R1-Compress: Long Chain-of-Thought Compression via Chunk Compression and Search

Yibo Wang[1]*, **Haotian Luo**[2]*, **Huanjin Yao**[1], **Tiansheng Huang, Haiying He**[2]
Rui Liu[3], **Naiqiang Tan**[3], **Jiaxing Huang**[4], **Xiaochun Cao**[2] **Dacheng Tao**[4] **Li Shen**[2]†
[1] Tsinghua University [2] Shenzhen Campus of Sun Yat-sen University
[3] Didichuxing Co. Ltd [4] Nanyang Technological University

Abstract

Chain-of-Thought (CoT) reasoning enhances large language models (LLMs) by enabling step-by-step problem-solving, yet its extension to Long-CoT introduces substantial computational overhead due to increased token length. Existing compression approaches—instance-level and token-level—either sacrifice essential local reasoning signals like reflection or yield incoherent outputs. To address these limitations, we propose R1-Compress, a two-stage chunk-level compression framework that preserves both local information and coherence. Our method segments Long-CoT into manageable chunks, applies LLM-driven inner-chunk compression, and employs an inter-chunk search mechanism to select the short and coherent sequence. Experiments on Qwen2.5-Instruct models across MATH500, AIME24, and GPQA-Diamond demonstrate that R1-Compress significantly reduces token usage while maintaining comparable reasoning accuracy. On MATH500, R1-Compress achieves an accuracy of 92.4%, with only a 0.6% drop compared to the Long-CoT baseline, while reducing token usage by about 20%.

1 Introduction

Chain-of-Thought (CoT) reasoning [15, 33, 39, 40, 41] has recently emerged as a powerful technique that enables large language models (LLMs) to perform complex reasoning tasks, such as mathematical problem solving [9, 18] and code generation [2, 11], by decomposing the reasoning process into a sequence of intermediate steps. Recent advancements, including OpenAI's o1 [25], DeepSeek-R1 [6], leverage reinforcement learning to scale to Long-CoT, further improving performance and enabling LLMs to tackle real-world tasks.

However, the extended token length in Long-CoT incurs substantial computational overhead, leading to slower inference and a dramatic increase in KV cache memory usage [31, 28, 32]. These factors significantly hinder practical deployment and impose greater demands on hardware infrastructure. Therefore, developing efficient compression methods for Long-CoT that preserve their reasoning capabilities is of critical importance for enabling scalable and deployable reasoning systems.

Existing methods for CoT compression can be broadly categorized into two paradigms: instance-level compression and token-level compression. Instance-level compression includes C3oT [12] and CoT-Valve [21]. C3oT utilize powerful LLMs like GPT-4 to directly compress entire CoT sequences. CoT-Valve compresses the length of CoT by identifying and manipulating a specific direction in the parameter space. These methods aim to retain the essential reasoning path while reducing the global token count. Token-level compression, such as TokenSkip [35], adopt a more fine-grained strategy by

*Equal contribution

†Corresponding Author: Li Shen (shenli6@mail.sysu.edu.cn)

Preprint.

identifying and skipping unimportant tokens. This allows for a compressed representation that retains detailed local information.

However, our evaluation results show that the instance-level compression can degrade the local information by reducing the global token count—the reflection in Long-CoT is reduced, leading to a decline in performance. As reflection is a crucial capability within Long-CoT that enables LLMs to self-reflect and explore the correct answer, it needs to be preserved through a more fine-grained compression approach. TokenSkip as a token-level method could preserve local information such as reflection well by skipping only unimportant tokens. However, through observation and analysis, we find that this direct token-skipping approach often leads to incoherent compressed CoT, creating a gap from the natural language patterns typically used by LLMs.

