DEMYSTIFYING LONG CHAIN-OF-THOUGHT REASONING IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling inference compute enhances reasoning in large language models (LLMs), with long chains-of-thought (CoTs) enabling strategies like 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 mechanics of long CoT reasoning, identifying the key factors that enable models to generate long CoT trajectories. Through extensive supervised fine-tuning (SFT) and RL experiments, we present four main findings: (1) While SFT is not strictly necessary, it simplifies training and improves efficiency; (2) Reasoning capabilities tend to emerge with increased training compute, but their development is not guaranteed, making reward shaping crucial for stabilizing CoT length growth; (3) Scaling verifiable reward signals is critical for RL. We find that leveraging noisy, web-extracted solutions with filtering mechanisms shows strong potential, particularly for out-of-distribution (OOD) tasks such as STEM reasoning; and (4) Core abilities like error correction are inherently present in base models, but incentivizing these skills effectively for complex tasks via RL demands significant compute, and measuring their emergence requires a nuanced approach.

1 INTRODUCTION

Large language models (LLMs) (Brown et al., 2020; Touvron et al., 2023; Anthropic, 2023; OpenAI, 2023) have demonstrated remarkable reasoning abilities in domains like mathematics (Cobbe et al., 2021) and programming (Chen et al., 2021). Recently, OpenAI's o1 models (OpenAI, 2024) have demonstrated significant breakthroughs in these tasks. A key distinguishing feature of these models is their ability to scale up inference compute with long CoTs, which include 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, which helps to avoid reward hacking in reinforcement learning (RL) at scale. However, a comprehensive understanding of how models learn and generate long CoTs remains limited. In this work, we 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 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. So we introduce a cosine length-scaling reward with a repetition penalty, which stabilizes CoT growth while encouraging emergent 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 them up 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. (2024b). While these "silver" supervision signals introduce uncertainty, we find

that, with appropriate filtration, they show promise, especially in out-of-distribution (OOD) reasoning scenarios such as STEM problem-solving.

4) *Origins of Long CoT Abilities and RL Challenges* Core skills like branching and error validation are inherently present in base models, but effective RL-driven incentivization demands careful designs. We examine RL incentives on long CoT generation and discuss nuances in analyzing the features like emergent behaviors and length scaling.

2 IMPACT OF SFT ON LONG COT

In this section, we compare long and short CoT data for SFT and in the context of RL initialization.

2.1 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.

Setup. To curate the SFT data, for long CoT, we distill from `QwQ-32B-Preview` (we discuss other long CoT data construction methods in §2.3). For short CoT, we distill from `Qwen2.5-Math-72B-Instruct`, which is a capable short CoT model in math reasoning. Specifically, we perform rejection sampling by first sampling N candidate responses per prompt and then filtering for ones with correct answers. For long CoT, we use $N \in \{32, 64, 128, 192, 256\}$, while for short CoT, we use $N \in \{32, 64, 128, 256\}$, skipping one N for efficiency. In each case, the number of SFT tokens is proportional to N. We use the base model `Llama-3.1-8B` (Meta, 2024). Please refer to Appendix K.3 for more details about the SFT setup.

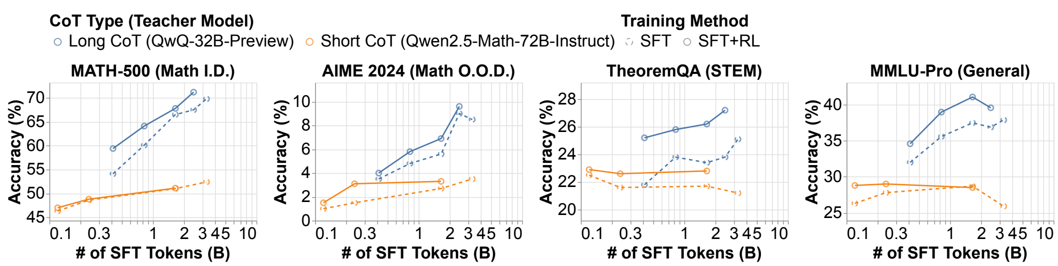


Figure 1: Scaling curves of SFT and RL on `Llama-3.1-8B` with long CoTs and short CoTs. SFT with long CoTs can scale up to a higher upper limit and has more potential to improve with RL.

Takeaway 2.1 for SFT Scaling Upper Limit

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

Result. The dashed lines in Figure 1 illustrate that as we scale up the SFT tokens, long CoT SFT continues to improve model accuracy, whereas short CoT SFT saturates early at a lower accuracy level. For instance, on MATH-500, long CoT SFT achieves over 70% accuracy and has yet to plateau even at 3.5B tokens. In contrast, short CoT SFT converges below 55% accuracy, with an increase in SFT tokens from approximately 0.25B to 1.5B yielding a marginal absolute improvement of about 3%.

2.2 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 §2.1, 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 §3. Further details about our RL setup and hyperparameters can be found in Appendix K.4 & K.5.1 respectively.

Takeaway 2.2 for SFT Initialization for RL

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

Result. The gap between solid and dashed lines in Figure 1 shows that models initialized with long CoT SFT can usually be effectively 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% absolute, while short CoT models have almost the same accuracies before and after RL.

2.3 Sources of Long CoT SFT Data

To curate long CoT data, we compare two approaches: (1) **Construct** long CoT trajectories by prompting short CoT models to generate primitive actions and sequentially combining 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 K.8) which defined the following actions: `clarify`, `decompose`, `solution_step`, `reflection`, and `answer`. We employed multi-step prompting with a short CoT model (e.g., `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 around 200k SFT samples, and use the same RL setup as in §2.2.

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**

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.

Takeaway 2.3 for Long CoT Cold Start

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

Result. Table 1 shows that the model distilled from emergent long CoT patterns generalizes better than the constructed pattern, and can be further significantly improved with RL, while the model trained on constructed patterns 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 improves its accuracy by around 20% relative, 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.

3 Impact of Reward Design on Long CoT

This section examines reward design, focusing on its influence on CoT length and model performance.

3.1 CoT Length Stability

Recent studies on long CoT (DeepSeek-AI, 2025; Kimi Team, 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 based on length and repetition penalties.

Setup. We used two different models fine-tuned on long CoT data distilled from `QwQ-32B-Preview` using the MATH train split, with a context window size of 16K. The models were `Llama3.1-8B` and `Qwen2.5-Math-7B`. We used a rule-based verifier along and a simple reward of 1 for correct answers. We shall refer to this as the *Classic Reward*. More details can be found in Appendix K.5.2.

