

Efficient Long CoT Reasoning in Small Language Models

Zhaoyang Wang¹ * Jinqi Jiang² * Tian Qiu³ * Hui Liu⁴ Xianfeng Tang⁴ Huaxiu Yao¹

¹University of North Carolina at Chapel Hill

²Huazhong University of Science and Technology

³Fudan University ⁴Amazon

{zhaoyang, huaxiu}@cs.unc.edu

Abstract

Recent large reasoning models such as DeepSeek-R1 exhibit strong complex problems solving abilities by generating long chain-of-thought (CoT) reasoning steps. It is challenging to directly train small language models (SLMs) to emerge long CoT ability. Thus, distillation becomes a practical method to enable SLMs for such reasoning ability. However, the long CoT often contains a lot of redundant contents (e.g., overthinking steps) which may make SLMs hard to learn considering their relatively poor capacity. To address this issue, we propose a simple-yet-effective method to prune unnecessary steps in long CoT, and then employ an on-policy method for the SLM itself to curate valid and useful long CoT training data. In this way, SLMs can learn more efficient long CoT reasoning and preserve competitive performance at the same time. Experimental results across a series of mathematical reasoning benchmarks demonstrate the effectiveness of the proposed method in distilling long CoT reasoning ability into SLMs, maintaining competitive reasoning performance while significantly reducing redundant reasoning steps.

1 Introduction

Chain-of-thought (CoT) prompting has become the de-facto technique for eliciting multi-step reasoning in both large and small language models [22, 23, 21]. Recent breakthroughs such as OpenAI’s o1 [15], DeepSeek-R1 [5], and Kimi-k1.5 [20] show that lengthening the CoT trace can further unlock reasoning performance on challenging tasks. While these efforts extend the boundaries of what LLMs can achieve, they also introduce new challenges to small language models (SLMs) with about 7B parameters which often use distillation methods to learn such long CoT reasoning [5, 4].

While long CoT is necessary in scaling the performance of reasoning, its increasing length introduces significant computational inefficiency. The generated long CoT traces often contain many redundant reasoning steps even to the very simple question such as “1+1=?”, as shown in Figure 1. Those redundant reasoning steps may not only bring unnecessary

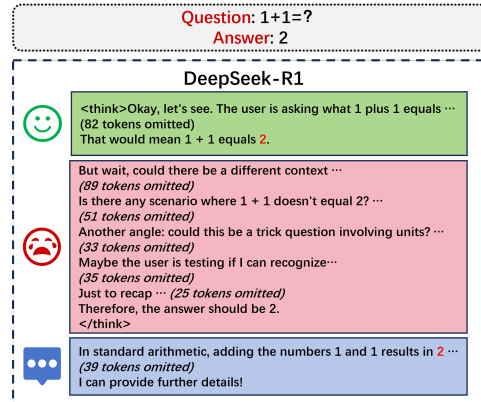


Figure 1: Example of redundant reasoning to a simple question by DeepSeek-R1 [5]

* Equal contribution.

computation burden during test time, but also affect the reasoning performance [19, 1, 24, 11]. Moreover, long CoT with redundant reasoning steps can make barriers to the distillation process, since SLMs have relatively poor capability and generalization to such complex and lengthy reasoning pattern.

Existing works propose to use heuristic rules such as minimum reasoning length with correct final answer [2], design length based rewards for reinforcement learning [1, 27, 26], or advanced prompting methods [24, 13, 25, 6, 14]. However, these methods either rely on re-designing the rewards during reinforcement learning which is often less effective than direct distillation from larger reasoning models [5, 4] and requires more computation resources, or do not consider the target SLM’s reasoning ability when selecting the long CoT training data. This leads to the question: *How can CoT traces generated by large reasoning models be efficiently distilled into SLMs?*

In this paper, we first observed that SLMs may not need to learn the entire CoT reasoning process, but rather focus on the essential reasoning steps that lead to the correct answer. Then, we propose a simple binary cutting method to prune redundant reasoning steps in the long CoT, which searches the shortest concise CoT steps that lead to the correct answer while maintaining acceptable search complexity. Further, we introduce an on-policy distillation method to help the aforementioned search method find tailored long CoT steps for the target SLM. Finally, we use those pruned long CoT data to fine-tune the target SLM with supervised fine-tuning (SFT) and direct preference optimization (DPO) [17]. In summary, our contributions are three-fold:

- 1) We observed that long CoT data generated by large reasoning models often contains unnecessary reasoning steps, which can be harmful for direct distilling into SLMs.
- 2) We propose a simple yet effective method to prune redundant reasoning steps in long CoT, which uses binary cutting for efficiently searching and on-policy validation that tailors to the target SLM’s own capability.
- 3) Experiments and analysis demonstrate the effectiveness of our method in enabling SLMs with efficient long CoT reasoning, which significantly reduces generating redundant reasoning steps while preserving performance.

2 Method

2.1 Background

Large reasoning models (e.g., DeepSeek-R1) are capable of generating long CoT reasoning steps to solve complex problems. However, these models usually produce excessive reasoning steps, where many of them are devoted to repeatedly verifying or reconfirming the correctness of an already correct answer. This behavior leads to unnecessarily long outputs and may also be harmful for the reasoning performance [11].

The goal of this paper is to enable open-source SLMs to take advantage of long CoT reasoning ability while maintaining efficiency. To achieve this, most of existing works introduce length based penalties into the reward function [1, 26, 27], then train SLMs with similar reinforcement learning adopted by DeepSeek-R1 [5]. However, this approach is often less effective than direct distillation from larger reasoning models [5] in terms of both performance and training efficiency. Thus, we choose to search segments of the long CoT reasoning that meet two constraints: (i) *valid*, it should be sufficient for the target SLM to generate the correct final answer, and (ii) *efficient*, it should be as short as possible to maximize efficiency. Then, we can use those long CoT segments to fine-tune the target SLM to achieve efficient long CoT reasoning.