Based on the above finding, it seems that effectively compressing Long-CoT cannot be achieved solely through instance-level or token-level methods. Therefore, we propose a chunk-level compression approach, which better preserves chunk-level local information and can be implemented via prompting LLMs, thus maintaining linguistic coherence. However, since each chunk is compressed independently, contextual connections between chunks are lost. A subsequent question is that:

*Although coherence within each chunk can be ensured, how to ensure coherence across **inter chunks**?*

Driven by this question, we propose a chunk search mechanism that generates multiple compressed candidate chunks and employs a search model to select the most coherent one. Conditioned on the previously selected optimal chunk, the search model identifies the candidate with the highest likelihood of maintaining continuity, thereby enhancing coherence across the compressed reasoning process. To improve efficiency, we first filter each chunk's candidates to retain a smaller subset.

To this end, we propose R1-Compress in Figure 1, a two-stage method designed to compress Long-CoT on chunk-level: i) The original CoT is segmented into multiple chunks based on predefined length and formatting constraints. Within each chunk, an LLM is prompted to perform local compression. ii) We first generate multiple compressed candidates for each chunk and a chunk-level search is performed to obtain the short and coherent one. By combining inner-chunk compression with inter-chunk search, our method yields a compressed yet consistent CoT, enabling efficient and coherent reasoning. We evaluate our method on Qwen2.5-14B-Instruct and Qwen2.5-32B-Instruct [29] on the subset of Open-Math-R1 dataset with responses generated by DeepSeek-R1. Experiments are conducted on the MATH500 [9] and AIME24 [22] benchmarks for mathematical reasoning, and GPQA [30] for out-of-distribution reasoning. Results show that our method consistently reduces inference token usage across model scales and datasets while maintaining comparable accuracy. Our method achieves 92.4% accuracy on MATH500—only 0.6% below the Long-CoT baseline (93%), with about 20% reduction in token usage (from 2406 to 1949). The contributions of this paper:

- We find that instance-level compression methods tend to overlook local information—such as reducing the number of reflections in Long-CoT—which negatively impacts performance. In addition, our analysis reveals that token-level methods often lead to CoT lacking coherence.
- To preserve the local information of Long-CoT and generate coherent reasoning chains, we propose R1-Compress, a two-stage chunk-level approach. This method combines inner-chunk compression with inter-chunk search to produce CoT that are both efficient and coherent.
- Extensive results demonstrate that our method can effectively reduce the length of CoT while maintaining the model's reasoning performance across reasoning benchmark.

2 Related Work

Chain-of-Thought. [34, 14] prompting has emerged as a powerful technique for improving the reasoning capabilities of large language models (LLMs). By encouraging the model to solve complex problems step by step, CoT significantly enhances the accuracy and interpretability of its outputs. CoT is particularly effective for tasks that requiring multiple solving steps, such as mathematical problem-solving and logical reasoning. Beyond the basic CoT paradigm, many innovative frameworks like Tree of Thought (ToT) [42] and Graph of Thought (GoT) [1] expand upon the CoT architecture by investigating various reasoning trajectories or integrating structures based on networks. Besides, Chain-of-thought reasoning also enable human to comprehend the model's decision-making pathway, thereby rendering the reasoning process both transparent and credible.

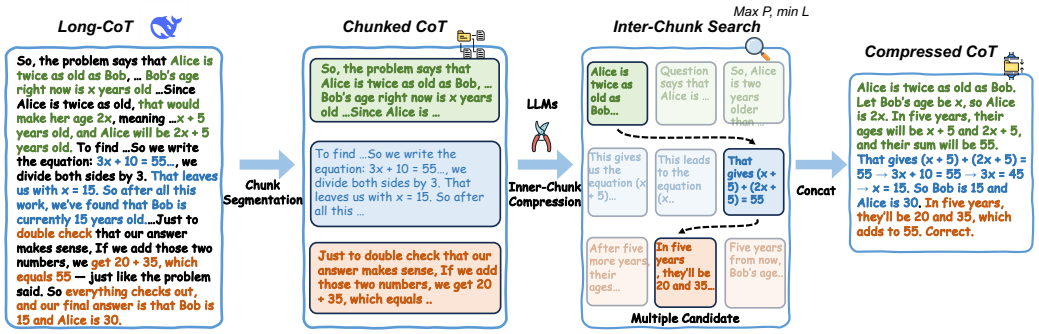


Figure 1: Pipeline of our method. The Long-CoT is segmented into chunks, multiple compressed candidates for each chunk are generated using a LLM, and then a compressed CoT is constructed chunk by chunk through inter-chink search with length filtering and probability selection.