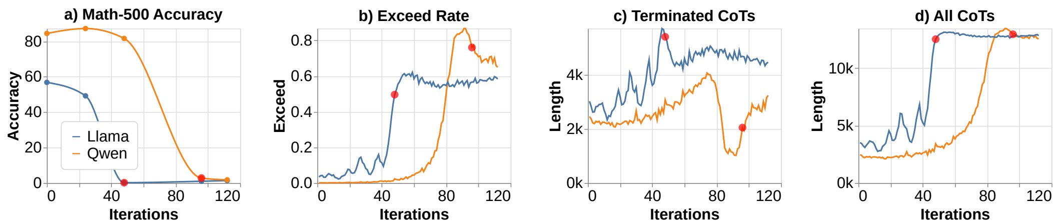


Figure 2: Both `Llama3.1-8B` and `Qwen2.5-Math-7B` models trained under RL with the Classic Reward manifested emergent CoT length scaling past the context window size, resulting in a decline in MATH-500 accuracy. The red points on the charts correspond to the iteration where the accuracy dropped to near zero. "Terminated CoTs" refer to responses that conclude within the context length.

Takeaway 3.1 for CoT Length Stability

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

Results. We observed that both models increased their CoT length during training, 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 size leveled off at a certain threshold below 1 (Figure 2). This suggests that exceeding the limit started to apply significant downward pressure on the CoT length distribution, and highlights the context window size's role in implicit length penalization. Notably, a trajectory might be penalized even without an explicit exceed-length penalty due to reward or advantage normalization, both of which are standard in RL.

3.2 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.

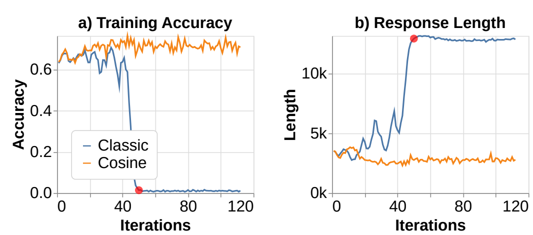


Figure 3: `Llama3.1-8B` trained with length shaping using the Cosine Reward exhibited more stable (a) training accuracy and (b) response length. This stability led to improved performance on downstream tasks (Figure 4). Red points indicate where training accuracy dropped to near zero.

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 8. This is a *sparse* reward, only awarded once at the end of the CoT based on the correctness of the answer. The formula can be found in equation 1 in the appendix.

Takeaway 3.2 for Active Scaling of CoT Length

Reward shaping can be used to stabilize and control CoT length while improving accuracy. (Figure 3, 4)

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 K.5.3.

Result. We found that the Cosine Reward significantly stabilized the length scaling behavior of the models under RL, thereby also improving the training accuracy and RL efficiency (Figure 3). We also observed improvements in performance on downstream tasks (Figure 4).

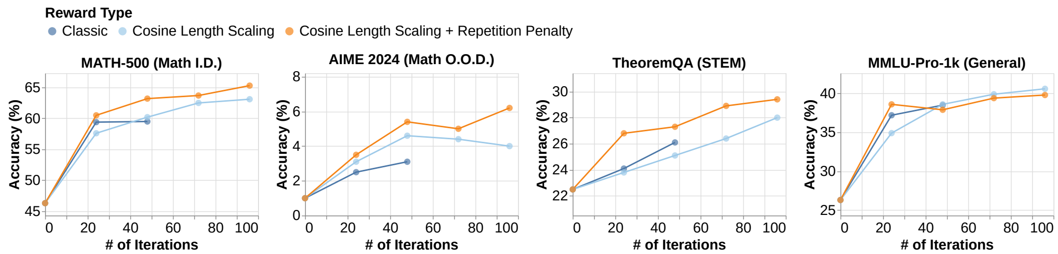


Figure 4: Performance of `Llama-3.1-8B` trained with different reward functions on a variety of evaluation benchmarks.

3.3 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 K.5.4.

Takeaway 3.3 for Cosine Reward Hyperparameters

Cosine Reward can be tuned to incentivize various length scaling behaviors. (Figure 9)

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 risk aversion, where the ratio of the correct and wrong rewards impacts how confident the model has to be about an answer to derive a positive reward from terminating its CoT with this answer.

3.4 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 K.5.5.

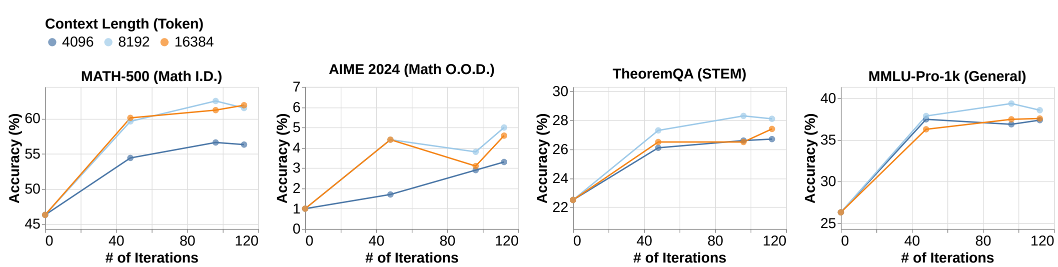


Figure 5: Performance of `Llama-3.1-8B` trained with different context window sizes. All experiments used the same number of training samples.

Takeaway 3.4 for Context Window Size

Models might need more training samples to learn to utilize larger context window sizes. (Figure 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 same number of training samples (Figure 5). We see this as an indication that models need more training compute to learn to fully utilize longer context window sizes, resonating with the findings of Hou et al. (2025).

3.5 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 keyword "`alternatively,`" appeared in the CoT (Figure 11). 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 §3.6).

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 K.5.6.

Result. The repetition penalty resulted in better downstream task performance and also shorter CoTs, with better utilization of inference compute (Figure 4).

> **Takeaway 3.5 for Length Reward Hacking**
>
> Length rewards will be hacked with enough compute (Figure 11), but this can be mitigated using a repetition penalty. (Figure 4)

3.6 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 K.5.7.

> **Takeaway 3.6 for Optimal Discount Factors**
>
> Different kinds of rewards and penalties have different optimal discount factors. (Table 4)

Result. A lower discount factor effectively enforces the repetition penalty, whereas a higher discount factor enhances the correctness reward and the exceed-length penalty. The higher factor allows the model to be adequately rewarded for selecting a correct answer earlier in the CoT (Figure 4). We observed a rather interesting phenomenon where decreasing 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 12, Extract in Appendix J). 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 4).

4 Scaling up Verifiable Reward

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

In this section, We compare two main approaches to obtain rewards from noisy verifiable data: 1) to extract short-form answers and use a rule-based verifier; 2) to use a model-based verifier capable of processing free-form responses. Here, a key factor is whether the QA pair can have a short-form answer, so we also compare whether the dataset is filtered for samples with short-form answers.

Setup. We implement the model-based verifier by prompting `Qwen2.5-Math-7B-Instruct` with the raw reference solution. To extract short-form answers, we first prompt `Llama-3.1-8B-Instruct` to extract from the raw responses and then 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. For SFT we train `Llama-3.1-8B` on the filtered dataset as initialization for reinforcement learning (RL). In the RL stage, we use the full 462k prompt set in the unfiltered setup and the 115k subset in the filtered setup, training with 30k prompts and 4 responses per prompt. Further details about the model-based verifier, the answer extraction and the RL hyperparameters can be found in Appendix & K.5.8 & K.6 & K.7 respectively.