In the following sections, we will first present the curation of streamlined long CoT data which includes the proposed binary cutting method and on-policy validation method, and then detail the fine-tuning process for the target SLM. The whole framework is illustrated in Figure ??.

2.2 Streamlining Long CoT

2.2.1 Response Sampling

In this paper, we investigate the long CoT data generated by DeepSeek’s R1, where a model response typically contains two parts: (1) an intermediate reasoning (i.e., long CoT) segment enclosed in

<think>...</think> tags, and (2) a summary segment that delivers the final answer. And individual reasoning steps are naturally separated by two consecutive line breaks “\n\n” [5].

As shown in Figure ??, to collect such data, we adopt simple response sampling from DeepSeek R1: for an input question Q , a full response R is the concatenation of the intermediate reasoning part T and the final-answer summary. We define a reasoning step s as a single unit (delimited by “\n\n”), yielding $T = [s_1, \dots, s_n]$. Together with the ground-truth answer A for the input question Q , this gives the original datasets $D_{\text{original}} = \{(Q_i, T_i, A_i)\}_{i=1}^N$. Our goal is, for each data i , to search and extract a contiguous subsequence of steps $T_i^{j:k} = [s_j, \dots, s_k]$ that maximizes reasoning efficiency (reducing unnecessary steps) while guaranteeing that the generated final answer remains correct.

2.2.2 Binary Cutting

Previous work [2] proposes First-Correct Solutions (FCS), which truncates the reasoning process to the minimal prefix yielding a correct answer. FCS requires checking every prefix $T^{1:k} = [s_1, \dots, s_k]$ for $k = 1, \dots, n$. While this linear search requires $O(n)$ complexity, an unconstrained search over all 2^n subsequences would be intractable and break the reasoning chain’s coherence. To maintain contiguity yet improve efficiency, we likewise restrict our search to prefixes, but replace the linear search with a binary cutting strategy. This reduces the computation complexity to $O(\log_2 n)$ while still guaranteeing the discovery of the minimal valid prefix. Specifically, at each iteration we compute $m = \lfloor (low + high)/2 \rfloor$, truncate the original reasoning chain $T = [s_1, \dots, s_n]$ to the prefix $T^{1:m}$, and invoke the validation function $\phi(Q, T^{1:m}, A)$ which decides whether the segment is good and valid. If the prefix still yields the correct answer ($\phi = 1$), we update $best \leftarrow T^{1:m}$ and set $high \leftarrow m$ to search even shorter prefixes. However, if the prefix fails to yield the correct answer ($\phi = 0$), we introduce a backtracking mechanism to recover the last valid prefix: resetting low to the last mid-point and perform a binary search upward toward n , computing $m = \lceil (low + n)/2 \rceil$ at each step until ϕ returns true again. The proposed binary cutting method with backtracking mechanism first aggressively cutting, which guarantees that the selected steps form a contiguous prefix, and recovers any essential steps that might have been over-pruned. In contrast, as 61.83% of the samples contain the answer in the last 10 steps, cutting from the beginning is not effective for SLM training. This approach effectively reduces the search space and time complexity to $O(\log_2 n)$, while ensuring that the selected reasoning steps are both valid and efficient.

2.2.3 On-Policy Validation

Existing pruning methods such as FCS assume a single “oracle” validation criterion, typically provided by an additional judge model, to judge whether a truncated CoT remains correct. However, this ignores the fact that different SLMs exhibit distinct reasoning biases and strengths. To generate training data that is tailored to the target SLM’s own inductive preferences, we let the SLM M itself serve as the validator in an on-policy paradigm. Specifically, we construct a specialized prompt P_{policy} (see Figure ??, part (2)) which asks model M to produce the final answer given only the question Q and a candidate prefix $T^{1:k}$. We then can define the validation function ϕ :

$$\phi(Q, T^{1:k}, A; M) = \mathbf{1}\{M_t(Q, T^{1:k}, P_{\text{policy}}) = A\}. \quad (1)$$

During binary cutting, each prefix is accepted only if $\phi = 1$ under M . By relying on the target model’s own outputs rather than an external judge model, we ensure that the distilled CoT segments align with the SLM’s native reasoning capacity. This on-policy mechanism overlooked by prior methods yields a more coherent long CoT segment, since each retained prefix is one that the SLM can already interpret correctly. The whole method for streamlining long CoT is illustrated in Appendix ?. See Appendix C for the full on-policy prompt.

2.3 Fine-tuning SLM

After applying binary cutting and on-policy validation methods, we obtain a distilled dataset $D_{\text{distill}} = \{(Q_i, R_i, Y_i)\}_{i=1}^N$, where R is the original whole response, and Y is the pruned concise response containing the pruned thinking part and the final response part. We can use this dataset D_{distill} to fine-tune SLM to learn efficient long CoT reasoning via supervised fine-tuning (SFT) and direct preference optimization (DPO) [17].

2.3.1 SFT Training

The most straightforward approach to leverage the obtained distilled data is to apply SFT training on the target model. Given the distilled dataset D_{distill} , the target SLM M is fine-tuned to maximize the likelihood of the pruned reasoning R conditioned on the input question Q as follows:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(Q,Y) \sim D_{\text{distill}}} \log M(Y|Q). \quad (2)$$

After training, the SLM is expected to generate more concise and efficient reasoning steps while maintaining the correctness of the final answer.

2.3.2 DPO Training

To help the target model better distinguish between “good” and “bad” reasoning steps, we can leverage preference learning methods such as DPO to further fine-tune the model. Here, “good” refers to the pruned response with concise reasoning steps, while “bad” refers to the original response with redundant reasoning steps. The DPO training objective can be formulated as follows:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(Q,R,Y) \sim \mathcal{D}_{\text{distill}}} \left[\log \sigma \left(\beta \log \frac{M(Y|Q)}{M_{\text{ref}}(Y|Q)} - \beta \log \frac{M(R|Q)}{M_{\text{ref}}(R|Q)} \right) \right], \quad (3)$$

where $\sigma(\cdot)$ denotes the logistic function, $\beta = 0.1$ is a hyperparameter of DPO, and M_{ref} is the frozen reference model typically the SLM after SFT training. Thanks to the significant difference between two responses R and Y in terms of the response length, the DPO training can effectively help the target SLM to learn the preference of concise reasoning steps over redundant ones.