Efficient Reasoning. Some approaches[17] adopt sampling-based and post-training techniques to fine-tune existing Long-CoT models for shorter outputs. For example, Overthinking[4] utilizes DPO and SimPO to construct preference datasets for concise reasoning and trains models accordingly. O1-Pruner[19] establishes baselines for CoT length and accuracy via sampling, then employs offline optimization to shorten CoT without degrading performance. Concise Reasoning [23] leverages simple fine-tuning strategies based on self-generated concise CoT obtained through best-of-N sampling and few-shot prompting. While other methods use different reasoning paradigm to enhance efficiency. For example, Speculative Thinking[38] enables large reasoning models to guide smaller ones during inference at the reasoning level; LightThinker[43] dynamically compress intermediate thoughts during reasoning and Sleep-time Compute[16] allows models to "think" offline about contexts before queries are presented. Additionally, methods like COCONUT[8] and CCOT[5] enable reasoning in the latent space. Besides, some other work [36, 26, 20, 27, 44, 37, 10, 24, 13, 7] also design novel reasoning paradigms for efficiency.

Chain-of-Thought Compression. Several methods aim to directly compress Chain-of-Thought (CoT) of Large Reasoning Models. C3oT employs LLMs to compress CoT end-to-end. CoT-Valve[21] controls the parameter space to generate CoT samples with varying levels of compression for training models that output shorter reasoning paths. TokenSkip[35] selectively removes tokens based on their estimated importance within the CoT.

3 Revisiting Long-CoT Compression

3.1 Problem Setup

Long-CoT. Long-CoT approaches, such as OpenAI's o1 and DeepSeek-R1, exhibit the ability to identify and correct their own mistakes by decomposing complex reasoning steps into simpler subproblems. This iterative process significantly enhances the model's reasoning capability. Long-CoT typically consists of multiple steps. In this work, we adopt the responses generated by DeepSeek-R1 as representative Long-CoT.

SFT with Compressed Long-CoT. We focuses on compressing the token length of Long-CoT by directly reducing the original Long-CoT into shorter reasoning chains. In our setup, the Long-CoT baseline refers to the model fine-tuned using the original Long-CoT responses, while Long-CoT Compression method denotes the model fine-tuned on the compressed versions of Long-CoT. The latter retains the reasoning capabilities of Long-CoT while reducing the number of output tokens.

3.2 Revisiting Instance-level Compression

In this section, we investigate existing instance-level compression methods and discuss their limitations in preserving local information, particularly the reflection steps within Long-CoT. Further experimental analysis reveals that this reduction in reflection leads to a decline in performance.

Reflection. Reflection is the model's ability to evaluate and revise its reasoning process during problem-solving. It enables the model to recognize and correct its own mistakes, decompose complex