Prompt Set	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
Unfiltered	Rule-Based	45.4	3.3	25.9	35.1
	Model-Based	47.9	3.5	26.2	40.4
Filtered	Rule-Based	**48.6**	3.3	**28.1**	**41.4**
	Model-Based	47.9	**3.8**	26.9	**41.4**

Table 2: Performance of RL with different verifiers and prompt filtering methods. All the models here are fine-tuned from `Llama-3.1-8B`. The "MATH Baseline" is the model trained with SFT and RL on MATH only in Table 3. The other models are trained with SFT by distillation from `QwQ-32B-Preview` and RL with different setups.

Result. Table 2 shows that RL with the rule-based verifier on the filtered prompt set with short-form answers achieves the best performance across most benchmarks under the same number of RL samples. This might indicate that rule-based verifier after appropriate filtration can produce the highest-quality reward signals from noisy verifiable data. Moreover, compared to the model trained on human-annotated verified data (MATH), leveraging noisy yet diverse verifiable data still significantly boosts performance on O.O.D. benchmarks, with absolute gains of up to 2.9% on TheoremQA and 6.8% on MMLU-Pro-1k. In contrast, applying a rule-based verifier to unfiltered data results in the worst performance. This might be caused by its low training accuracy on free-form answers, while the model-based verifier achieves much better performance.

Takeaway 4 for RL with Noisy Verifiable Data

To obtain reward signals from noisy verifiable data, the ruled-based verifier after filtering the prompt set for short-form answers works the best. (Table 2)

5 Exploration on RL from the Base Model

DeepSeek-R1 (DeepSeek-AI, 2025) has demonstrated that long chain-of-thought reasoning can emerge by scaling up reinforcement learning compute on a base model. Recent studies (Zeng et al., 2025; Pan et al., 2025) have attempted to replicate this progress by running a relatively small number of RL iterations to observe the emergence of long CoT behavior (e.g., the "aha moment" (DeepSeek-AI, 2025), an emergent realization moment that enables critical functions like self-validation and correction). We discuss nuances in measuring their emergence in this section. For more related analysis and results, please refer to Appendix D & E & F

5.1 Nuances in Analysis Based on Emergent Behaviors

Self-validation behaviors are sometimes flagged as emergent behaviors or "aha-moment" by the model's exploration, since such patterns are rare in short CoT data. However, we notice that sometimes self-validation behaviors already exist in the base model and reinforcing them through RL requires strict conditions, such as a strong base model.

Setup. We follow the setup from Zeng et al. (2025) to train `Qwen2.5-Math-7B` using PPO with a rule-based verifier on approximately 8k MATH level 3-5 questions, but we use our own rule-based verifier implementation. For inference, we adopt temperature $t = 0$ (greedy decoding), as our preliminary experiments show that $t = 0$ usually significantly outperforms $t > 0$ for models obtained by direct RL from `Qwen2.5-Math-7B`. We use the maximum output length of 4096 tokens considering the training context length of 4096 tokens. Note that we use zero-shot prompting for the base model to avoid introducing biases to the output pattern. We select five representative keywords, “wait”, “recheck”, “alternatively”, “retry” and “however” from long CoT cases in previous works (OpenAI, 2024; DeepSeek-AI, 2025; Pan et al., 2025; Zeng et al., 2025), and calculate their frequencies to quantify the extent to which the model does self-validation. Further details about the RL hyperparameters can be found in Appendix K.5.9.

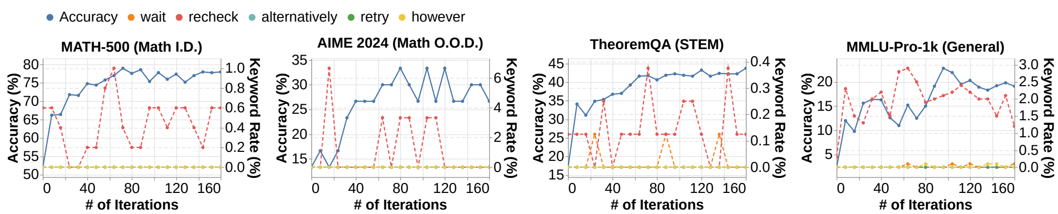


Figure 6: Dynamics of accuracies and reflection keyword rates on different benchmarks during our RL from the base model `Qwen2.5-Math-7B`. We do not see the keyword rates of self-validation patterns significantly improve during the RL training even though the accuracy is steadily increasing.

Result. Figure 6 shows that our RL from `Qwen2.5- Math-7B` effectively boosts the accuracies, but does not increase the frequency of the “recheck“ pattern existing in the output of the base model, nor effectively incentivize other reflection patterns such as “wait” and “alternatively”. This indicates that RL from the base model does not necessarily incentivize reflection patterns. Sometimes such behaviors exist in the base model’s output and RL does not substantially enhance them.

5.2 Nuances in Analysis Based on Length Scaling

The length scaling up is recognized as another important feature of the effective exploration of the model. However, we notice that sometimes length scaling up can be accompanied by the KL divergence decreasing, which raises the possibility that the length is mainly driven by the KL penalty, reverting back to the base model’s longer output, rather than by the model’s exploration.

Setup. The setup is the same as in §5.1. Besides the output token length, we also calculate the “coding rate”. We classify the model’s output as “coding” if it contains the “`` ```python ``”, since `Qwen2.5-Math-7B` uses both natural language and coding to solve mathematical problems. Note that we don’t execute the code in the coding output.

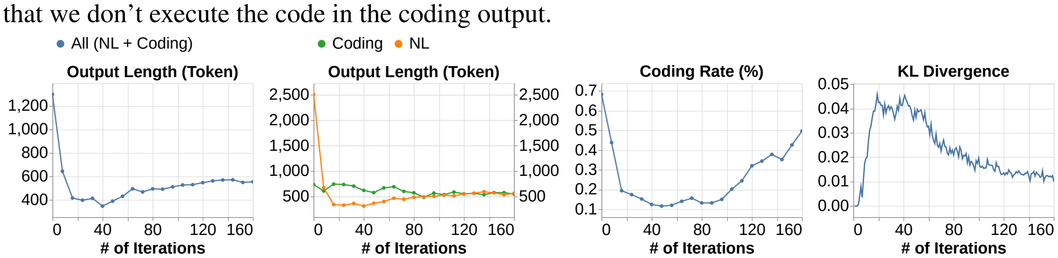


Figure 7: Dynamics of the output token lengths and the coding rate on MATH-500 and the KL divergence of the policy over the base model on MATH Lv3-5 (training data) during our RL from `Qwen2.5-Math-7B`. “NL” means Natural Language.