3 Experiments

3.1 Experimental Setup

Datasets. To evaluate not only the reasoning performance but also the efficiency of long CoT reasoning, we benchmark SLMs on three mathematical reasoning datasets of increasing difficulty: GSM8K [3], MATH [7], and AIME [10]. GSM8k is a primary school level mathematical dataset requiring basic arithmetic and logic. MATH we used is a widely used subset of original dataset which contains 500 challenging high school competition-level math problems. AIME consists of extremely difficult math problems spanning from 1983 to 2025, aiming to test model’s generalization ability.

Models. We conducted experiments mainly on two popular open-source SLMs, Llama-3.1-8B-Instruct [12], Qwen2.5-7B-Instruct [16], and Qwen2.5-14B-Instruct [16], which do not originally own the long CoT reasoning ability.

Baselines Methods. In experiments, we mainly compare our method with the following baseline methods: (1) “Base”, using the original SLM, as the baseline performance. (2) “Full”, as described in Guo et al. [5], we directly use the original long CoT which may contain redundant reasoning steps for training, in order to directly show the necessity to remove such redundant steps. (3). “Short CoT”, using the normal CoT data without scaling length to train the SLM, in order to demonstrate the effectiveness of scaling CoT. (4). “FCS”, First-Correct Solutions strategy [2] which linearly search the first occurred segment of long CoT with the correct final answer, in order to demonstrate the superiority of our method in pruning unnecessary reasoning steps and enhancing long CoT of SLMs. Following original implementations, we use a LLM Qwen2.5-14B-Instruct [16] to segment the thinking part instead of nature line break.

Implementation Details. We use a long CoT data D_{original} OpenR1-Math-220k [4]², which has 220k math problems from NuminaMath 1.5 [8], each paired with two to four long CoT reasoning generated by DeepSeek-R1 [5]. To form the distilled dataset, we perform binary cutting search with on-policy validation on the “train” split of this dataset which has about 93.7k samples. During this process we (1) discard any instance for which the model still fails to reach the correct answer even when the full chain is provided, (2) remove chains longer than 200 steps, and (3) constrained by

²<https://huggingface.co/datasets/open-r1/OpenR1-Math-220k>

limited compute resources retain only those whose token count after cutting is below 4,096. After these filters we obtain the pruned long-CoT set D_{distill} about 40k samples for training. In most cases, the target SLM is fine-tuned by SFT training in 3 epochs, and DPO training in 1 epoch, with a learning rate of $1e-6$. We also observed that DPO alone can reduce the likelihood of the preferred response; therefore we blend the SFT loss into the DPO objective for stability. The mixing weight is chosen according to model scale: 0.3 for Llama-7B and Qwen-7B (trained on 5k DPO samples), and 0.1 for Qwen-14B (trained on 10k DPO samples).

Table 1: Main Results. Note that only Base and Short CoT report the full token counts, while all other methods count only tokens within `<think>...</think>`. The subscripts of Acc and Token indicate the percentage of decline or rise of the model relative to the model with SFT_{Full} method.

Model	Method	GSM8K		MATH		AIME	
		Acc(%)	#Token	Acc(%)	#Token	Acc(%)	#Token
Llama	Base	77.33	173	36.60	374	9.97	505
	SFT _{Full}	87.11	1049	58.80	4887	11.15	11190
	SFT _{Short CoT}	60.05 _(31.1%↓)	314 _(70.1%↓)	23.40 _(60.2%↓)	1522 _(68.8%↓)	4.82 _(56.8%↓)	2269 _(79.7%↓)
	SFT _{FCS}	83.93 _(3.7%↓)	619 _(41.0%↓)	52.4 _(10.9%↓)	1587 _(67.5%↓)	9.11 _(18.3%↓)	3049 _(72.7%↓)
	SFT+DPO _{FCS}	87.57 _(0.5%↑)	598 _(43.0%↓)	50.20 _(14.6%↓)	1392 _(71.5%↓)	13.40 _(20.2%↑)	2135 _(80.9%↓)
	SFT _{Ours}	86.05 _(1.2%↓)	627 _(40.2%↓)	57.00 _(3.1%↓)	3723 _(23.8%↓)	11.25 _(0.9%↑)	8623 _(22.9%↓)
Qwen-7b	SFT+DPO _{Ours}	86.20 _(1.0%↓)	438 _(58.2%↓)	57.20 _(2.7%↓)	2006 _(58.9%↓)	15.43 _(38.4%↑)	5088 _(54.5%↓)
	Base	85.75	290	75.00	615	23.15	1504
	SFT _{Full}	89.31	1721	74.60	4398	25.29	11412
	SFT _{Short CoT}	64.67 _(27.6%↓)	125 _(92.7%↓)	43.20 _(42.1%↓)	487 _(88.9%↓)	13.61 _(46.2%↓)	987 _(91.4%↓)
	SFT _{FCS}	56.63 _(36.6%↓)	660 _(61.6%↓)	39.40 _(47.2%↓)	1064 _(75.8%↓)	17.26 _(31.7%↓)	2399 _(79.0%↓)
	SFT+DPO _{FCS}	78.24 _(12.4%↓)	524 _(69.5%↓)	49.40 _(33.8%↓)	1914 _(56.5%↓)	21.22 _(16.1%↓)	4131 _(63.8%↓)
Qwen-14b	SFT _{Ours}	89.46 _(0.2%↑)	856 _(50.3%↓)	75.20 _(0.8%↑)	2413 _(45.1%↓)	24.65 _(2.5%↓)	5655 _(50.4%↓)
	SFT+DPO _{Ours}	87.34 _(2.2%↓)	705 _(59.0%↓)	72.20 _(3.2%↓)	857 _(80.5%↓)	25.40 _(0.4%↑)	2894 _(74.6%↓)
	Base	84.69	256	74.60	730	26.15	1814
	SFT _{Full}	92.12	1203	79.20	3950	34.73	10144
	SFT _{FCS}	88.17 _(4.3%↓)	2477 _(105.9%↑)	69.40 _(12.4%↓)	6348 _(60.7%↑)	20.79 _(40.1%↓)	13651 _(34.6%↑)
	SFT _{Ours}	92.57 _(0.5%↑)	654 _(45.6%↓)	80.04 _(1.1%↑)	1523 _(61.4%↓)	34.62 _(0.3%↓)	10008 _(1.3%↓)
	SFT+DPO _{Ours}	90.60 _(1.6%↓)	309 _(74.3%↓)	83.80 _(5.8%↑)	942 _(76.1%↓)	33.44 _(3.7%↓)	2382 _(76.5%↓)