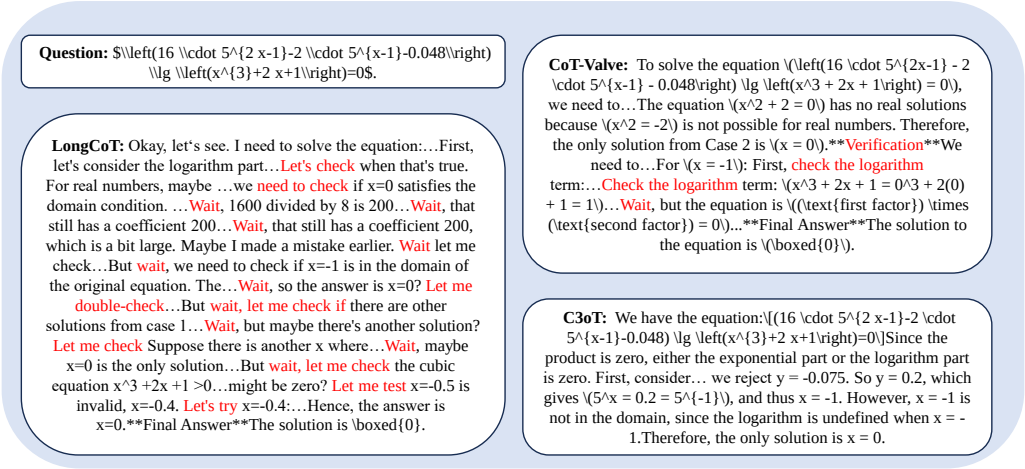


Figure 2: Comparison of LongCoT, CoT-Valve, and C3oT. Red text indicates reflection-related phrases such as "Wait".

steps into simpler components, and adapt its strategy when the current approach proves ineffective. This iterative process of self-assessment and adjustment plays a crucial role in enhancing the model's overall reasoning capability.

We select the C3oT that simply prompting the LLMs to obtain the compressed CoT, CoT-Valve that manipulating a specific direction in the parameter space to reduce the length of CoT to Compress the Long-CoT from DeepSeek-R1. As shown in Figure 2, we find that the compressed CoT obtained through these two methods are able to preserve certain key steps and reach the final answer. However, compared to the original Long-CoT, they omit a considerable number of intermediate steps and exploratory attempts—particularly the processes of reflection and strategy switching that are often essential for arriving at the correct solution. Since reflection is a critical reasoning skill that models are expected to learn from Long-CoT supervision, the absence of this capability prompts an important question:

Does the reduced frequency of reflection in compressed Long-CoT adversely affect the performance of models fine-tuned on it?

To evaluate this, we count the occurrences of indicative reflection-related keywords (See Sec. 5.3 for more details,) that is also adopted by other methods [3]. We calculate the average number of reflection steps in 500 examples for both the original Long-CoT and the CoT compressed by the two mentioned methods. We then evaluate the performance of models fine-tuned using these different CoT. As shown in Table 1, the quantitative results reveal that as the number of reflection steps decreases, the performance of the fine-tuned model deteriorates—indicating that the absence of reflection impairs the model's reasoning ability.

Conclusion. Instance-level compression methods operate from a global perspective and fail to preserve local information such as reflection, which is crucial for reasoning. Since the presence of reflection significantly affects the reasoning ability of fine-tuned models, a more fine-grained compression strategy is needed to effectively retain such local information.

Table 1: Comparison of methods on average reflection and accuracy on MATH500.

Method	Avg. Reflection	Accuracy (%)
Long-CoT	18.68	88.0
C3oT	0.15	65.8
CoT-Valve	8.36	77.4

3.3 Revisiting Token-level Compression

Token-level methods are inherently capable of preserving local information in Long-CoT, such as reflection. In this section, we explore existing token-level compression approaches and analyze the coherence issues observed in the resulting CoT. Furthermore, we perform a quantitative analysis using loss values derived from probabilistic predictions.

Question: Let $a_n\ (n\geq 1)$ be the value for which $\int_x^{2x} e^{-t^n}dt\ (x\geq 0)$ is maximal. Find $\lim_{n\to\infty} \ln a_n.$

Origin : Okay, so I need to find the limit as n approaches infinity of the natural logarithm of a_n, where a_n is the value that maximizes the integral from x to 2x of e^{-t^n} dt for x >= 0. Hmm, let's break this down step by step. First, I remember that to find the value of x that maximizes a function, we take the derivative of that function with respect to x and set it equal to zero. So, the integral from x to 2x of e^{-t^n} dt is our function,let's call it F(x)...<OMITTED>