Result. Figure 7 (1) shows that the length of the output token increases after an initial drop, but never exceeds the initial length of the base model. Zeng et al. (2025) suggest that the initial drop may be due to the model transitioning from generating long coding outputs to shorter natural language outputs. However, Figure 7 (2) indicates that natural language outputs are actually longer than coding outputs, and the initial drop in length occurs in both types of output. Furthermore, Figure 7 (3) shows that the coding rate subsequently increases again, suggesting that the distinction between coding and natural language may not significantly impact the optimization process. Moreover, we suspect that the subsequent length scaling up is not from the model’s exploration, since when the length scales up, the KL divergence of the policy over the base model drops, as shown in Figure 7 (4). This might indicate that it is the KL penalty influencing length. If that is the case, there is little potential for the policy output length to exceed the base model’s since the exploration is limited by the KL constraint.

REFERENCES

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

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

A.Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pp. 21–29, 1997. doi: 10.1109/SEQUEN.1997.666900.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Hyung Won Chung. Dont teach. incentivize. Presentation slides, 2024. URL `https://t.co/2sjhynKxzJ`. Slide 48.

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

Tri Dao. 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.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, SHUM KaShun, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training, 2023.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. Advancing language model reasoning through reinforcement learning and inference scaling, 2025.

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

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework, 2024.

Kimi Team. Kimi k1.5: Scaling reinforcement learning with llms, 2025.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, and Adam Roberts. The flan collection: Designing data and methods for effective instruction tuning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (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.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

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/.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

Jiayi Pan, Junjie Zhang, Xingyao Wang, and Lifan Yuan. Tinyzero, 2025. URL https://github.com/Jiayi-Pan/TinyZero. Accessed: 2025-01-24.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text, 2023.

Qwen Team. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement, 2024a.

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

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 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.

Ahad Rana. Common crawl building an open web-scale crawl using hadoop, 2010. URL `https://www.slideshare.net/hadoopusergroup/common-crawlpresentation`.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 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.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

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

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

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. DART-math: Difficulty-aware rejection tuning for mathematical problem-solving. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothe Lacroix, Baptiste Rozire, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 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.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision, 2024b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

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

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MAmmoTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024a.

Xiang Yue, Tuney Zheng, Ge Zhang, and Wenhu Chen. Mammoth2: Scaling instructions from the web. *NeurIPS 2024*, 2024b.

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

Weihao Zeng, Yuzhen Huang, Wei Liu, Keqing He, Qian Liu, Zejun Ma, and Junxian He. 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.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

6 Appendix

A 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 goal is to *demystify long chain-of-thought reasoning* in LLMs. Through systematic analysis and ablations, we extract key insights and offer practical strategies to enhance and stabilize its performance.

A.1 Notation

Let x be a query, and let y be the 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 Validation and Correction: The model detects inconsistencies or mistakes in its intermediate steps and takes corrective actions to restore coherence and accuracy.

A.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.

A.3 Reinforcement Learning (RL)

After optional SFT initialization, we can further optimize the generation of long CoT 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. We adopted Proximal Policy Optimization (PPO) (Schulman et al., 2017) as the default policy optimization method in our experiments. We also briefly discuss REINFORCE (Sutton

& Barto, 2018) method in subsection B.3. We adopt a rule-based verifier as the reward function, which compares the predicted answer with the ground truth answer directly. The resulting updates push the policy to generate tokens that yield higher reward.

A.4 Training Setup

We adopt `Llama-3.1-8B` Meta (2024) and `Qwen2.5 -7B-Math` Qwen Team (2024a) as the base models, which are representative general and math-specialized models respectively. For both SFT and RL, we use the 7,500 training sample prompt set of MATH (Hendrycks et al., 2021) 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; Yue et al., 2024a; 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. (2024b) that is more diverse but without gold supervision signals like ground truth answers in §4. We train the models with the OpenRLHF framework Hu et al. (2024).

A.5 Evaluation Setup

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 (MATH-500 test set) and out-of-domain evaluations (AIME 2024, TheoremQA, MMLU-Pro-1k). By default, we generate from 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 K.1 for further details on the evaluation setup.

B Discussions and Future Work

In this work, we demystify long CoT reasoning in LLMs. In this section, we outline potential future directions.

B.1 Scaling up Model Size

We believe that model size is the primary factor limiting the emergence of the behavior observed in subsection 5.1. Hyung Won Chung Chung (2024) recently shared a similar perspective, suggesting that smaller models may struggle to develop high-level reasoning skills and instead rely on heuristic-based pattern recognition. Future research could investigate RL using a larger base model.

B.2 RL Infrastructure Is Still in Its Infancy

While attempting to scale up the model size, we encountered significant challenges in expanding to 32B, ultimately determining that the required number of GPUs was too large to proceed. We observe that open-source RL frameworks (e.g., OpenRLHF Hu et al. (2024)) often coordinate multiple systems optimized for different training and inference workloads, leading to multiple copies of model parameters being stored in memory. Additionally, algorithms like PPO alternate between these workloads synchronously and sequentially, further limiting efficiency. These factors contribute to low hardware utilization, an issue that is particularly exacerbated in long CoT scenarios due to the higher variance in CoT length, which leads to stragglers during inference Kimi Team (2025). We look forward to advancements in machine learning and systems research that will help overcome these limitations and accelerate progress in long CoT modeling.

B.3 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 13). As this instability may be due to an untuned setup (Appendix K.5.10), we refrain from making general claims about the algorithm. We present this as an observation that may be useful to the community.

B.4 Scaling up Verification

While our findings demonstrate that combining rule-based verifiers with prompt set filtering is highly effective, designing such rules and curating prompt sets across different domains remains labor-intensive. More fundamentally, this approach embeds human-designed heuristics into the RL environment, reflecting how we think rather than allowing for emergent learning. As highlighted in The Bitter Lesson[1], manually encoding human intuition tends to be an inefficient long-term strategy. This raises an intriguing question: how can verification signals be scaled effectively? Is there an equivalent of pretraining in the context of designing RL environments? We look forward to future research on silver supervision signals and the potential for self-supervised approaches in RL verification.

B.5 Latent Capabilities in Base Models

Reasoning is a latent capability in base models that has only recently been unlocked. Our analysis suggests that one possible source of this emergent thinking is human dialogue on Internet discussion forums. This raises a broader question: what other abilities exist, waiting to be elicited from the vast reservoir of human knowledge and experience embedded in pre-training data? We look forward to more detailed analyses tracing model behaviors back to their data origins, which could yield new insights and help uncover hidden capabilities within base models.

C 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 models 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 §2.2, using the MATH training set.

Table 3: Adding data with a silver supervision signal is often 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**

Result. Table 3 shows that incorporating silver-supervised data improves average performance. Adding WebInstruct data to long CoT SFT yields a substantial 510% absolute accuracy gain on MMLU-Pro-1k over using MATH alone. Furthermore, mixing MATH and WebInstruct data achieves the best average accuracy across benchmarks.