3.2 Main Results

Table 1 shows the reasoning performance and efficiency of different methods.

Scaling CoT improves reasoning. All the long CoT based methods including “Full”, “FCS” and “Ours” surpasses the model with “short CoT”, which validates the effectiveness of scaling CoT length in mathematical reasoning.

Long CoT contains redundant steps. First, we shot that, for the easy reasoning task GSM8K, “Full” requires an average of 1051 tokens for reasoning, while does not show obvious better performance than other methods which often need less than half of the generation. Figure 3 clearly shows that about 40% long CoT data can be streamlined over 50% redundant reasoning steps by our method. This supports the perspective that long CoT contains redundant steps especially for the easy tasks. Then, compared to “Full”, our method significantly decreases the number of generated thinking tokens, while only slightly decreases the reasoning performance for most cases. For example, our Llama achieves 54.00% accuracy on MATH after SFT, slightly lower than full long-CoT SFT (54.80%), while reducing average number of tokens by 29.07%. Qwen model after distillation exhibits a modest decrease in accuracy compared to “Full” on AIME dataset, that might because Qwen is strong enough on this dataset as the base model.

Our method achieves efficient long CoT. Compared to the baseline method “Full” and another long CoT pruning method “FCS”, ours often show better reasoning performance than “FCS”, while significantly increases the efficiency in long CoT reasoning. We also find that DPO training contributes the most to reducing the tokens, however, it harms the reasoning performance for hard tasks. We further stat the results when at the same on-policy condition in Table 7. For two target SLMs, Llama and Qwen, our method reduces the number of tokens to an average of 61.83% and 50.75%,

respectively. However, the FCS and Random baseline methods retain more tokens, suggesting that the proposed binary cutting method is both efficient and effective in searching valid segments of long CoT reasoning. Additionally, We evaluate the impacts of SFT loss weights in Appendix B.1.

4 Analysis

Table 2: Ablation study results.

Model	Method	GSM8K		MATH		AIME	
		Acc (%)	#Token	Acc (%)	#Token	Acc (%)	#Token
Qwen-14b	SFT _{Ours}	92.57	654	80.04	1523	34.62	10008
	SFT _{Random}	90.45 (2.3%↓)	1514 (131.5%↑)	70.00 (12.5%↓)	5035 (230.6%↑)	23.47 (32.2%↓)	11929 (19.2%↑)

4.1 Ablation Study

We conduct two types of ablation studies to evaluate the effectiveness of design choices of our method, shown in Table 2. First, we want to show that our binary cutting with backtracking mechanism can effectively identify redundant reasoning steps instead of randomly deleting. Thus, we introduce “Random” variant which randomly delete intermediate steps before applying SFT training. From the results, we can find that random deletion leads to a substantial drop in accuracy across all datasets, indicating that preserving informative reasoning steps is essential for reasoning. Second, to assess the role of our on-policy validation, we train Qwen using concise CoT generated by the Llama model. The performance is clearly inferior to our on-policy method, highlighting the importance of policy alignment between SLM’s own reasoning capacity and long CoT data. More result on 3b model are in Appendix B.2.

In addition, we compare our method with other reinforcement learning based methods [18, 9] for long CoT reasoning, results are shown in table 3. RL methods have shown promise in optimizing reasoning paths by learning from feedback signals. However, they often require extensive training and fine-tuning on specific tasks. In contrast, our approach leverages existing long CoT data and focuses on efficient pruning, making it more adaptable to various SLMs without heavy retraining. Preliminary results indicate that while RL methods can achieve high accuracy, our method maintains competitive performance with significantly lower computational overhead.

Table 3: Comparison with reinforcement learning based methods on Qwen-7b.

Model	GSM8K		MATH		AIME	
	Acc(%)	#Tok	Acc(%)	#Tok	Acc(%)	#Tok
GRPO-length	90.60	284	75.20	804	21.22	1656
o1-Pruner	90.52	1305	67.60	5539	14.15	13623
SFT _{Ours}	89.46	856	75.20	2413	24.65	5655
DPO _{Ours}	87.34	705	72.20	489	25.40	2894