Tokenskip: Okay, ~~need to~~ find limit as n approaches infinity of natural logarithm of a_n, where a_n is value maximizes integral from x to 2x of e^{-t^n} dt for x >= 0. Let's break this down step by step. ~~First remember~~ to find value of x that maximizes a function, ~~take derivative~~ of function with respect to x set it equal to zero. integral from x to 2x of e^{-t^n} dt ~~is our function,'s call it~~ F(x). ~~F'(x) = 0 will give us critical points~~, can check if it's maximum, ~~compute F''(x)~~. Using Fundamental Theorem of Calculus, ~~derivative of integral with variable limits.integral from~~ a(x) to b(x) of f(t derivative f(b(x)) * b'(x) - f(a(x)) * a'(x).....<OMITTED>

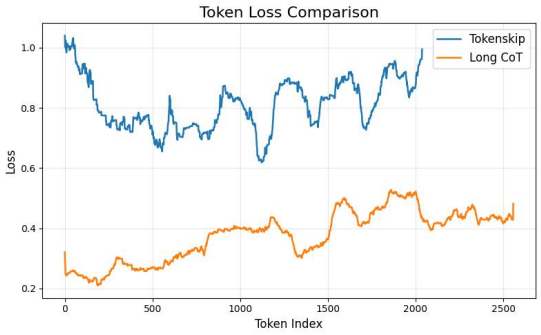


Figure 3: Left: Example of TokenSkip CoT Compression, Right: Token-level loss curves of Long-CoT and TokenSkip.

We select TokenSkip as a representative token-level method to compress Long-CoT generated by DeepSeek-R1. As shown in Figure 3 Left, although TokenSkip can identify and remove unimportant tokens—thus partially preserving the original semantic content—we observe that the compressed CoT often exhibit clear incoherence, for example, "` of function`" and "`, can check if`". We attribute this to a mismatch between the compressed outputs and the natural language patterns expected by LLMs. This gap not only results in incoherent outputs after supervised fine-tuning, but also affects the training dynamics by increasing the prediction loss due to the unnatural text of the input.

We quantify coherence using token-level loss, computed as the negative log-likelihood of each token in the compressed CoT predicted by the LLM. (See more details in Sec. 5.4). As shown in Figure 3 Right, the token-level loss of the Tokenskip is generally higher than that of the origin, which further indicates a significant inconsistency between its output and the original content, making it less aligned with the typical output patterns of LLMs.

Conclusion. Token-level compression methods often produce incoherent CoT, which can negatively impact the training process and lead to models that generate incoherent outputs. In contrast, instance-level methods, such as C3oT, compress global information through prompting LLMs, resulting in more coherent outputs.

3.4 Derived Insight

Based on the above analysis, we argue that effectively compressing Long-CoT cannot be achieved solely through instance-level or token-level methods. To address this, we propose compressing Long-CoT at the **chunk level**. Chunk-level compression allows for better preservation of local information within each chunk while maintaining stronger inner-chunk coherence. To ensure coherence across chunks, we introduce a inter-chunk search mechanism, which selects the most coherent sequence of chunks. Additionally, we incorporate a search over compression lengths to further enhance compression efficiency.

4 Method

4.1 Chunk Segmentation

We are given a dataset of problem-solution pairs, denoted as $\mathcal{D} = \{(x_k, y_k)\}_{k=1}^{N}$ where x_k represents a problem and $y_k = [y_k^1, \ldots, y_k^{m_k}]$ denotes its corresponding solution generated by a large language model (LLM) parameterized by $\boldsymbol{\theta}$, denoted as $\pi_{\boldsymbol{\theta}}(\cdot \mid x_k)$. Each solution y_k (CoT) is segmented into a sequence of m_k constituent chunks: $y_k = [c_{k,1}, c_{k,2}, \ldots, c_{k,m_k}]$

To obtain the chunks $c_{k,j}$ from the raw text y_k, we use the following segmentation strategy:**Minimum length requirement**: A chunk must contain at least a predefined minimum number of characters or tokens (e.g., 50 words). **Double newline boundary**: A chunk ends when two consecutive newline characters (‘/n/n‘) are encountered, provided that the current chunk has met the minimum length requirement. This results in variable-length chunks that are semantically meaningful and structurally

coherent, often corresponding to paragraphs or logical substeps in a solution. This chunking strategy ensures that each $c_{k,j}$ captures a complete unit of reasoning or explanation, which is essential for later compression and search.

