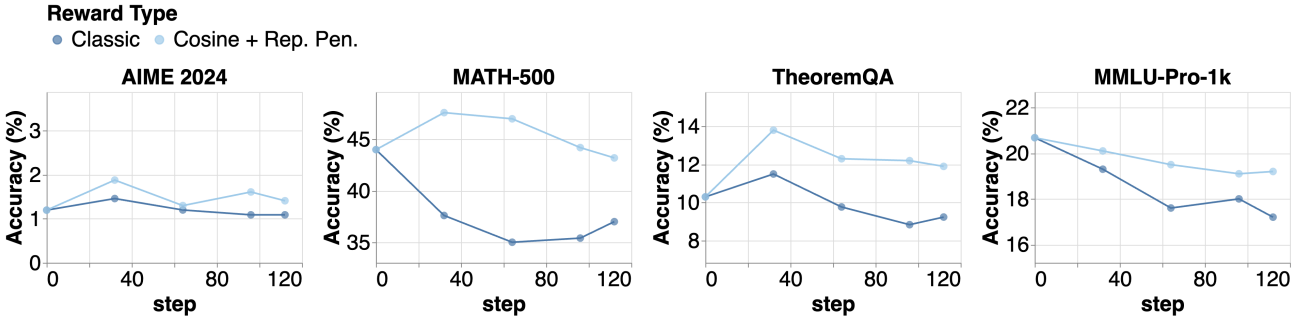


Figure 7. `Qwen-2.5-1.5B` trained with length shaping using the Cosine Reward resulted in better performance on downstream tasks.

same number of training samples (Figure 8). We see this as an indication that models need more training compute to learn to fully utilize longer context window sizes, which is consistent with the findings of (Hou et al., 2025).

Takeaway 4.5 for Context Window Size

Models might need more training samples to learn to utilize larger context window sizes. (Figure 8)

4.6. Length Reward Hacking

We observed that with enough training compute, the model started to show signs of reward hacking, where it increased the lengths of its CoTs on hard questions using repetition rather than learning to solve them. We also noted a fall in the branching frequency of the model, which we estimated by counting the number of times the pivot keyword "`alternatively,`" appeared in the CoT (Figure 10).

We mitigated this by implementing a simple N-gram repetition penalty (Algorithm 1). We observed that the penalty was most effectively applied on repeated tokens, rather than as a sparse reward for the entire trajectory. Similarly, we found that discounting the repetition penalty when calculating the return was effective. Specific feedback about where the repetition occurred presumably made it easier for the model to learn not to do it (see more in §4.7).

Setup. We used the `Llama3.1-8B` model fine-tuned on long CoT data distilled from `QwQ-32B-Preview`. We ran two RL training runs, both using the Cosine Reward, but with and without the repetition penalty. For more details, please refer to Appendix E.5.6.

Result. The repetition penalty resulted in better downstream task performance and also shorter CoTs, meaning there was better utilization of inference compute (Figure 5).

Observation. Our experiments revealed a relationship between the repetition penalty, training accuracy, and the Cosine Reward. When training accuracy was low, the Cosine Reward exerted greater upward pressure on CoT length, leading to increased reward hacking through repetition. This, in turn, required a stronger repetition penalty. Future work could further investigate these interactions and explore dynamic tuning methods for better optimization.

Takeaway 4.6 for Length Reward Hacking

Length rewards will be hacked with enough compute (Figure 10), but this can be mitigated using a repetition penalty. (Figure 5)

4.7. Optimal Discount Factors

We hypothesized that applying the repetition penalty with temporal locality (i.e., a low discount factor) would be most effective, as it provides a stronger learning signal about the specific offending tokens. However, we also observed performance degradation when the discount factor for the correctness (cosine) reward was too low.

To optimally tune both reward types, we modified the GAE formula in PPO to accommodate multiple reward types, each with its own discount factor γ: $\hat{A}_t = \sum_{l=0}^{L} \sum_{m}^{M} \gamma_m^l r_{m,t+l} - V(s_t)$. For simplicity, we set $\lambda = 1$, which proved effective, though we did not extensively tune this parameter.

Setup. We ran multiple RL experiments with the same `Llama3.1-8B` model fine-tuned on `QwQ-32B-Preview` distilled long CoT data. We used the Cosine Reward and repetition penalty but with different combinations of discount factors. For more details, please see Appendix E.5.7.

Result. A lower discount factor is effective for the repetition penalty. A higher discount factor is effective for the correctness reward and the exceed length penalty, which allowed the model attempts at finding a solution earlier in the CoT to be adequately rewarded for a correct answer (Figure 6).

We observed a rather interesting phenomenon where de-

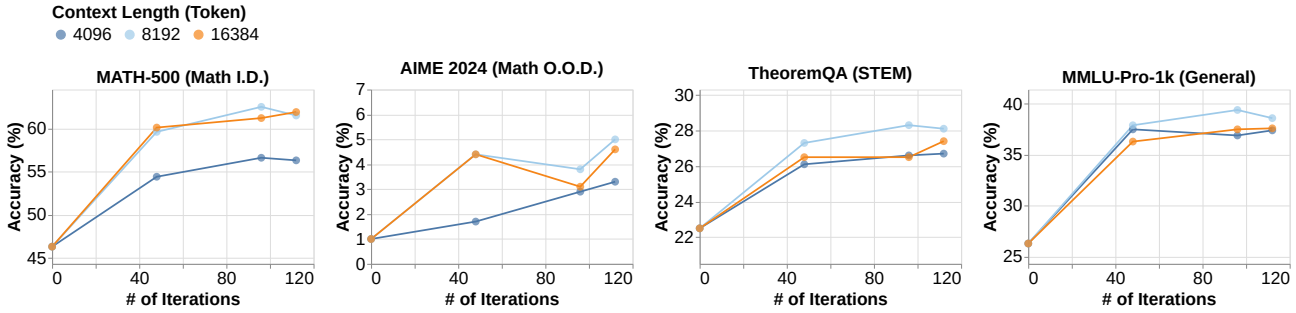


Figure 8. Performance of models trained with different context window sizes. All experiments used the same number of training samples.

creasing the discount factor γ of the correctness (cosine) reward increased the branching frequency in the model's CoT, making the model quickly give up on approaches that did not seem to lead to a correct answer immediately (Figure 11, Extract in Appendix D). We hypothesize that this short-term thinking was due to a relatively small number of tokens preceding the correct answer receiving rewards, which means stepping stones to the right answer are undervalued. Such behavior degraded performance (Figure 6). However, we think this qualitative result might be of potential interest to the research community, due to its similarity to the relationship between behaviors like delayed gratification and the distribution of rewards given to the biological brain (Gao et al., 2021).