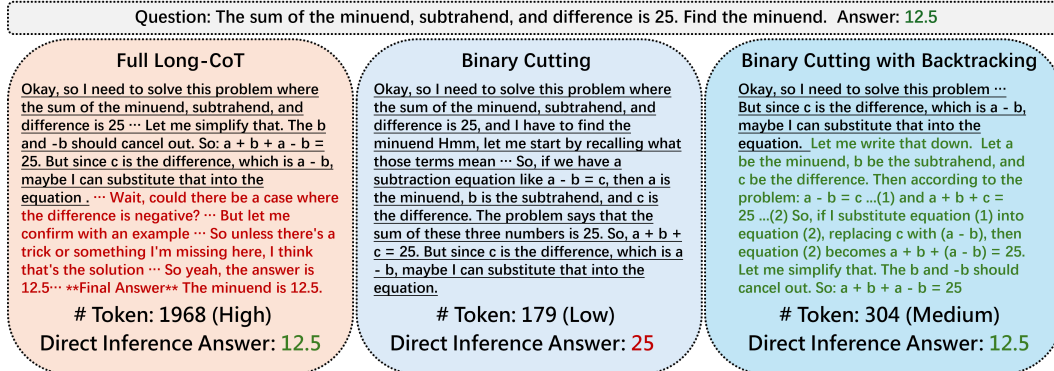


Figure 2: A case study of streamlining process of long CoT using our method. Underlined parts are the segments retained after binary cutting and are consistent across all three versions. Red text indicates steps removed during binary cutting. Green text marks reasoning steps restored in backtracking.

4.2 Quality of Streamlined CoT

We use GPT-4.1 as a LLM-as-a-Judge to automatically evaluate the quality of CoT reasoning streamlined by different methods. A reward-based scoring prompt available at Appendix C guides GPT-4.1 to rate each reasoning sample based on its correctness, completeness, conciseness and reasoning quality. We report the average scores from GPT-4.1 over 100 randomly selected examples. At the same time, we conduct human evaluation where annotators are asked to rank the four long CoT outputs into top-1 choice, middle two choice, and bottom one choice, based on overall reasoning quality and reasoning conciseness. This ranking scheme provides a coarse but interpretable assessment of human preference across different methods.

As shown in Table 4, we find that “Full” long CoT achieves the highest score due to its completeness. Among the other three streamlining methods, “Random” strategy performs poorly due to its natural drawback in logical coherence, while FCS receives lower scores in conciseness. Our method, while less complete than “Full”, achieves better balance between conciseness and logical consistency in reasoning. Human evaluations are generally consistent with the LLM-based evaluation: “Full” method is most frequently ranked at the top one due to its completeness, while our method receives significantly more top and middle rankings than the other baseline methods, thanks to its backtracking mechanism and on-policy validation.

Table 4: LLM judgment and human preference ranking on different long CoT data. **T**, **M**, **B** denote top-1, middle-2, bottom-1, respectively.

Method	LLM	T%	M%	B%
Full	4.89	61	34	5
FCS	4.20	13	57	30
Random	4.24	2	54	44
Ours	4.44	24	55	21

4.3 Case Study

Figure 2 illustrates an example of streamlining a long CoT sample by our method with binary cutting and backtracking mechanism. The original full long CoT can prompt the SLM to infer the correct answer directly, but it comes with the overthinking issue. After several rounds of binary cutting, the over-concise CoT is no longer sufficient to make SLM arrive at the correct final answer. We are surprising to find that in the backtracking stage, our method restores part of previously removed steps, which successfully guides the SLM to generate the correct answer. More examples are in Appendix D.

5 Conclusion

In this paper, we tackle the challenge of distilling long CoT reasoning from large reasoning models into SLMs. To address the issue of redundant Chain-of-Thought (CoT) in LLM reasoning, we propose a binary cutting algorithm with backtracking, which locates the shortest contiguous prefix of the original reasoning that still yields a correct final answer in only $O(\log_2 n)$ complexity. Crucially, we further introduce on-policy validation that uses the target SLM itself as the judge of whether a truncated segment of reasoning remains valid and useful for generating the final answer, thereby adapting the distilled data to the SLM’s own reasoning strengths. Extensive experiments on multiple math reasoning datasets demonstrate that our approach preserves competitive reasoning performance while significantly reducing the redundant tokens in long CoT reasoning.

References

- [1] Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.
- [2] Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for $2+3=?$ on the overthinking of o1-like llms, 2025. URL <https://arxiv.org/abs/2412.21187>.
- [3] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL <https://arxiv.org/abs/2110.14168>.

- [4] Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL <https://github.com/huggingface/open-r1>.
- [5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, and Ruoyu Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL <https://arxiv.org/abs/2501.12948>.
- [6] Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning, 2025. URL <https://arxiv.org/abs/2412.18547>.
- [7] 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, 2021. URL <https://arxiv.org/abs/2103.03874>.
- [8] Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. [<https://huggingface.co/AI-MO/NuminaMath-1.5>] (https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.
- [9] Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning, 2025. URL <https://arxiv.org/abs/2501.12570>.
- [10] MAA. American invitational mathematics examination — aime. American Invitational Mathematics Examination – AIME 2024, February 2024, February 2024. URL <https://maa.org/math-competitions/american-invitational-mathematics-examination-aime>.
- [11] Sara Vera Marjanović, Arkil Patel, Vaibhav Adlakha, Milad Aghajohari, Parishad BehnamGhader, Mehar Bhatia, Aditi Khandelwal, Austin Kraft, Benno Krojer, Xing Han Lü, et al. Deepseek-r1 thoughtology: Let’s< think> about llm reasoning. *arXiv preprint arXiv:2504.07128*, 2025.
- [12] Meta-AI. Introducing meta llama 3: The next generation of open models. <https://ai.meta.com/blog/meta-llama-3-1/>, 2024. Accessed: 2025-05-18.
- [13] Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. Self-training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*, 2025.
- [14] Sania Nayab, Giulio Rossolini, Marco Simoni, Andrea Saracino, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. Concise thoughts: Impact of output length on llm reasoning and cost, 2025. URL <https://arxiv.org/abs/2407.19825>.
- [15] OpenAI. Learning to reason with llms., 2024. URL <https://openai.com/index/learning-to-reason-with-llms>. Accessed: 2025-03-05.
- [16] Qwen-Team. Qwen2.5: A party of foundation models, September 2024. URL <https://qwenlm.github.io/blog/qwen2.5/>.
- [17] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.
- [18] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- [19] Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.