Takeaway C for SFT with Noisy Verifiable Data

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

[1] http://www.incompleteideas.net/IncIdeas/BitterLesson.html

D Potential Reasons Why Emergent Behavior is Not Observed with `Qwen2.5-Math-7B`

Our detailed analysis of RL from `Qwen2.5-Math-7B`, as presented in §5.1 and §5.2, suggests that it fails to fully replicate the training behavior of `DeepSeek-R1`. We identify the following potential causes: 1) The base model, being relatively small (7B parameters), may lack the capacity to quickly develop such complex abilities when incentivized. 2) The model might have been overexposed to MATH-like short instruction data during (continual) pre-training and annealing, leading to overfitting and hindering the development of long CoT behaviors.

E Comparison between RL from the Base Model and RL from Long CoT SFT

We compare the performance of RL from the base model and RL from long CoT SFT and find that RL from long CoT SFT generally performs better.

Setup. We compare using the base model `Qwen2.5- Math-7B`. The results of RL from the base model are from the model trained in §5.1. For RL from long CoT SFT, we adopt a setup similar to §2.2. Specifically, we choose the 7.5k MATH training set as the prompt set, curate the SFT data by rejection sampling with 32 candidate responses per prompt using `QwQ-32B-Preview`, and perform PPO using our cosine length-scaling reward with repetition penalty and our rule-based verifier, sampling 8 responses per prompt and training for 8 epochs. To adapt `Qwen2.5-Math-7B` with a pre-training context length of only 4096 tokens to long CoT SFT and RL, we multiply its RoPE (Su et al., 2024) θ by 10 times. We don't report the results of RL with classic reward from long CoT SFT since it collapses. For evaluation, we adopt our default temperature sampling setup for RL from long CoT SFT as in §A.5 and greedy decoding setup for RL from the base model as in §5.1 for the best performance. Further details about the distillation, SFT hyperparameters and RL hyperparameters can be found in Appendix K.2 & K.3 & K.5.9, respectively.

Result. Table 5 shows that, on `Qwen2.5-Math-7B`, RL initialized from the long CoT SFT model significantly outperforms RL from the base model and further improves upon the long CoT SFT itself. Specifically, RL from long CoT SFT with our cosine reward surpasses RL from the base model by a substantial 8.7% on average and improves over the SFT initialization by 2.6%. Notably, simply applying SFT with long CoT distilled from `QwQ-32B-Preview` already yields strong performance.

F Long CoT Patterns in Pre-training Data

Based on the results in §5.1, we hypothesize that incentivized behaviors, such as the model revisiting its solutions, may have already been partially learned during pre-training. To examine this, we employed two methods to investigate whether such data are already present on the web.

Firstly, we used a generative search engine Perplexity.ai to identify webpages explicitly containing problem-solving steps that approach problems from multiple angles or perform verification after providing an answer. The query we used and the examples we identified are in Appendix L.1).

Secondly, we used `GPT-4o` to generate a list of phrases that are characteristic of the "aha moment" (Appendix L.2.1), then used the MinHash algorithm Broder (1997) to search through OpenWebMath Paster et al. (2023), a dataset filtered from the CommonCrawl Rana (2010) frequently used in pre-training. We found that there was a significant number of matches in discussion forum threads, where the dialogue between multiple users showed similarity to long CoT, with multiple approaches being discussed along with backtracking and error correction (Appendix L.2.2). This raises the intriguing possibility that long CoT originated from human dialogue, although we should also note that discussion forums are a common source of data in OpenWebMath.

Based on these observations, we hypothesize that RL primarily guides the model to recombine skills it already internalized during pre-training towards new behaviors to improve performance on complex problem-solving tasks. Given the broad scope of this paper, we leave a more in-depth investigation of this behavior to future work.

G 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. OpenAIs 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 RLs 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.

H FIGURES AND TABLES

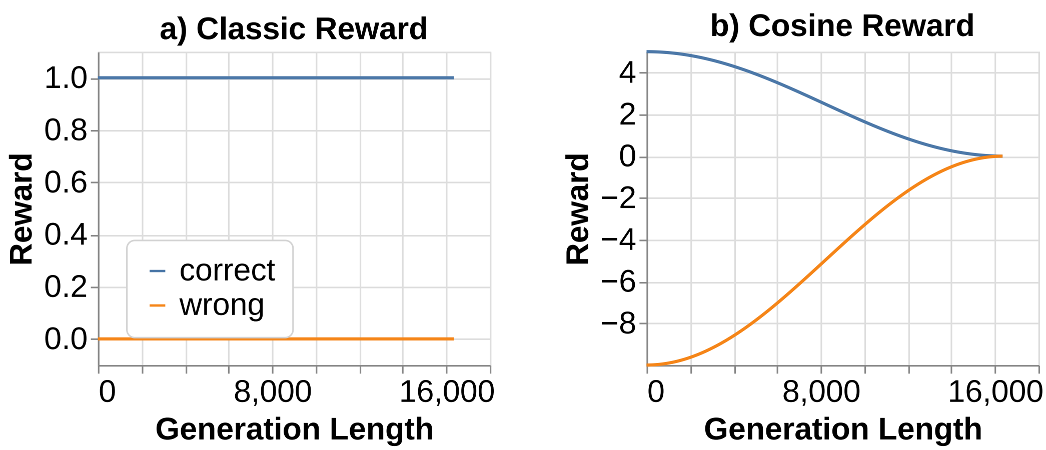
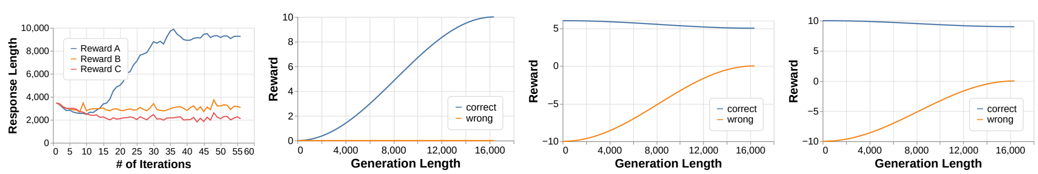


Figure 8: The Classic and Cosine Reward functions. The Cosine Reward varies with generation length.



(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 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: $r_0^c = 10, r_L^c = 9, r_0^w = -10, r_L^w = 0$.

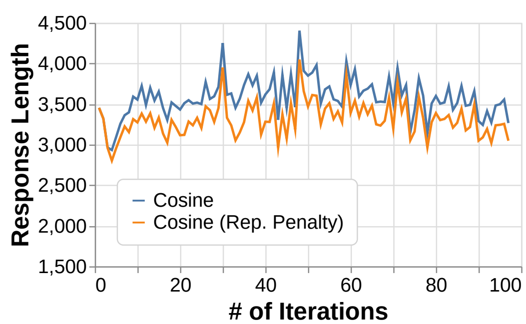


Figure 10: Training response length of models trained with Cosine Reward with and without repetition penalty. We see that repetition penalty reduced the length.