Takeaway 4.7 for Optimal Discount Factors

Different kinds of rewards and penalties have different optimal discount factors. (Figure 6)

5. Scale up Verifiable Reward

Verifiable reward signals like ones based on ground-truth answers are essential for stabilizing long CoT RL. However, it is difficult to scale up such data due to the limited availability of high-quality human-annotated verifiable data. As an attempt to counter this, we explore using other data that is more available despite more noise, like QA pairs extracted from web corpora. Specifically, we experiment with the WebInstruct dataset (Yue et al., 2024). For efficiency, we construct WebInstruct-462k, a deduplicated subset derived via MinHash (Broder et al., 1998).

5.1. SFT with Noisy Verifiable Data

We first explore adding such diverse data to SFT. Intuitively, despite less reliable supervision signals, diverse data might facilitate the model's exploration during RL.

Setup. We experiment with three setups, varying the proportion of data without gold supervision signals: 0%, 100%, and approximately 50%. We conduct long CoT SFT by distilling from `QwQ-32B-Preview`. For data with gold supervision signals (MATH), ground truth answers are used for rejection sampling. In contrast, for data from WebInstruct without fully reliable supervision signals but with a much larger scale, we sample one response per prompt from the teacher model without filtration. For RL here, we adopt the same setup as in §3.2, using the MATH training prompt set.

Result. Table 3 shows that incorporating silver-supervised data improves average performance. Adding WebInstruct data to long CoT SFT yields a substantial 5–10% absolute accuracy gain on MMLU-Pro-1k over using MATH alone. Furthermore, mixing MATH and WebInstruct data achieves the best average accuracy across benchmarks.

Table 3. Adding data with a silver supervision signal is beneficial. "WebIT" is the abbreviation of WebInstruct.

Long CoT SFT Data	Training Method	MATH 500	AIME 2024	Theo. QA	MMLU Pro-1k	AVG
100% MATH	SFT	54.1	3.5	21.8	32.0	27.9
	SFT + RL	**59.4**	4.0	**25.2**	34.6	30.8
100% WebIT	SFT	41.2	0.8	21.9	41.1	26.3
	SFT + RL	44.6	1.9	22.5	**43.3**	28.1
50% MATH + 50% WebIT	SFT	53.6	**4.4**	23.5	41.7	30.8
	SFT + RL	57.3	3.8	25.1	42.0	**32.1**

Takeaway 5.1 for SFT with Noisy Verifiable Data

Adding noisy yet diverse data to SFT facilitates balanced performance across different tasks. (Table 3)

5.2. Scale up RL with Noisy Verifiable Data

We compare two main approaches to obtain rewards from noisy verifiable data: 1) the rule-based verifier that is usually accurate but limited in answer forms; 2) the model-based verifier capable of processing free-form responses but susceptible to hacking.

Setup. We implement the model-based verifier by prompting `Qwen2.5-Math-7B-Instruct` with the raw reference solution. To curate SFT data, we prompt `Llama-3.1-8B-Instruct` to extract short-

form answers and apply rejection sampling with `QwQ-32B-Preview`. Specifically, we generate two responses per prompt from WebInstruct-462k and discard cases where neither response aligns with the extracted reference answers. This process yields approximately 189k responses across 115k unique prompts. We perform SFT from `Llama-3.1-8B` on the filtered dataset as initialization for RL. We compare the two types of verifiers with two prompt setups: (1) the 462k full set, which contains prompts asking for free-form answers; (2) the 115k subset used in SFT, which is filtered for prompts asking for short-form answers by the rejection sampling process. Note that we adopt different RL hyperparamters for these two prompt setups. For further details on the model-based verifier, answer extraction and RL hyperparameters, please refer to Appendix E.6 & E.7 & E.5.8 respectively.

Table 4. Performance of RL with different verifiers on unfiltered noisy verifiable data.

Verifier Type	MATH 500	AIME 2024	Theo. QA	MMLU Pro-1k
SFT Initialization	46.6	1.0	23.0	28.3
Rule-Based	45.4	3.3	25.9	35.1
Model-Based	**47.9**	**3.5**	**26.2**	**40.4**

Table 5. Performance of RL with different verifiers on filtered noisy veriable data. The "MATH Baseline" is the model trained with SFT and RL on MATH only in Table 3.

Verifier Type	MATH 500	AIME 2024	Theo. QA	MMLU Pro-1k
MATH Baseline	59.4	4.0	25.2	34.6
SFT Initialization	46.6	1.0	23.0	28.3
Rule-Based	**49.3**	**2.7**	**27.8**	**43.5**
Model-Based	47.7	2.3	26.4	42.2

Result. Table 4 & 5 show that, the model-based verifier performs better on the unfiltered WebInstruct-462k prompt set, while the rule-based verifier performs better on the filtered 115k subset. The disadvantage of rule-based verifier on unfiltered data might be caused by low training accuracy, while the model-based verifier can achieve higher training accuracy, as shown in Figure 13. Moreover, compared to the model trained on human-annotated verifiable data (MATH), leveraging noisy yet diverse verifiable data with rule-based verifier after filtration significantly boosts performance on OOD benchmarks, with absolute gains of up to 2.6% on TheoremQA and 8.9% on MMLU-Pro-1k.

Takeaway 5.2 for RL with Noisy Verifiable Data

To utilize noisy verifiable data in RL, the model-based verifier performs better on unfiltered data, while the rule-based verifier performs better with appropriate filtration for prompts. (Table 4 & 5)

6. Other Factors that Impact Long CoT

6.1. RL Infrastructure for long CoT is still in its infancy

We observe that open-source RL frameworks (e.g., OpenRLHF (Hu et al., 2024) tend to orchestrate multiple systems optimized for different training and inference workloads, and that this often results in multiple copies of model parameters in memory. Also, algorithms like PPO alternate between these workloads synchronously and sequentially. These factors result in low hardware utilization, which is particularly pronounced in the long CoT scenario due to higher variance in CoT length resulting in stragglers during inference (Kimi Team, 2025). We also ran into some difficulties scaling up model size to 32B, and we decided the number of GPUs required was too large to proceed. We eagerly anticipate advances in the ML and systems area that will undoubtedly accelerate research on long CoTs.