- [20] Kimi Team. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL <https://arxiv.org/abs/2501.12599>.
- [21] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=1PL1NIMMrw>.
- [22] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL <https://openreview.net/forum?id=yzkSU5zdwD>. Survey Certification.
- [23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.
- [24] Tong Wu, Chong Xiang, Jiachen T Wang, and Prateek Mittal. Effectively controlling reasoning models through thinking intervention. *arXiv preprint arXiv:2503.24370*, 2025.
- [25] Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms, 2025. URL <https://arxiv.org/abs/2502.12067>.
- [26] Junjie Yang, Ke Lin, and Xing Yu. Think when you need: Self-adaptive chain-of-thought learning. *arXiv preprint arXiv:2504.03234*, 2025.
- [27] Jingyang Yi and Jiazheng Wang. Shorterbetter: Guiding Reasoning Models to Find Optimal Inference Length for Efficient Reasoning. *arXiv*, 2025.

A Pseudo code

Algorithm 1 Streamlining Long CoT.

Require: Triplet (Q, T, A) where $T = [s_1, \dots, s_n]$, target model M , Validate function ϕ
Ensure: Shortest valid contiguous long CoT segment $T^{1:k}$

```

1:  $low \leftarrow 1, \quad high \leftarrow n, \quad best \leftarrow T^{1:n}$ 
2: while  $low < high$  do
3:    $mid \leftarrow \lfloor (low + high)/2 \rfloor$ 
4:    $T' \leftarrow T^{1:mid}$ 
5:   if  $\phi(Q, T', A, M)$  is true then
6:      $best \leftarrow T'$ 
7:      $high \leftarrow mid$  ▷ can still shorten
8:   else
9:     break ▷ enter backtracking
10:  end if
11: end while
12:  $low \leftarrow mid, \quad high \leftarrow n$  ▷ Backtracking: recover any over-pruned steps
13: while  $low < high$  do
14:    $mid \leftarrow \lceil (low + high)/2 \rceil$ 
15:    $T' \leftarrow T^{1:mid}$ 
16:   if  $\phi(Q, T', A, M)$  is true then
17:     return  $T'$  ▷ found minimal valid prefix
18:   else
19:      $low \leftarrow mid + 1$ 
20:   end if
21: end while
22: return  $best$ 

```

B More Experiment Details

B.1 DPO Training Settings

Table 5 presents the hyperparameters used in DPO training under different settings, including loss weights and data configurations.

Table 5: Experimental results of DPO with different SFT loss weights and data sizes.

Model+Method	SFT _{weight} / #Data	GSM8K		MATH		AIME	
		Acc (%)	#Token	Acc (%)	#Token	Acc (%)	#Token
Llama-7b+DPO	0.1 / 5K	85.97	332	60.20	1557	13.50	3906
	0.2 / 5K	86.88	350	59.60	1787	13.50	4335
	0.3 / 5K	86.73	386	56.20	1887	13.18	4726
	0.1 / 10K	85.90	434	58.00	1890	13.72	4604
	0.2 / 10K	86.66	424	58.40	1916	13.83	5059
	0.3 / 10K	86.20	438	57.20	2006	15.43	5088
Qwen-7b+DPO	0.1 / 5K	86.50	714	68.80	445	18.65	1234
	0.2 / 5K	86.43	719	73.60	696	20.69	2011
	0.3 / 5K	87.34	705	72.20	857	25.40	2894
	0.1 / 10K	87.41	598	72.60	907	22.08	3090

B.2 Results on tiny model

After experimenting with the Llama-3.2-3b-instruct, we observed that the proposed method fails. We attribute this to two main factors: (1) the 3B model’s capacity is too limited, and (2) our training set is insufficient—only about 40 k effective examples are available. Detailed results are shown in Table 6.

Table 6: 3b model results. We also conducted experiments on Llama-3.2-3b-instruct

Model	Method	GSM8K		MATH		AIME	
		Acc (%)	#Token	Acc (%)	#Token	Acc (%)	#Token
Llama-3b	Base	67.85	158	40.00	442	16.93	782
	SFT _{Ours}	47.38	1440	22.20	3075	1.82	8879

B.3 Distribution of remaining tokens

Figure 3 shows the percentage distribution of remaining tokens when streamlined by two target models. About 40% CoT data contains over 50% redundant reasoning segments that can be removed by our method.

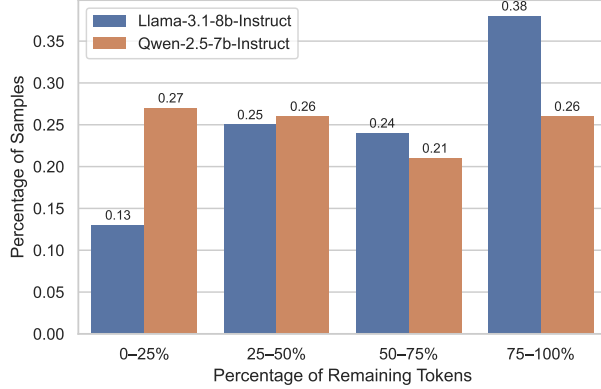


Figure 3: Distribution of the remaining tokens ratio across different percentage intervals after streamlining.

B.4 Performance of streamling method

Table 7 compares the remaining tokens of different streamling methods. Our method reduce more redundant tokens than the FCS and Random baseline methods, which shows that our binary cutting method is both effective and efficient in finding the valid reasoning segments in long CoT.

Table 7: Average token usage and remaining ratio by different streamlining methods. Note that the discrepancy in original token counts arises because, to ensure comparable training data across methods, only data deemed valid by the on-policy method is reused by FCS.