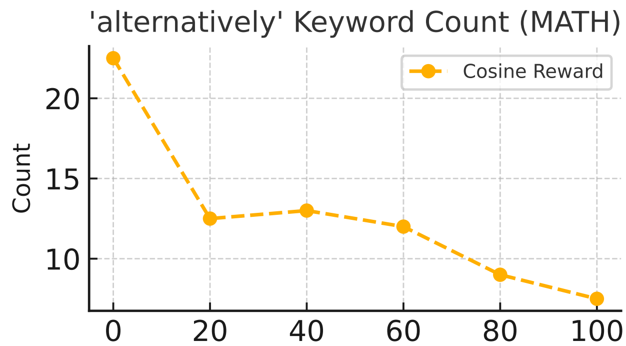


Figure 11: CoT branching frequency, estimated by the keyword count of the pivot word ”alternatively,”, decreased under the Cosine Reward with more training compute. We attributed this, along with increased repetition, to reward hacking.

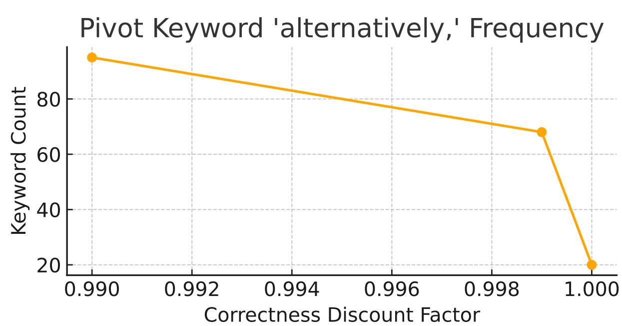


Figure 12: Branching frequency in CoT at different γ_c values. Lowering the discount factor increased branching frequency, causing the model to abandon problem-solving approaches more quickly.

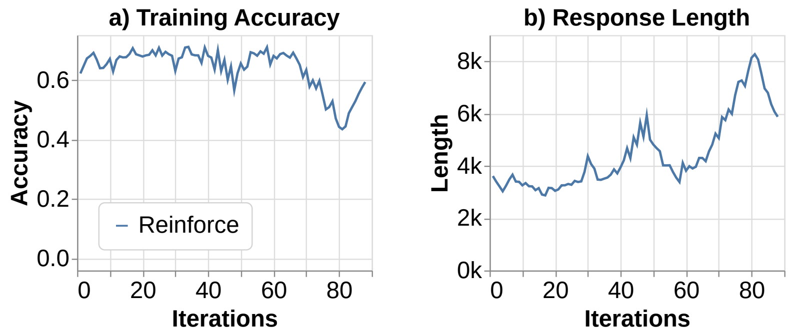


Figure 13: Reinforce with classic reward shows signs of training instability.

Table 4: 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

Table 5: Performance of different models based on `Qwen2.5-Math-7B`. The SFT data here is distilled with rejection sampling from `QwQ-32B-Preview`.

Setup	MATH 500	AIME 2024	Theo. QA	MMLU Pro-1k	AVG
Base (0-shot)	52.0	13.3	17.1	2.4	21.2
(Direct) RL	77.4	23.3	43.5	19.7	41.0
SFT	84.0	24.4	42.2	38.5	47.3
SFT + RL	**85.9**	**26.9**	**45.4**	**40.6**	**49.7**

I Algorithms and Formulas

I.1 Cosine Reward Formula

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

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 $L_{\text{gen}} = L_{\text{max}}$,

r_e : Exceed length penalty,

Inputs:

C : Correctness (0 or 1),

L_{gen} : Generation length.

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

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

I.2 N-gram Repetition Penalty

Algorithm 1 N-gram Repetition Penalty

1: **Input:**
2: s : sequence of tokens
3: l : sequence length
4: N : n-gram size
5: P : penalty value
6: m : maximum sequence length
7: **Output:** $r \in \mathbb{R}^m$
8: $seq \leftarrow s[1:l]$ {Extract subsequence of length l}
9: $ngrams \leftarrow \emptyset$ {Set of observed n-grams}
10: $r \leftarrow \vec{0} \in \mathbb{R}^m$ {Initialize reward vector}
11: **for** $j \leftarrow 1$ to $|seq| - N + 1$ **do**
12: $ng \leftarrow (seq[j], seq[j+1], ..., seq[j+N-1])$ {Current n-gram}
13: **if** $ng \in ngrams$ **then**
14: **for** $t \leftarrow j$ to $j + N - 1$ **do**
15: $r[t] \leftarrow P$ {Apply penalty}
16: **end for**
17: **end if**
18: $ngrams \leftarrow ngrams \cup \{ng\}$
19: **end for**
20: **Output:** r

J 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.

K Experimental Setup

K.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 14 shows the distribution before/after the downsampling.

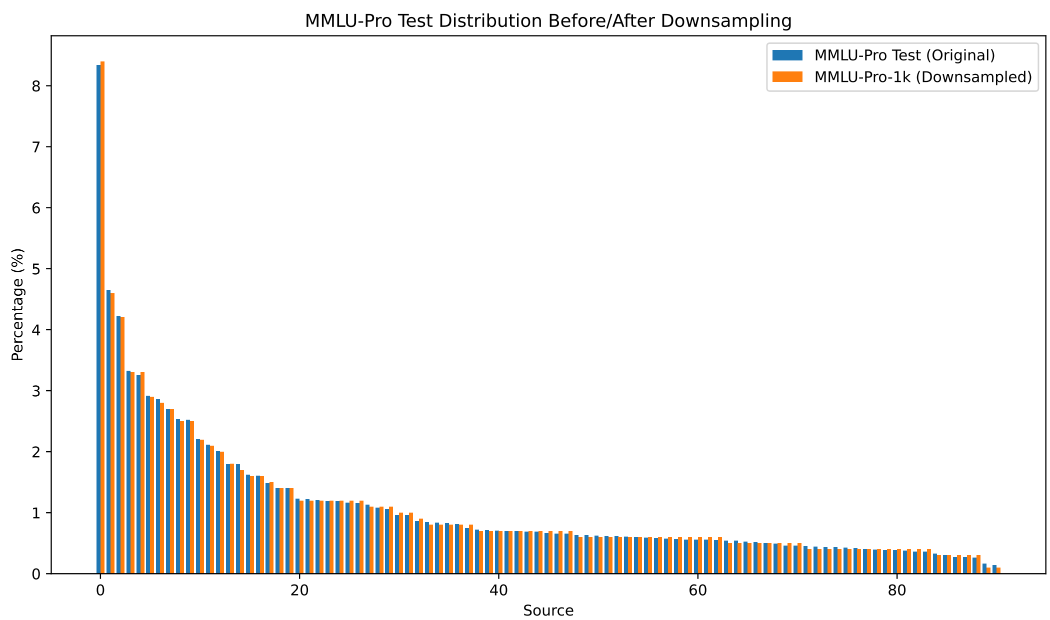


Figure 14: MMLU-Pro test distribution before/after downsampling for the MMLU-Pro-1k subset. The subset is i.i.d. to the full set.

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[2], an elaborate answer grader capable of processing complex mathematical objects like matrices and functions, keeping consistent with the sampling and reward 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.

[2]`https://github.com/tongyx361/symeval`

K.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.