6.2. REINFORCE is more tricky to tune than PPO

We also explored REINFORCE++ (Hu, 2025) as a faster alternative to PPO for scaling up data. However, we found it to be significantly more unstable than PPO, leading to lower training accuracies (Figure 14). As this instability may be due to an untuned setup (Appendix E.5.9), we refrain from making general claims about the algorithm. Instead, we present this as an observation that may be useful to the community.

7. Conclusion

In this work, we demystify long CoT reasoning in LLMs. We show that long CoT in SFT raises the performance ceiling and enhances subsequent RL improvements. To address instability in CoT length scaling during RL, we introduce a cosine length scaling reward with a repetition penalty. Additionally, we explore the benefits of leveraging noisy web-extracted data, showing that it improves both SFT and RL when used with reference-guided rewards.

Acknowledgement

The authors would thank Yuanzhi Li for insightful discussions on this topic. The authors would also thank the SimpleRL team, particularly Weihao Zeng and Junxian He, for sharing their training experiences and experimental observations. Additionally, the authors appreciate Wenhu Chen, Xiaoyi Ren, Chao Li, Ziqiao Ma, Jiayi Pan, Xingyao Wang, and Seungone Kim for their valuable comments and discussions during the early or final stages of the project. Finally, the authors would acknowledge the DeepSeek-R1 and Kimik1.5 teams for their technical report releases, which inspired several additional experiment designs of this paper. This work was supported in part by a Carnegie Bosch Institute Fellowship to Xiang Yue.

Impact Statement

This paper aims to provide insights into scaling inference compute and training strategies to enable long chain-of-thought reasoning in large language models. The broader impacts of this work primarily relate to the potential for enhanced reasoning and problem-solving capabilities across various domains, where models capable of interpretable and verifiable reasoning could drive innovation and improve decision-making. Our findings emphasize the importance of ensuring robust training data preparation, stability, and alignment with verifiable ground truths. We encourage future research to actively develop safeguards that ensure these capabilities are used responsibly. This includes careful design of reward shaping and training protocols to minimize unintended consequences while maximizing societal benefits.

References

Anthropic. Introducing claude, 2023. URL https://www.anthropic.com/index/introducing-claude.

Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 327–336, 1998.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Chen, W., Yin, M., Ku, M., Lu, P., Wan, Y., Ma, X., Xu, J., Wang, X., and Xia, T. TheoremQA: A theorem-driven question answering dataset. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=Wom397PB55.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 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.

Dao, T. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Dong, H., Xiong, W., Goyal, D., Zhang, Y., Chow, W., Pan, R., Diao, S., Zhang, J., KaShun, S., and Zhang, T. Raft: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023.

Feng, X., Wan, Z., Wen, M., McAleer, S. M., Wen, Y., Zhang, W., and Wang, J. Alphazero-like tree-search can guide large language model decoding and training, 2023.

Gao, Z., Wang, H., Lu, C., Lu, T., Froudist-Walsh, S., Chen, M., Wang, X.-J., Hu, J., and Sun, W. The neural basis of delayed gratification. *Science Advances*, 7(49):eabg6611, 2021. doi: 10.1126/sciadv.abg6611. URL https://www.science.org/doi/abs/10.1126/sciadv.abg6611.

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.

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 Vanschoren, J. and Yeung, S. (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Hou, Z., Lv, X., Lu, R., Zhang, J., Li, Y., Yao, Z., Li, J., Tang, J., and Dong, Y. Advancing language model reasoning through reinforcement learning and inference scaling, 2025. URL https://arxiv.org/abs/2501.11651.

Hu, J. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv preprint arXiv:2501.03262*, 2025.

Hu, J., Wu, X., Zhu, Z., Xianyu, Wang, W., Zhang, D., and Cao, Y. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework, 2024. URL https://arxiv.org/abs/2405.11143.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Kimi Team. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

Lamb, A. M., ALIAS PARTH GOYAL, A. G., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. Professor forcing: A new algorithm for training recurrent networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/16026d60ff9b54410b3435b403afd226-Paper.pdf.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., Gu, Y., Malik, S., Graf, V., Hwang, J. D., Yang, J., Bras, R. L., Tafjord, O., Wilhelm, C., Soldaini, L., Smith, N. A., Wang, Y., Dasigi, P., and Hajishirzi, H. Tulu 3: Pushing frontiers in open language model post-training, 2024.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Longpre, S., Hou, L., Vu, T., Webson, A., Chung, H. W., Tay, Y., Zhou, D., Le, Q. V., Zoph, B., Wei, J., and Roberts, A. The flan collection: Designing data and methods for effective instruction tuning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22631–22648. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/longpre23a.html.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL https://arxiv.org/abs/1608.03983.

Meta. Introducing meta llama 3: The most capable openly available llm to date., 2024. URL https://ai.meta.com/blog/meta-llama-3.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

OpenAI. Learning to reason with llms, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback, 2022.

Pan, J., Zhang, J., Wang, X., and Yuan, L. Tinyzero. https://github.com/Jiayi-Pan/TinyZero, 2025. Accessed: 2025-01-24.

Qwen Team. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement, 2024a. URL https://arxiv.org/abs/2409.12122.

Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown, 2024b. URL https://qwenlm.github.io/blog/qwq-32b-preview/.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter

models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, 2020.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Singh, A., Co-Reyes, J. D., Agarwal, R., Anand, A., Patil, P., Liu, P. J., Harrison, J., Lee, J., Xu, K., Parisi, A., et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Tong, Y., Zhang, X., Wang, R., Wu, R., and He, J. DART-math: Difficulty-aware rejection tuning for mathematical problem-solving. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=zLU21oQjD5.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., and Chen, W. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024a. URL https://openreview.net/forum?id=y10DM6R2r3.

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, 2024b.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022.

Yu, L., Jiang, W., Shi, H., YU, J., Liu, Z., Zhang, Y., Kwok, J., Li, Z., Weller, A., and Liu, W. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

Yuan, Z., Yuan, H., Li, C., Dong, G., Tan, C., and Zhou, C. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Yue, X., Zheng, T., Zhang, G., and Chen, W. Mammoth2: Scaling instructions from the web. *arXiv preprint arXiv:2405.03548*, 2024.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Zeng, W., Huang, Y., Liu, W., He, K., Liu, Q., Ma, Z., and He, J. 7b model and 8k examples: Emerging reasoning with reinforcement learning is both effective and efficient. https://hkust-nlp.notion.site/simplerl-reason, 2025. Notion Blog.