Method	Model	#Orig	#Remaining	Ratio
FCS	Llama	3659.45	2531.68	69.18%
	Qwen	3875.14	2695.19	69.55%
Random	Llama	4665.98	2333.51	50.01%
	Qwen	4919.19	2466.74	50.15%
Ours	Llama	3660.35	2263.17	61.83%
	Qwen	3875.96	1967.19	50.75%

C Prompt Examples

We provide representative prompt examples used in different stages of our pipeline, including on-policy generation and evaluation with LLM-as-a-Judge.

On-policy Prompt. Figure 4 shows a typical prompt used to generate a final answer directly.

LLM-as-a-Judge Prompt. Figure 5 shows how we guide the LLM to score CoT based on predefined rules.

On-policy answering prompt

Given a mathematical problem and a series of thinking steps, your task is to determine the final answer based on the provided steps.

Instructions:

- Follow the given thinking steps exactly to derive the answer.
- Do not add any explanations, reasoning, or modifications.
- Output only the final answer in the specified format.

Formatting Requirement:

- The output should be strictly formatted as:

###Answer: <final_answer>

Now, here is the problem: {question}

The thinking steps are given below: {thinking_steps}

Figure 4: On-policy answering prompt format.

LLM-as-a-judge prompt

Review the user’s question and the corresponding chain-of-thought (CoT) reasoning using the additive 5-point scoring system described below. Points are accumulated based on the satisfaction of each criterion:

- Add 1 point if the CoT is relevant and initiates a reasoning process related to the user’s inquiry, even if it is incomplete or contains some irrelevant steps.
- Add another point if the CoT addresses a substantial portion of the reasoning needed for the question, but does not completely resolve it or contains gaps in logic.
- Award a third point if the CoT outlines the basic logical structure needed to solve the problem, demonstrating generally sound reasoning without unnecessary tangents.
- Grant a fourth point if the CoT is clearly and logically structured, with each step building on the last. The reasoning should be easy to follow, focused on solving the user’s question, and reasonably efficient, though not necessarily optimal in brevity or precision. It’s acceptable to stop the reasoning early if the remaining steps are obvious or strongly implied.
- Bestow a fifth point if the CoT provides a complete, correct, and concise line of reasoning that leads efficiently to the final answer. It should include all necessary intermediate steps and be of appropriate length—neither too brief to omit key logic nor too long to include redundant or tangential content—reflecting expert-level clarity, precision, and efficiency.

User: {question}

<response>{response}</response>

After examining the user’s instruction and the response:

- Briefly justify your total score, up to 100 words. Focus on relevance, logical completeness, clarity, and whether only the necessary reasoning steps were included without redundancy.
- Conclude with the score using the format: “Score: <total points>”

Remember to assess from the AI Assistant perspective, utilizing web search knowledge as necessary. To evaluate the response in alignment with this additive scoring model, systematically attribute points based on the outlined criteria.

Figure 5: LLM-as-a-judge prompt format.

D Examples of Streamlined CoT

We show qualitative examples of how our method removes redundant reasoning steps while preserving essential logic and the final answer. The green part is the CoT after binary cutting, the blue part marks the restored steps in backtracking, while the red part is removed redundant part.

Question

A normal magic square of order n is an arrangement of the integers from 1 to n^2 in a square such that the n numbers in each row, each column, and each of the two diagonals sum to a constant, called the *magic sum* of the magic square. Compute the magic sum of a normal magic square of order 8.

Response of DeepSeek-R1

Okay, so I need to find the magic sum of a normal magic square of order 8. Hmm, let me think. A magic square is a square grid where the numbers in each row, column, and both main diagonals add up to the same number, called the magic sum.

Since it's a normal magic square, it uses the numbers from 1 to n^2 , where n is the order. In this case, n is 8, so the numbers go from 1 to 64.

First, I remember there's a formula for the magic sum of a normal magic square. Let me try to recall. I think it involves the total sum of all the numbers in the square divided by the number of rows (or columns), since each row needs to add up to the magic sum. Let me check if that makes sense.

The total sum of numbers from 1 to n^2 can be calculated using the formula for the sum of the first m integers: $\frac{m(m+1)}{2}$. Here, m is n^2 , so the total sum would be $\frac{n^2(n^2+1)}{2}$. Then, since there are n rows, each row must add up to the total sum divided by n . So the magic sum S should be $\frac{n^2(n^2+1)}{2n}$. Simplifying that, we get $S = \frac{n(n^2+1)}{2}$.

For example, a 3x3 magic square. The magic sum should be 15. Plugging $n = 3$ into the formula: $\frac{3(3^2+1)}{2} = \frac{3(9+1)}{2} = \frac{3 \times 10}{2} = 15$. That checks out. What about a 4x4? The magic sum should be 34. Let's see: $\frac{4(4^2+1)}{2} = \frac{4(16+1)}{2} = \frac{4 \times 17}{2} = 34$.

Therefore, for $n = 8$, substituting into the formula: $S = \frac{8(8^2+1)}{2}$. First, calculate 8 squared: $8 \times 8 = 64$. Then add 1: $64 + 1 = 65$. Multiply by 8: $8 \times 65 = 520$. Then divide by 2: $\frac{520}{2} = 260$.

Wait, just to be thorough, let me recast the problem another way. Instead of using the formula, maybe I can compute the sum of numbers from 1 to 64 and then divide by 8. Let's see. The sum from 1 to 64 is $\frac{64 \times 65}{2} = \frac{4160}{2} = 2080$. Then divide by 8 (since there are 8 rows) to get the magic sum: $\frac{2080}{8} = 260$. Same result.

A sanity check: in a magic square, the magic sum is also equal to the number of the square (n) multiplied by the average of the first and last numbers. Since the numbers run from 1 to n^2 , the first number is 1 and the last is n^2 . Their average is $\frac{1+n^2}{2}$. So magic sum should be $S = n \times \frac{1+n^2}{2}$. Wait, that's the same as the formula we had earlier: $S = \frac{n(n^2+1)}{2}$. Yep, that's exactly the same. So plugging in $n = 8$: $S = 8 \times \frac{1+64}{2} = 8 \times \frac{65}{2} = \frac{8}{2} \times 65 = 4 \times 65 = 260$. Perfect, same answer.