4.2 Inner-Chunk Compression

The simplification process for a given pair (x, y) (dropping the index k for clarity) involves the following steps: For each chunk c_i in the solution $y = [c_1, c_2, \ldots, c_m]$, we utilize a separate LLM, parameterized by ϕ and denoted as π_ϕ, to generate multiple simplified candidate versions. Given the original chunk c_i and a suitable prompt p, we sample M candidate simplified chunks from the conditional distribution $\pi_\phi(\cdot|p, c_i)$. These candidates for chunk c_i are denoted as $\{\hat{c}_i^j\}_{j=1}^M$.

$$\hat{c}_i^j \sim \pi_\phi(\cdot \mid p, c_i), \quad \text{for } j = 1, \ldots, M \tag{1}$$

This process is applied independently to each chunk c_i of the original solution y. The prompt p is carefully designed to guide the LLM toward generating simplified and concise versions of the input chunk while preserving its original meaning. The full prompt used in our experiments is provided in the Appendix.

4.3 Inter-Chunk Search

After obtaining M candidate simplified chunks $\{\hat{c}_i^j\}_{j=1}^M$ for each original chunk c_i in y, we aim to construct a complete simplified solution sequence $y^* = [\hat{c}_1^*, \hat{c}_2^*, \ldots, \hat{c}_m^*]$ by selecting one optimal candidate $\hat{c}_i^*$ for each position i. The selection criteria prioritize brevity and a low "loss", where loss is inversely related to the probability assigned by the original LLM π_θ to the simplified sequence. We employ a greedy search approach:

Length-based Filtering: For each position i, we first filter the set of M candidates $\{\hat{c}_i^j\}_{j=1}^M$. We discard the $\alpha \cdot M$ longest candidates, keeping the $(1-\alpha)M$ shortest ones, where $\alpha \in [0, 1)$ is a predetermined filtering ratio. Let the filtered set of candidates for position i be $\tilde{\mathcal{C}}_i \subseteq \{\hat{c}_i^j\}_{j=1}^M$.

Probability-based Selection: We iteratively select the best simplified chunk for each position $i = 1, \ldots, m$. At position i, having selected the optimal simplified chunks $\hat{c}_1^*, \ldots, \hat{c}_{i-1}^*$ for the preceding positions, we choose the candidate $\hat{c}_i^* \in \tilde{\mathcal{C}}_i$ that maximizes the conditional probability under the original LLM π_θ, given the original problem x and the previously selected simplified chunks:

$$\hat{c}_i^* = \arg\max_{\hat{c} \in \tilde{\mathcal{C}}_i} \pi_\theta(\hat{c} \mid x, \hat{c}_1^*, \ldots, \hat{c}_{i-1}^*) \tag{2}$$

For the first chunk ($i = 1$), the selection is based solely on the probability conditioned on the problem x:

$$\hat{c}_1^* = \arg\max_{\hat{c} \in \tilde{\mathcal{C}}_1} \pi_\theta(\hat{c} \mid x) \tag{3}$$

4.4 Compressed CoT

The final simplified solution y^* for the problem x is constructed by concatenating the sequence of optimally selected simplified chunks:

$$y^* = [\hat{c}_1^*, \hat{c}_2^*, \ldots, \hat{c}_m^*] \tag{4}$$

This entire process is applied to each (x_k, y_k) pair in the dataset $\mathcal{D}$ to obtain a dataset $\mathcal{D}_{\text{compressed}}$ of simplified solutions.

4.5 Fine-tuning with Compressed CoT

After obtaining the compressed dataset $\mathcal{D}_{\text{compressed}} = \{(x_k, y_k^*)\}_{k=1}^N$, we perform standard supervised fine-tuning (SFT) on the base model π_θ to better align it with the simplified reasoning trajectories.