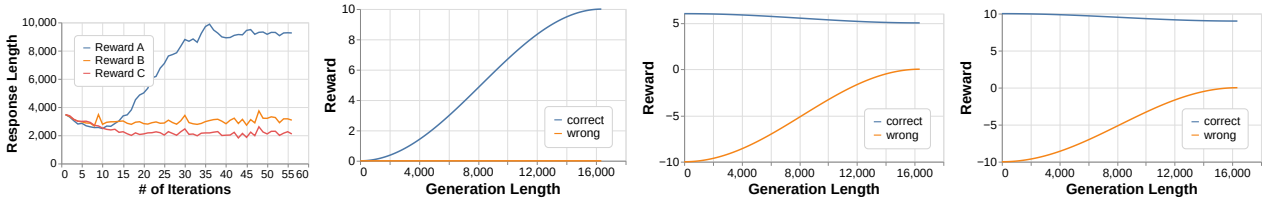
Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=VqkAKQibpq.

A. Related Work

Complex reasoning and chain of thought prompting. Large Language Models (LLMs) have demonstrated remarkable capabilities in various natural language processing tasks, including complex reasoning. A significant advancement in improving LLM reasoning ability is the implementation of Chain of Thought (CoT) prompting (Wei et al., 2022). This technique involves guiding models to generate intermediate reasoning steps, thereby improving their performance on tasks that require logical deduction and multistep problem solving. Initial studies (Lambert et al., 2024; Wei et al., 2022; Longpre et al., 2023; Yu et al., 2024) focused on short CoT, where models produce concise reasoning paths to arrive at solutions. Although effective for straightforward problems, short CoT can be limiting when addressing more intricate tasks that necessitate deeper deliberation. OpenAI's o1 (OpenAI, 2024) series models were the first to introduce inference-time scaling by increasing the length of the CoT reasoning process. This approach helps LLMs tackle complex problems by breaking them into finer steps and reflecting during problem-solving, leading to more accurate and comprehensive solutions. In this work, we explore long CoT by identifying key factors that enable models to exhibit this behavior, encouraging advanced reasoning capabilities.

Reinforcement learning for LLM. Reinforcement Learning (RL) has proven effective in enhancing LLM performance across domains. RL techniques, such as Reinforcement Learning from Human Feedback (RLHF), align model outputs with human preferences, improving coherence (Ouyang et al., 2022). Recent studies (Kimi Team, 2025; DeepSeek-AI, 2025; Lambert et al., 2024) leverage RL to enable LLMs to explore reasoning paths autonomously for complex problems. DeepSeek-R1 (DeepSeek-AI, 2025) achieves strong performance in mathematics, coding, and reasoning tasks without relying on a trained reward model (Lightman et al., 2024; Wang et al., 2024b) or tree search (Feng et al., 2023; Snell et al., 2024). Notably, this capability emerges even in base models without supervised fine-tuning, albeit at the cost of output readability. Similarly, Kimi K1.5 (Kimi Team, 2025) enhances general reasoning with RL, focusing on multimodal reasoning and controlling thought process length. These works highlight RL's role in optimizing reasoning when intermediate steps are hard to supervise, and only final outcomes are verifiable. Our research share a similar setup but with more detail on disentangling how different model behaviors emerge under varying training conditions and initialization strategies.

B. Figures and Tables



(a) Response lengths under different Cosine Rewards (b) Reward A (c) Reward B (d) Reward C

Figure 9. (a) Tuning the hyperparameters of the Cosine Reward results in different length scaling behavior. Note that Reward A results in some performance degradation on downstream tasks due to the model's reduced ability to stop within the context window. (b) Reward A: $r_0^c = 0, r_L^c = 10, r_0^w = r_L^w = 0$, (c) Reward B: $r_0^c = 6, r_L^c = 5, r_0^w = -10, r_L^w = 0$ (d) Reward C: $r_0^c = 10, r_L^c = 9, r_0^w = -10, r_L^w = 0$.

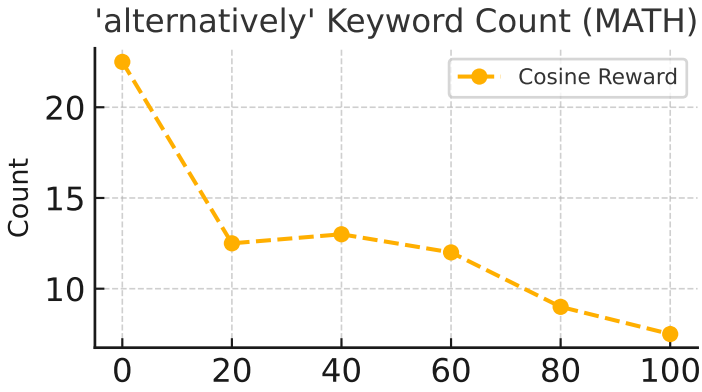


Figure 10. CoT branching frequency, estimated by the keyword count of the pivot word "alternatively,", decreased under the Cosine Reward with more training compute. We attributed this, along with increased repetition, to reward hacking.

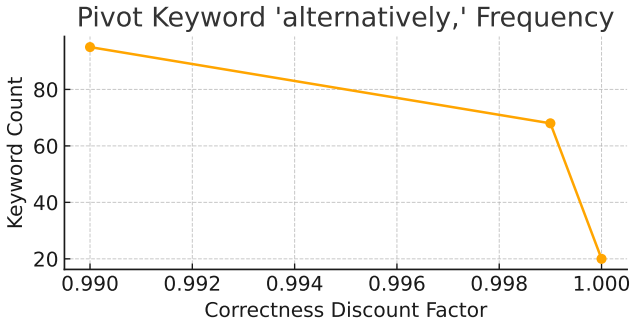


Figure 11. Branching frequency in CoT at different γ_c values. Lowering the discount factor increased branching frequency, causing the model to abandon problem-solving approaches more quickly.