K.3 Details about SFT Setup

We use OpenRLHF (Hu et al., 2024) for our SFT experiments. By default, we adopt the SFT hyperparameters in Table 6.

For efficiency, we utilize Flash Attention 2 (Dao, 2024) and ZeRO (Rajbhandari et al., 2020) 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 6: SFT Hyperparameters

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

K.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.

K.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 most of our RL setups. Also, the `Samples` column refers to the number of samples per prompt.

K.5.1 DETAILS OF SECTION 2.2 (SFT INITIALIZATION FOR RL)

SFT Data: CoT data distilled from `QwQ-32B-Preview` or `Qwen2.5-Math-72B-Instruct` with the MATH train split with different number of candidate responses per prompt.

Table 7: 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: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma = 1$	4	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6	0.01

K.5.2 DETAILS OF SECTION 3.1 (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

K.5.3 DETAILS OF SECTION 3.2 (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

K.5.4 DETAILS OF SECTION 3.3 (COSINE REWARD HYPERPARAMETERS)

SFT Data: 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
Llama3.1-8B	Cosine: $r^c_0 = 0$ $r^c_L = +10$ $r^w_0 = 0$ $r^w_L = 0$ $r_e = -10$ 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: $r^c_0 = +6$ $r^c_L = +5$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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: $r^c_0 = +10$ $r^c_L = +9$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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

K.5.5 DETAILS OF SECTION 3.4 (CONTEXT WINDOW SIZE)

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: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ 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

K.5.6 DETAILS OF SECTION 3.5 (LENGTH REWARD HACKING)

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: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$	$\lambda = 1$ $\gamma = 1$	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

K.5.7 DETAILS OF SECTION 3.6 (OPTIMAL DISCOUNT FACTORS)

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: $r^c_0 = +2$ $r^c_L = +1$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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: $r^c_0 = +2$ $r^c_L = +1$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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: $r^c_0 = +2$ $r^c_L = +1$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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: $r^c_0 = +2$ $r^c_L = +1$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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: $r^c_0 = +2$ $r^c_L = +1$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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: $r^c_0 = +2$ $r^c_L = +1$ $r^w_0 = -10$ $r^w_L = 0$ $r_e = -10$ 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

K.5.8 DETAILS OF SECTION 4 (RL WITH NOISY VERIFIABLE DATA)

SFT Data: 115k filtered from 462k instances of long CoT data distilled from `QwQ-32B-Preview` with WebInstruct.

Table 14: 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 instances	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 instances	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 KL: 0.01
Llama3.1-8B	Filtered (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 instances	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 KL: 0.01
Llama3.1-8B	Filtered (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 instances	4	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 9e-6 KL: 0.01

K.5.9 DETAILS OF SECTION 5 (EXPLORATION ON RL FROM THE BASE MODEL)

Table 15: Hyperparameters

Base Model	Rewards	GAE	Episodes	Samples	BS	Epochs	Context Length	LR	KL
Qwen2.5-Math-7B	Correct: $+1$ Wrong: -0.5 No Answer: -1	$\lambda = 0.95$ $\gamma = 1$	20	8	1024 (Train: 128)	1	Prompt: 1024 Gen: 3072	Actor: 5e-7 Critic: 9e-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
Qwen2.5-Math-7B	Cosine: $r_0^c = +2$ $r_L^c = +1$ $r_0^w = -10$ $r_L^w = 0$ $r_e = -10$ Rep. Penalty: $P = -0.05$ $N = 40$	$\lambda = 1$ $\gamma = 1$	8	8	512	1	Prompt: 2048 Gen: 14336	Actor: 5e-7 Critic: 4.5e-6	0.01

K.5.10 DETAILS OF SECTION B.3 (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

K.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>`
```

K.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{...}`.

K.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.

K.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 ← [prompt] {Initialize singleton chain of thought sequence from prompt}
state ← "clarify"
while True do
    if state = "clarify" then
        output ← prompt_action_clarify()
        (state, thought) ← parse(output)
        chain_of_thought.append(thought)
    else if state = "decompose" then
        output ← prompt_action_decompose()
        (state, thought) ← parse(output)
        chain_of_thought.append(thought)
    else if state = "solution_step" then
        output ← prompt_action_solution_step()
        (state, thought) ← parse(output)
        chain_of_thought.append(thought)
    else if state = "reflection" then
        output ← prompt_action_reflection()
        (state, thought) ← parse(output)
        chain_of_thought.append(thought)
    else if state = "answer" then
        output ← prompt_action_answer()
        (state, thought) ← parse(output)
        chain_of_thought.append(thought)
        return chain_of_thought {Terminate after answer action}
    end if
end while
```

K.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}
```

L Long CoT Patterns in Pre-training Data

L.1 Snapshot of webpages

Source: brilliant.org

The following two examples demonstrate how explicit verification after answering a question can naturally exist on a webpage.

Explicit verification

$x + 7 = 10$
This problem can be solved by subtracting 7 from each side.
$x + 7 - 7 = 10 - 7$
$x = 3$
Once the problem is solved, the solution can be verified by rewriting the problem with 3 substituted for x.
$3 + 7 = 10$
$10 = 10$
Both sides are equal, verifying that $x = 3$ is a valid solution.

Explicit verification that found an error

$x + 7 = 10$ A student rushing through her homework might mistakenly write $x = 2$ as the solution to this problem. If she takes a moment to rework the equation with her answer, she will realize the answer is incorrect.
$x + 7 = 10$
$2 + 7 = 10$
$9 = 10$
Since $9 \neq 10$, the student knows she needs to go back and find a different solution to the problem.

Source: kidswholovemath.substack.com

Attempt the question from different perspective

The Double Check Game
Regardless of the scenario, we can play the double check game!
The game is simple: we try to solve the problem in as many different ways as possible.
Elementary School Example
Math problem is: $78 - 57 = ?$
To play the game, we try to solve the problem in as many different ways as possible.
The first solution:
$? = 78 - 57$
Break apart the 57:
$? = 78 - 50 - 7$
$? = 28 - 7$
$? = 21$
A second solution:
$? = 78 - 57$
Subtract an easier number from 78:
$? = 78 - 60 + 3$
$? = 18 + 3$
$? = 21$
A third solution:
$? = 78 - 57$
Subtract 57 from an easier number:
$? = 80 - 57 - 2$
$? = 23 - 2$
$? = 21$
...

L.2 OpenWebMath

L.2.1 Queries

We used `GPT-4o` to generate examples of typical pivot keywords found in long CoT. These were used to find documents in OpenWebMath that have interesting properties characteristic of long CoT trajectories.

”Aha” Phrases