The training objective is to maximize the log-likelihood of the compressed outputs given the input problems:

$$\mathcal{L}_{\text{SFT}}(\theta) = \sum_{k=1}^{N} \log \pi_\theta(y_k^* | x_k) \tag{5}$$

This fine-tuning step encourages the model to generate concise yet faithful reasoning chains.

5 Experiments

5.1 Experiment Settings

Dataset. For training, we use the OpenR1-Math-220k dataset, a large-scale benchmark for mathematical reasoning. It consists of 220k math problems, each responses generated by DeepSeek-R1. For evaluation, we leverage two widely used mathematical reasoning benchmarks. MATH500 and AIME24. GPQA-Diamond as an out-of-distribution benchmark. More details are in Appendix A.2.

Baseline. We consider two primary baselines: **CoT-Valve** and **TokenSkip**. **R1-Compress** $_{random}$ is a variant of R1-Compress that randomly selects a candidate chunk during compression. **Long-CoT** refers to supervised fine-tuning (SFT) on the original DeepSeek-R1 responses without any compression. **Base** denotes the model without SFT. In our experiments, the base model is fine-tuned on the compressed CoT generated by each method. More details can be found in Appendix A.3.

Metric. We employ the following three metrics to evaluate the model's performance. **Accuracy**: For MATH500 and GPQA-Diamond, we report pass@1 accuracy. For AIME24, due to its small size, we report avg@10 accuracy. **Token** (Token Length): The average token length of generated responses. **Valid** (Valid Token Length): The average token length of responses that are answered correctly.

Implementation Details. We primarily evaluate our method using the Qwen2.5-Instruct series (14B/32B). All evaluations are conducted using the `lighteval` framework, following the widely adopted Long-CoT evaluation setting, with a temperature of 0.6 and a top-p of 0.95.

For supervised fine-tuning (SFT), we use a learning rate of 1e-5 and train for 4 epochs using the `LlamaFactory` library. For chunk compression, we utilize LLaMA3.1-70B-Instruct with a sampling temperature of 0.75 and generate 8 candidate chunks. The chunk search method is performed by the DeepSeek-R1-Distill-Qwen-14B model, More details are in Appendix A.1.

5.2 Main Results

For the Qwen2.5-14B-Instruct model in Table 2, we observe that R1-Compress achieves consistent improvements over the Long-CoT baseline by significantly reducing the average token length—ranging from a reduction of 412 tokens on MATH500 to 1056 tokens on GPQA-Diamond. Importantly, R1-Compress attains the highest accuracy and lowest token length across all three benchmarks comparing with other baselines, demonstrating its ability to effectively compress Long-CoT without compromising its reasoning effectiveness. Our method also performs well on the out-of-distribution benchmark GPQA-Diamond, highlighting its generalizability. Compared to R1-Compress $_{random}$, our full method further improves both accuracy and token efficiency, validating the effectiveness of the proposed inter-chunk search in selecting shorter and more coherent CoT.

As we scale up to the Qwen2.5-32B-Instruct model, R1-Compress continues to outperform all baselines in terms of token length while achieving the best or comparable accuracy. On MATH500, our method achieves a strong accuracy of 92.4%, with only 0.6% performance drop compared to the Long-CoT baseline (93.0%), while reducing the average token length by around 500 tokens. Furthermore, the valid token length is reduced by approximately 20% (from 2406 to 1949) under nearly equal numbers of correct responses. The consistent improvement over R1-Compress $_{random}$ on the larger model further supports the robustness and scalability of our proposed search strategy.

5.3 Reflection Evaluation

We conduct this analysis by counting the occurrences of reflection-related keywords: “wait”, “alternatively”, "emm", "hmm". These tokens indicate shifts in reasoning or self-reflection.

Table 2: Main experiment results. We present the performance of two models and report accuracy (↑), average token length (Token) (↓), valid token length (Valid) (↓) on three reasoning benchmark.