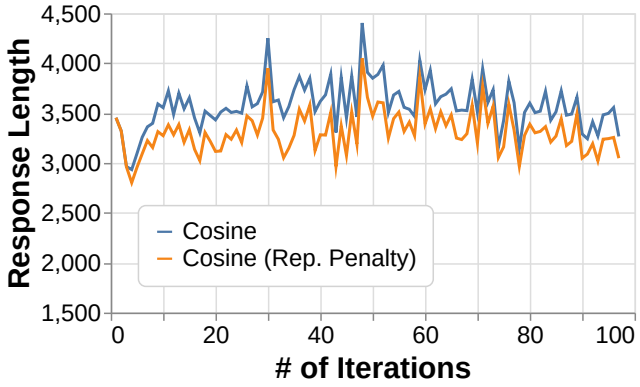


Figure 12. Training response length of models trained with Cosine Reward with and without repetition penalty. We see that repetition penalty reduced the length.

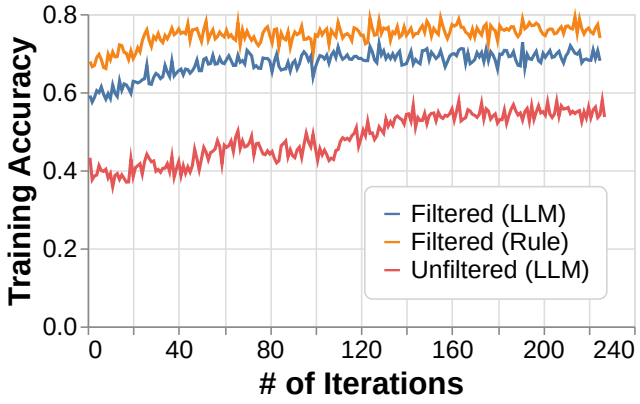


Figure 13. Training accuracies of models using rule-based and LLM-based verifiers on filtered and unfiltered data.

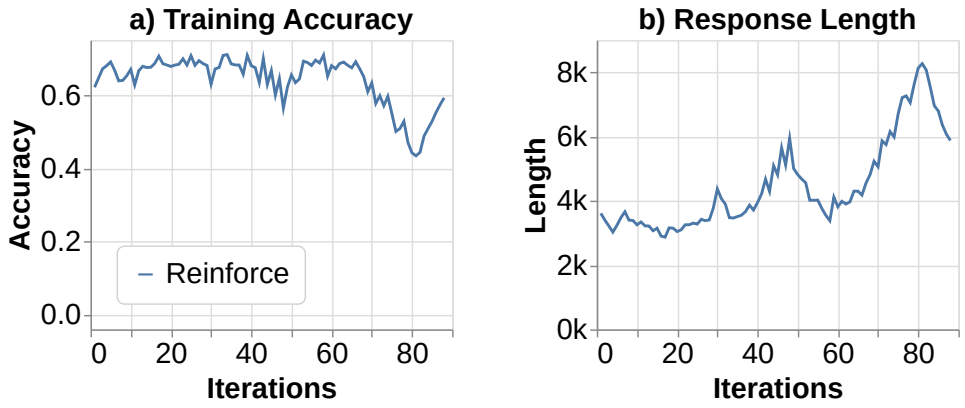


Figure 14. Reinforce with classic reward shows signs of training instability.

Table 6. Performance of model trained with different discount factors for the correctness (cosine) reward and repetition penalty. We see that different reward types have different optimal values.

Correctness Discount	Repetition Discount	MATH -500	AIME 2024	Theo. QA	MMLU -Pro-1k
SFT		50.4	3.5	20.6	32.4
1.000	1.000	55.7	**5.0**	25.7	34.5
	0.999	**58.0**	4.6	**26.0**	**36.5**
	0.99	57.8	3.8	24.5	33.3
0.999	0.999	53.5	2.1	19.5	30.7
	0.99	55.2	1.7	18.5	32.0
0.99	0.99	47.9	0.2	15.6	25.5

C. Algorithms and Formulas

C.1. Cosine Reward Formula

$$\textbf{CosFn}(t, T, \eta_{min}, \eta_{max}) = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{t\pi}{T})) \tag{1}$$

The formula above is commonly used as the learning rate schedule during gradient descent optimization. It was introduced by (Loshchilov & Hutter, 2017).

C.2. N-gram Repetition Penalty

Algorithm 1 N-gram Repetition Penalty

Input:
  s : sequence of tokens
  l : sequence length
  N : n-gram size
  P : penalty value
  m : maximum sequence length
Output: $r \in \mathbb{R}^m$
$seq \leftarrow s[1:l]$ {Extract subsequence of length l}
$ngrams \leftarrow \emptyset$ {Set of observed n-grams}
$r \leftarrow \vec{0} \in \mathbb{R}^m$ {Initialize reward vector}
for $j \leftarrow 1$ to $|seq| - N + 1$ **do**
  $ng \leftarrow (seq[j], seq[j+1], ..., seq[j+N-1])$ {Current n-gram}
  if $ng \in ngrams$ **then**
    for $t \leftarrow j$ to $j + N - 1$ **do**
      $r[t] \leftarrow P$ {Apply penalty}
    end for
  end if
  $ngrams \leftarrow ngrams \cup \{ng\}$
end for
Output: r

D. Extracts

Short-Term Thinking ($\gamma_c = 0.99$)

But in this case, we saw that the fixed points don't satisfy the equation

$$p(p(x)) = x.$$

Wait, maybe the solutions to $p(p(x)) = x$ are not the fixed points of p, but something else.
Alternatively, perhaps I can consider that $p(p(x)) = x$ implies that p is a bijection, but since $p(x)$ is quadratic, it's not necessarily invertible over all real numbers.
Wait, but the problem says that $p(p(x)) = x$ has four real solutions, so perhaps there are four values of x that satisfy this equation.
Alternatively, perhaps I need to find the roots of

$$p(p(x)) - x = 0,$$

and solve for x. But that seems complicated. Maybe there's a better way.