```
"Let’s think step by step."
"Let’s go through this one step at a time."
"Breaking it down step by step..."
"Thinking about it logically, first..."
"Step 1: Let’s figure out the starting point."
"If we follow the steps carefully, we get..."
"To solve this, lets analyze it piece by piece."
"Going through this systematically, we have..."
"Okay, lets solve this gradually."
"Does that make sense?"
"Is this correct?"
"Wait, does that check out?"
"Am I missing something?"
"Hmm does that work?"
"Let me verify that."
"That makes sense, right?"
"Hold on, is this right?"
"Lets double-check this."
"Wait, actually..."
"Oh, hold on..."
"Wait a second..."
"Actually, let me rethink that."
"Hmm, let me go back for a moment."
"I might need to check this again."
"Let’s pause and reassess."
"Lets check by doing the reverse."
"Let’s verify by working backward."
"Can we check this by reversing the process?"
"To confirm, let’s undo the steps."
"A good way to verify is by reversing it."
"If we undo the operations, do we get the same result?"
...
```

L.2.2 MATCHES

Source: MC Stan Discussion Forum

The discussion below took place on a message board for the probabilistic programming framework MC Stan. The user Tiny has a question about how to interpret some data and multiple other users are responding. We can see the usual pivot keywords (highlighted in **bold**) characteristic of long CoT, including branching, self-correction and even an assessment of the feasibility of an approach.

Discussion on message board

```
So the question is then to find the right prediction task, looking
    at your setup, those may include:

...

    For a hypothetical future serial drawn from the same
     population  as the observed serials. (i.e. include the
    varying intercept via a new level and sample_new_levels =
     uncertainty  )
    For the  true  or  average  underlying system (i.e.
    ignore the varying intercept)
    In the experiments you actually observed (i.e. include the
    fitted varying intercepts for your experiments)

But you could also ask other stuff, like:

    What is the expected difference in some of the constants (or
    anything else) between two future experiments?

All of those (and more) should be answerable using the posterior of
     the model. But you still need to figure out which questions do
    you actually want to ask, as there is a lot of options

Does that make sense?

Best of luck with your model!

...

I am not sure I follow your thought here, but maybe thats just
    because I would have worded it differently?

...

An alternative approach would be to
    try to find a different parametrization of the model where the
    parameters are interpretable separately, but that might be
    hard.

Also, if this is the parametrization of the process used by many in
     the field, than maybe poeple would expect you to report as (\
    frac {L} {mol})^{n-1} s^{-1}, because that s what everybody
    has been doing (although possibly with fixed n)?

Does that make sense?

Can you not just recast the model (with modified parameters) as

...
```

Source: physicsforums.com

The discussion below took place on a physics forum. The user Songoku is asking for help with homework and another user BvU is trying to assist without revealing the solution directly. We see the usual pivot keywords indicating self-reflection, expression of uncertainty and formulation of hypotheses.

Discussion on a physics forum

```
# Cylinder in 3 D
1. Dec 13, 2017

### songoku
1. The problem statement, all variables and given/known data
Let r be a positive constant. Consider the cylinder x2 + y2 <= r2,
    and let C be the part of the cylinder that satisfies
0 <= z <= y.
(1) Consider the cross section of C by the plane x = t (-r <= t <=
    r), and express its area in terms of r, t.
(2) Calculate the volume of C, and express it in terms of r.

...

5. Dec 13, 2017
### BvU
Simple case: x = 0. So -1 <= y <= 1. In the yz plane 0 <= z <= y is
     a triangle.
What about y ?

6. Dec 13, 2017
### songoku
I think I am missing something
    here because I feel I can't really grasp the hint given.
Let me start from the basic again:
1. Let the x - axis horizontal, y - axis vertical and z - axis in /
     out of page. I imagine there is circle on xy plane with radius
    r then it extends out of page (I take out of page as z+) to form
     3 D cylinder. Is this
    correct?
2. Plane x = t is like the shape of a piece of paper hold
    vertically with the face of paper facing x - axis (I mean x -
    axis is the normal of the plane). Is this
    correct?
Thanks

7. Dec 14, 2017
### BvU
Yes

8. Dec 14, 2017
### songoku

"Consider the cross section of C by plane x = t" means plane x = t
    cuts the cylinder?
And the intersection will be rectangle?
...
```

Source: StackExchange

The user Baymax is asking for help on a probability problem and we see dialogue with another user Lulu. We see that the quick back-and-forth between them is similar to the kind of nimble branching behavior in long CoT where multiple solutions are quickly assessed and considered. We also see an expression of realization which can be easily re-cast as self-verification in a long CoT.

Discussion on Stack Exchange

```
# probability that we stop flipping after exactly ten flips in a
    biased coin flipping?

...

I thought that let us fix of getting a third head at last that is
    at 10th flip, so that we would stop there, and the remaining -
    getting two heads can be accommodated in the 9 trials. so there
    are $$9$$ choose 2 ways of getting two heads so the probability
    that we stop flipping after exactly ten flips is $$^9C_{2}$$ .
    $$\frac{1}{4}^3$$.$$\frac{3}{4}^7$$. Is this
    correct?

EDIT - Now the probability of getting exactly 3 heads? I got it to
    be $$^{10} C_{3} \frac{1}{4}^3 \frac{3}{4}^7$$. Should we get
    the same as the previous one? any reason why they should/should
    not be same?

    I think you switched $P(H),P(T)$ but the approach is
    good.     lulu Oct 1 '18 at 16:13
    oh i see now! thanks!     BAYMAX Oct 1 '18 at 16:13
    @lulu please see the edit     BAYMAX Oct 1 '18 at 16:30
    Your probability for exactly 3 heads is right as well. It
    should be obvious why the results have to be different. In the
    first case the outcome of the last flip is fix and in the second
     case the outcome of the last flip is not fix.     callculus Oct
     1 '18 at 16:31

...
```

Source: StackExchange

User88 interacts with multiple other users. Observe that they are helping to clarify each others' doubts, which is reminiscent of self-correction in long CoT trajectories.

Discussion on Stack Exchange

```
Choosing units for drug testing

Here's a third puzzle that I found in a book, slightly paraphrased
    because I don't entirely remember the format of the original.

...

How can he arrange the dosage amounts so that he ends up using all
    25 test packages, and the total units of dosage used in the
    tests are as low as possible?

The book had the answer, but one, it didn't explain how the answer
    was arrived at, and two, I don't remember what the answer was
    and no longer have that book with me.

    Am I missing something, or is the goal just to find 25 coprime
    numbers from 25 to 50?     Aza May 20 '14 at 4:33
    They don't have to be coprime. There just can't be any two
    where one is a factor of the other. And the range is from 1 to
    50, not 25 to 50.     Joe Z. May 20 '14 at 4:34
    Wouldn't a single
    test of 1 unit technically satisfy the requirement? Or am I
    missing something?  Ah, I guess you have
    to perform exactly 25 tests.     arshajii May 20 '14 at 14:28
    Yea. Wouldn't 1 win?     awesomepi May 20 '14 at 19:24
    You have to use all 25 tests.     Joe Z. May 20 '14 at 19:31

By logically starting from 26-50 and trying to shrink them one by
    one you can easily show: $8
    ,12,14,17,18,19,20,21,22,23,25,26,27,29,30,31,33,35,37,39,
    41,43,45,47,49$

Which equals $711$

...
```