So all these different approaches lead to the same result, 260. Therefore, I can be confident that the magic sum for an 8x8 normal magic square is 260.

****Final Answer**** The magic sum of a normal magic square of order 8 is 260.

Figure 6: It can be observed that with proper CoT prefix, the SLM can infer 260 without explicit calculation

Question

Today is January 31, 2015, welcome to the 2015 "Spring Cup" Final. The calculation result of the expression $\frac{2015/1+2015/0.31}{1+0.31}$ is:

Response of DeepSeek-R1

Okay, so I need to calculate this expression: $\frac{2015/1+2015/0.31}{1+0.31}$. Hmm, let me break this down step by step. First, let's look at the numerator: $\frac{2015}{1} + \frac{2015}{0.31}$. The denominator is $1 + 0.31$. Maybe I can simplify each part separately before combining them.

Starting with the numerator. The first term is $2015 \div 1$, which is straightforward. Dividing any number by 1 gives the number itself, so that's just 2015. The second term is $2015 \div 0.31$. Hmm, dividing by a decimal. Let me think how to compute that.

Dividing by 0.31 is the same as multiplying by $\frac{1}{0.31}$. Let me calculate $1 \div 0.31$. Well, $0.31 \times 3 = 0.93$, and 0.31×3.2 would be 0.992. So $1 \div 0.31$ is approximately 3.2258. But I should probably do this more accurately. Let's do the division: $1 \div 0.31$.

Alternatively, I can write 0.31 as a fraction, which is $\frac{31}{100}$. So $1 \div \frac{31}{100} = \frac{100}{31}$. So $\frac{2015}{0.31} = 2015 \times \frac{100}{31}$. Let me compute that.

$2015 \times 100 = 201500$. Then, $\frac{201500}{31}$. Let me do that division. $31 \times 6000 = 186000$. Then $201500 - 186000 = 15500$. $31 \times 500 = 15500$. So total is $6000 + 500 = 6500$. So $\frac{2015}{0.31} = 6500$? Wait, really? Let me check that again.

$31 \times 6500 : 31 \times 6000 = 186000$, $31 \times 500 = 15500$. Adding them together gives $186000 + 15500 = 201500$. Wait, but $\frac{2015}{0.31} = 2015 \times \frac{100}{31} = \frac{201500}{31}$. Wait, $31 \times 6500 = 201500$? Wait, 31×6500 . Let me compute $31 \times 6000 = 186000$ and $31 \times 500 = 15500$. Then $186000 + 15500 = 201500$. So yes, $31 \times 6500 = 201500$. But $2015 \times 100 = 201500$. So $\frac{201500}{31}$ is indeed 6500. So that part is correct. Therefore, $\frac{2015}{0.31} = 6500$.

So the numerator is $2015 + 6500$. Let's add those together. $2000 + 6500 = 8500$, and then the remaining 15 makes it 8515. So the numerator is 8515.

Now the denominator is $1 + 0.31 = 1.31$. So the entire expression simplifies to $\frac{8515}{1.31}$.

Hmm, so $8515 \div 1.31$. Let's see if we can compute that. Maybe there's a simplification here. Let me check if $1.31 \times 6500 = 8515$. Let me compute 1.31×6500 .

First, $1 \times 6500 = 6500$. Then 0.31×6500 . Let me compute $0.3 \times 6500 = 1950$. Then $0.01 \times 6500 = 65$. So $1950 + 65 = 2015$. Then $6500 + 2015 = 8515$. So yes, $1.31 \times 6500 = 8515$.

Therefore, $\frac{8515}{1.31} = 6500$.

Wait a minute, so the entire expression simplifies to 6500? That seems too straightforward. Let me recap to check for any mistakes. Original expression: $\frac{2015/1+2015/0.31}{1+0.31}$ Numerator: $2015 + \frac{2015}{0.31} = 2015 + 6500 = 8515$ Denominator: 1.31 $\frac{8515}{1.31} = 6500$

Yes, that seems correct. The key was recognizing that both the numerator and the denominator could be related through the same factor. Alternatively, maybe factor out 2015 from the numerator? Let me try that approach to verify.

Numerator: $2015 \left(\frac{1}{1} + \frac{1}{0.31} \right) = 2015 \left(1 + \frac{100}{31} \right) = 2015 \left(\frac{31+100}{31} \right) = 2015 \left(\frac{131}{31} \right)$

Denominator: $1.31 = \frac{131}{100}$

So the entire expression is $\frac{2015 \cdot \frac{131}{31}}{\frac{131}{100}} = 2015 \cdot \frac{131}{31} \cdot \frac{100}{131} = 2015 \cdot \frac{100}{31} = \frac{201500}{31} = 6500$.

Same result. So that confirms it. Therefore, the answer is 6500. But just to make sure, let's do the division $8515 \div 1.31$ step by step. First, write $\frac{8515}{1.31}$. To eliminate the decimal, multiply numerator and denominator by 100 to get $\frac{851500}{131}$.

Now compute $\frac{851500}{131}$. Let's see how many times 131 goes into 851500.

But earlier, we saw that $131 \times 6500 = 851500$. Because $131 \times 100 = 13100$, so $131 \times 6500 = 131 \times (65 \times 100) = (131 \times 65) \times 100$. Wait, 131×65 . Let's compute that.

$131 \times 60 = 7860$ and $131 \times 5 = 655$. So $7860 + 655 = 8515$. So $131 \times 65 = 8515$. Therefore, $131 \times 6500 = 8515 \times 100 = 851500$. Therefore, $\frac{851500}{131} = 6500$. So everything checks out. Therefore,

the result of the expression is 6500. Final Answer: 6500

Figure 7: It can be observed that the model has completed the inference before the original step "Let me check that again.", and the subsequent parts become redundant for the actual answer.