E. Experimental Setup

E.1. Evaluation Setup

Benchmarks Below are details of our evaluation benchmarks:

- **MATH-500** (Hendrycks et al., 2021): an in-domain mathematical reasoning benchmark. MATH consists of 12,500 problems from American high school math competitions. For efficiency, we adopt MATH-500, a widely-used i.i.d. subset of its test split.
- **AIME 2024**: an out-of-domain mathematical reasoning benchmark consisting of the 30 problems from American Invitational Mathematics Examination (AIME) 2024.
- **TheoremQA** (Chen et al., 2023): an out-of-domain STEM reasoning benchmark consisting of 800 samples. It covers 350+ theorems spanning across Math, EE&CS, Physics and Finance.
- **MMLU-Pro-1k** (Wang et al., 2024a): an out-of-domain general reasoning benchmark. MMLU-Pro comprises over 12,000 questions from academic exams and textbooks, spanning 14 diverse domains including Biology, Business, Chemistry, Computer Science, Economics, Engineering, Health, History, Law, Math, Philosophy, Physics, Psychology, and Others. For efficiency, we adopt an 1,000-sample i.i.d. subset of its test split, called MMLU-Pro-1k. We tried to keep the distribution identical to the original one. Figure 15 shows the distribution before/after the downsampling.

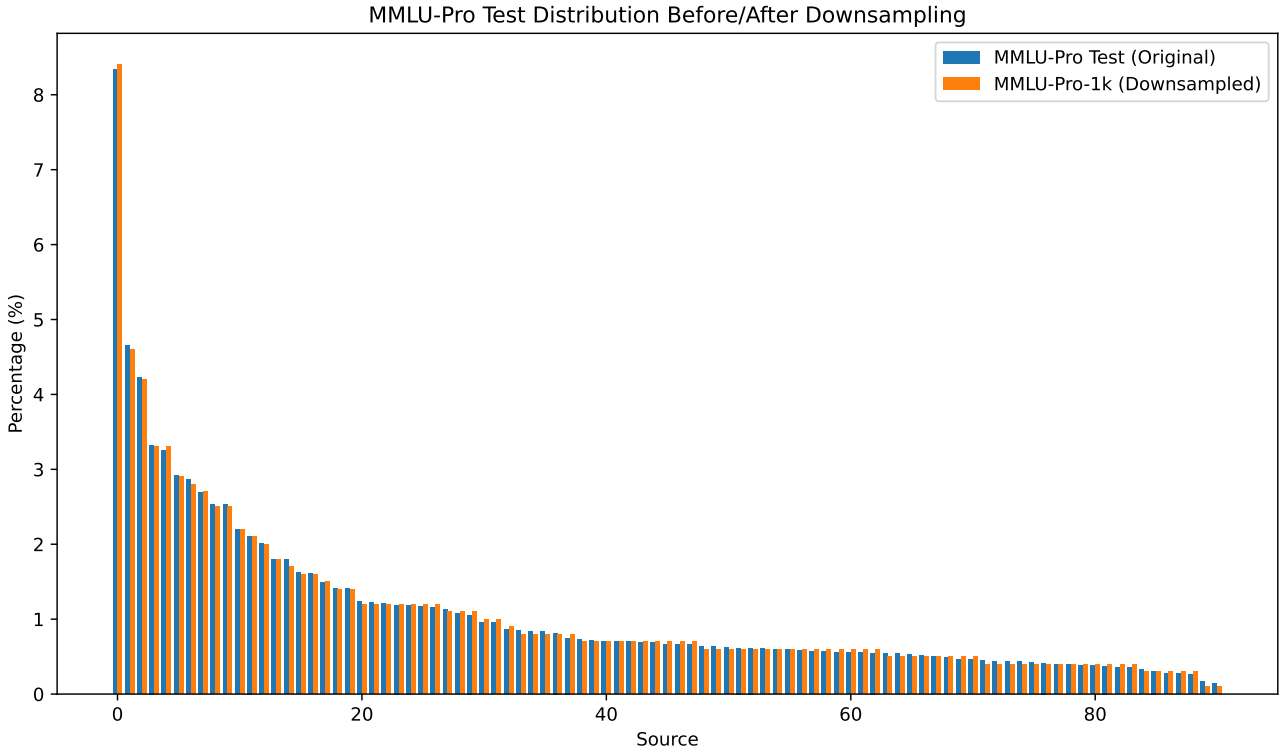


Figure 15. MMLU-Pro test distribution before/after downsampling for the MMLU-Pro-1k subset. The subset is i.i.d. to the full set.

Statistical Metrics We calculate the average accuracy with at least 4 random seeds. To tame the variance caused by the small size of AIME 2024, we sample 16 responses per prompt.

Implementation We adopt the vLLM library to accelerate the inference and SymEval[1], an elaborate answer grader capable of processing complex mathematical objects like matrices and functions, keeping consistent with the sampling and reward

[1] https://github.com/tongyx361/symeval

implementation in our RL setup. Note that a few RL experiments are carried out with an earlier version of the grader, causing nuanced performance differences.

E.2. Details about Distillation

To distill long CoT trajectories from `QwQ-32B-Preview`, we adopt the temperature $t = 1.0$, the top-p value of 0.95 and the maximum output length of 8192 tokens. Our preliminary experiments show that 8192 tokens show almost the same accuracy with `QwQ-32B-Preview` on MATH-500 as 16384 tokens, while costing significantly less time.

To distill short CoT trajectories from `Qwen2.5-Math-72B-Instruct`, we adopt the temperature $t = 0.7$, the top-p value of 0.95 and the maximum output length of 4096 tokens, since `Qwen2.5-Math-72B-Instruct` has a context limit of 4096 tokens and our preliminary experiments observe a non-negligible ratio of nonsense output when using $t = 1.0$.

Note the data is distilled with SGLang (Zheng et al., 2024) with an early version of our code.

When applying rejection sampling, we adopt the SymEval verifier as the grader.

E.3. Details abour SFT Setup

We use OpenRLHF (Hu et al., 2024) for our SFT experiments. By default, we adopt the SFT hyperparameters in Table 7.

For efficiency, we utilize Flash Attention 2 (Dao, 2024) and ZeRO (Rajbhandari et al., 2020) stage 1 based on the DeepSpeed library (Rasley et al., 2020). We uniformly set the micro batch size as 1 since we don't observe acceleration when increasing it.

Table 7. SFT Hyperparameters

Batch Size	Context Length	LR	Epochs
256	128K	5e-6	2

E.4. Details about RL Setup

We use OpenRLHF (Hu et al., 2024) for our RL experiments. When describing hyperparameters, we adopt the same naming conventions as OpenRLHF.

By default, we adopt the PPO algorithm with our cosine length-scaling reward function based on the ruled

E.5. Experiment Hyperparameters

Note that the `BS` column below refers to both `rollout_batch_size` (the number of prompts used in a sampling-training iteration) and `train_batch_size` (the number of samples used in a training update) because we adopt the same number for these two hyperparameters in all our RL setups. Also, the `Samples` column refers to the number of samples per prompt.

E.5.1. Details of Section 4.1 (CoT Length Stability)

SFT Data: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 8. Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Llama3.1-8B	Correct: +1	$\lambda = 1$ $\gamma = 1$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 4.5e-6	0.01
Qwen2.5-Math-7B	Correct: +1	$\lambda = 1$ $\gamma = 1$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 4.5e-6	0.01

E.5.2. Details of Section 4.2 (Active Scaling of CoT Length)

SFT Data: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 9. Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Llama3.1-8B	Correct: +1	$\lambda = 1$ $\gamma = 1$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 4.5e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$	$\lambda = 1$ $\gamma = 1$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 4.5e-6	0.01
Llama3.1-8B	Correct: +1	$\lambda = 1$ $\gamma = 1$	8	16	512	2	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$	$\lambda = 1$ $\gamma = 1$	8	16	512	2	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	8	16	512	2	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01

E.5.3. Details of Section 4.3 (Length Scaling at Different Model Sizes)

SFT Data:

`Qwen-2.5-32B`: The base model was trained under RL without a SFT stage.

`Qwen-2.5-1.5B`: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 10. Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Qwen-2.5-32B	Correct: +1	$\lambda = 1$ $\gamma = 1$	∞	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 Constant after 10-step linear warmup	0.01
Qwen-2.5-32B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	∞	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 Constant after 10-step linear warmup	0.01
Qwen-2.5-32B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	∞	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 Constant after 10-step linear warmup	0.01
Qwen-2.5-1.5B	Correct: +1	$\lambda = 1$ $\gamma = 1$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 Cosine with 3% step warmup	0.01
Qwen-2.5-1.5B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 Cosine with 3% step warmup	0.01

E.5.4. Details of Section 4.4 (Cosine Reward Hyperparameters)

SFT Data: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 11. Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Llama3.1-8B	Cosine: $r_0^c = 0$ $r_L^c = +10$ $r_0^w = 0$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +6$ $r_L^c = +5$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +10$ $r_L^c = +9$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01

E.5.5. Details of Section 4.5 (Context Window Size)

SFT Data: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 12. Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	8	8	512	1	Prompt: 2048 Gen: 2048	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	8	8	512	1	Prompt: 2048 Gen: 6144	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01

E.5.6. Details of Section 4.6 (Length Reward Hacking)

SFT Data: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 13. Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$	$\lambda = 1$ $\gamma = 1$	8	16	512	2	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	8	16	512	2	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01

E.5.7. Details of Section 4.7 (Optimal Discount Factors)

SFT Data: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 14. Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 1$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.999$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 0.999$ $\gamma_p = 0.999$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 0.999$ $\gamma_p = 0.99$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01
Llama3.1-8B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 0.99$ $\gamma_p = 0.99$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01

E.5.8. Details of Section 5.2 (RL with Noisy Verifiable Data)

SFT Data: 115k instances of filtered long CoT data distilled from `QwQ-32B-Preview` with WebInstruct.

Table 15. Hyperparameters

Base Model	RL Prompt Set Verifier	Rewards	GAE	Episodes Instances	Samples	BS	Epochs	Context Length	LR KL
Llama3.1-8B	Unfiltered (30k sampled) Symeval	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	1 30k	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 KL: 0.01
Llama3.1-8B	Unfiltered (30k sampled) LLM-as-a-judge	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	1 30k	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 KL: 0.01
Llama3.1-8B	Filtered (115k) Symeval	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	1 115k	8	512	1	Prompt: 2048 Gen: 14336	Actor: 1e-6 Critic: 4.5e-6 KL: 0.01
Llama3.1-8B	Filtered (115k) LLM-as-a-judge	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma_c = 1$ $\gamma_p = 0.99$	1 115k	8	512	1	Prompt: 2048 Gen: 14336	Actor: 1e-6 Critic: 4.5e-6 KL: 0.01

E.5.9. Details of Section 6.2 (REINFORCE is more tricky to tune than PPO)

SFT Data: Long CoT data distilled from `QwQ-32B-Preview` with the MATH train split.

Table 16. Hyperparameters

Base Model	Rewards	Gamma	Episodes	Samples	BS	Epochs	Context Length	LR	KL	Clip
Llama3.1-8B	Correct: +1	$\gamma = 1$	8 (stopped early)	8	512	1	Prompt: 2048 Gen: 14336	5e-7	0.01	0.1

E.6. Implementation of the Model-Based Verifier

We used `Qwen2.5-7B-Instruct` as our model-based verifier. It was provided with both the reference answer and the suffix of the long CoT. We truncated the long CoT to avoid confusing the verifier. We used the following prompt.

Prompt Template for Model-Based Verifier

```
Given the following last 20 lines of the LLM response to a math question
and the reference solution to that question, evaluate if the LLM response is correct
     based only on the LLM's final answer.

LLM response (last 20 lines):
...
{out}

Reference solution:
{ref}

Explain your thought process step-by-step before responding with 'Judgement: <
    correct/wrong/not_found>'
```

E.7. Implementation of Short-Form Answer Extraction

We use the `Llama-3.1-8B-Instruct` model to extract short-form answer from QA pairs in WebInstruct, with the following prompt template:

Prompt Template for Short-Form Answer Extraction

```
Problem: {Problem}

Solution: {Solution}

Based on the Problem and the Solution, extract a short final answer that is easy to
    check.
Provide the short final answer in the format of "The final answer is $$
\boxed{...}
$$"
- If the answer is a mathematical object, write it in LaTeX, e.g., "The final answer
     is $$
\boxed{\frac{1}{2}}
$$"
- If the answer is a boolean, write it as "True" or "False", e.g., "The final answer
     is $$
\boxed{True}
$$"
- If the Problem can't be answered in a short form, respond with "" like "The final
    answer is $$
\boxed{}
$$"
```

For generation parameters, we use temperature $t = 0$ (greedy decoding) and set the maximum output length as 512 tokens.

After generation, we simply extract the short-form answer from within the `\boxed{...}`.

E.8. Action Prompting Framework

We studied the publicly released CoTs of `o1-preview` and identified that its thoughts could be categorized into a few types of actions (listed below). To construct long CoTs, we designed prompts for each of these actions and implemented a multi-step prompting framework to sequence them. The framework ceded control flow of the CoT to the LLM, with

the LLM making branching or looping decisions while the framework acted more passively as a state machine reacting to the LLM outputs. The framework took care of the boilerplate around constructing the CoT with an append-only log and managed all of the orchestration.

- `clarify`: Making some observations about the problem in order to identify an approach to solve it.
- `decompose`: Breaking the current problem down into smaller and easier sub-problems to solve.
- `solution_step`: Computing a single step in the solution. In the context of math, this could be doing some arithmetic or symbolic manipulation.
- `reflection`: Evaluating the current approach and partial solution to see if any mistakes were made, any sub-goals were achieved, or if alternative approaches should be considered instead. Note that we used a strong teacher model `o1-mini` for the `reflection` action as that one was a more difficult prompt to respond to correctly as it requires self-correction.
- `answer`: Responding with a final answer and terminating the CoT.

E.8.1. CONTROL FLOW

Simplified description of the interaction between the framework and LLM.

Algorithm 2 Action Prompting State Machine

Input: $prompt$
Output: $chain\_of\_thought$ sequence
$chain\_of\_thought \leftarrow [prompt]$ {Initialize singleton chain of thought sequence from prompt}
$state \leftarrow$ "clarify"
while True **do**
  if $state =$ "clarify" **then**
    $output \leftarrow$ prompt_action_clarify()
    $(state, thought) \leftarrow$ parse$(output)$
    $chain\_of\_thought$.append$(thought)$
  else if $state =$ "decompose" **then**
    $output \leftarrow$ prompt_action_decompose()
    $(state, thought) \leftarrow$ parse$(output)$
    $chain\_of\_thought$.append$(thought)$
  else if $state =$ "solution_step" **then**
    $output \leftarrow$ prompt_action_solution_step()
    $(state, thought) \leftarrow$ parse$(output)$
    $chain\_of\_thought$.append$(thought)$
  else if $state =$ "reflection" **then**
    $output \leftarrow$ prompt_action_reflection()
    $(state, thought) \leftarrow$ parse$(output)$
    $chain\_of\_thought$.append$(thought)$
  else if $state =$ "answer" **then**
    $output \leftarrow$ prompt_action_reflection()
    $(state, thought) \leftarrow$ parse$(output)$
    $chain\_of\_thought$.append$(thought)$
    return $chain\_of\_thought$ {Terminate after answer action}
  end if
end while

E.8.2. Action Prompting Templates

Action: Clarify

```
You are a very talented mathematics professor.
In a few sentences, VERY CONCISELY rephrase the problem to clarify its meaning and
    explicitly state what needs to be solved. Highlight any assumptions, constraints
     and potential misinterpretations.
Do NOT attempt to solve the problem yet -- you are just clarifying the problem in
    your mind.

<problem>
{goal}
</problem>

Answer in the following format:

<clarification>
Problem clarification as instructed above
</clarification>
<goal>
Summarize the problem into a single statement describing the goal, e.g. Find the
    value of the variable w.
</goal>
```

Action: Decompose

```
You are a talented mathematics professor.
You already have a partial solution to a problem.
In a single sentence, propose candidates for the next subgoal as the next step of
    the partial solution that will help you make progress towards the current goal.
Do not repeat any subgoal, we don't want any infinite loops!
Do not suggest using a computer or software tools.

<current goal>
{current_goal}
</current goal>
<parent goal>
{parent_goal}
</parent goal>
<partial solution>
{solution}
</partial solution>

Format your answer as follows:

<thinking>
step-by-step thinking of what the next possible subgoal should be, as well as some
    other alternatives that might also work
remember, we want to solve the parent goal WITHOUT repeating the subgoals that are
    already DONE.
do not suggest verification or checking.
{parent_goal}
</thinking>
<sentence>
single sentence describing the subgoal
phrase it as if you were thinking to yourself and are considering this as a
    hypothesis (don't express too much certainty)
</sentence>
<sentence>
single sentence describing an *ALTERNATIVE* subgoal, without repeating previous ones
start off with "Alternatively,"
</sentence>
<sentence>
single sentence describing an *ALTERNATIVE* subgoal, without repeating previous ones
start off with "Alternatively,"
</sentence>
```

Action: Solution Step

```
You are an extremely PEDANTIC mathematics professor who loves to nitpick.
You already have a partial solution to a problem. Your task is to solve *only* the
    current goal.
You should include symbols and numbers in every sentence if possible.

<current goal>
{current_goal}
</current goal>
<partial solution>
{solution}
</partial solution>

BE VERY CONCISE. Include calculations and equations in your response if possible,
    and make sure to solve them instead of just describing them.
DO NOT SOLVE THE WHOLE QUESTION, JUST THE CURRENT GOAL: {current_goal}
Do not repeat any calculations that were already in this prior step:
{prior_step}
```

Action: Reflection

```
You are a talented mathematics professor.
You already have a partial solution to a math problem.
Verify whether the current subgoal has been achieved.

<current goal>
{current_goal}
</current goal>
{parent_goal}
<partial solution>
{solution}
</partial solution>

Format your answer as follows:

<verification>
Come up with a quick, simple and easy calculation to double check that the solution
    is correct.
This calculation should not re-compute the solution in the same way, as that would
    defeat the purpose of double-checking.
Use one of the following strategies:
- An easier, alternative method to arrive at the answer
- Substituting specific values into equations and checking for consistency
- Working backwards from the answer to derive the given inputs and then checking for
     consistency
Be consise. Do not suggest using a computer.
At the end of your verification, restate the answer from the current solution. Do
    not calculate it if it hasn't been solved.
Phrase it as if you are reflecting as you solve the problem.
</verification>
<current_goal_achieved>
true or false, depending on whether the solution is correct and the current goal has
     been achieved: {current_goal}
</current_goal_achieved>
<parent_goal_achieved>
true or false, depending on whether the parent goal has been achieved:
{parent_goal.target}
</parent_goal_achieved>
<new_goal>
If the solution is not correct or the current goal has not been achieved, suggest an
     alternative current goal here in a single sentence.
Start off with "Alternatively,"
Your goal should be sufficiently different from subgoals that have been solved or
    that have timed out:
{parent_goal_tree}
</new_goal>
```

Action: Answer

```
Extract the final answer, making sure to obey the formatting instructions.
Solution:
{solution}

Formatting instructions:
{format}
```