Methods	**MATH500**		**AIME24**		**GPQA-Diamond**	
	Accuracy	Token (Valid)	Accuracy	Token (Valid)	Accuracy	Token (Valid)
Qwen2.5-14B-Instruct						
Base	79.8	-	11.00	-	47.97	-
Long-CoT	88.0	3781 (2601)	30.00	12101(6402)	51.51	9600 (7830)
CoT-Valve	77.4	3733 (**1343**)	15.00	12972 (**4186**)	39.89	10257 (**6704**)
TokenSkip	82.8	4236 (2313)	17.66	13504 (4644)	33.83	11974 (8130)
R1-Compress $_{random}$	81.2	3880 (2033)	24.00	12444 (6381)	48.48	9524 (7153)
R1-Compress	**84.8**	**3369** (2074)	**25.66**	**11369** (5575)	**49.49**	**8544** (6962)
Qwen2.5-32B-Instruct						
Base	83.2	-	16.66	-	50.0	-
Long-CoT	93.0	3147 (2406)	50.66	**10541** (5997)	61.11	8054 (6199)
CoT-Valve	91.0	2718 (1891)	39.33	11357 (5898)	54.04	9578 (6891)
TokenSkip	89.8	3004 (**1871**)	**44.33**	10881 (6000)	**59.59**	8505 (5877)
R1-Compress $_{random}$	89.8	2899 (1965)	42.00	11135 (5705)	54.04	8335 (6510)
R1-Compress	**92.4**	**2661** (1949)	43.33	10747 (**5495**)	59.09	**6963** (**5005**)

As shown in Table 3, our method preserves significantly more reflection steps compared to other baselines—on average, six more than CoT-Valve. We preserve approximately 78% of the reflections found in Long-CoT, while achieving accuracy competitive with the original Long-CoT responses.

Table 3: Comparison of methods on average reflection and accuracy on MATH500.

Method	**Avg. Reflection**	**Accuracy (%)**
Long-CoT	18.68	88.0
C3oT	0.15	65.8
CoT-Valve	8.36	77.4
R1-Compress	**14.59**	**84.8**

Table 4: Coherence evaluation. Comparison of methods on token-level loss.

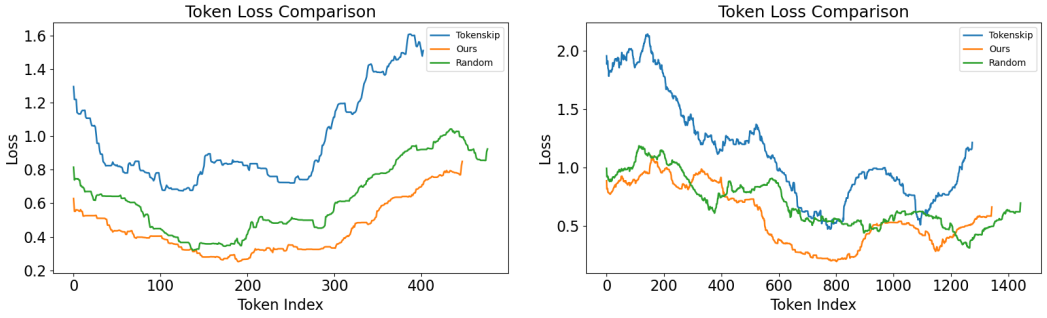
Method	**Token-Level Loss**
Long-CoT	0.41
TokenSkip	0.87
R1-Compress $_{random}$	0.63
R1-Compress	**0.59**

5.4 Coherence Evaluation

To quantitatively assess the coherence of compressed CoT, we compute the token-level log-likelihood loss using the DeepSeek-R1-Distill-Qwen-14B model. Specifically, given a compressed CoT as input and the original uncompressed CoT as reference, we evaluate the average token-level loss for TokenSkip, R1-Compress $_{random}$, R1-Compress.

Table 4 reports the average token-level loss for each method. The results indicate that both of our methods achieve lower token-level loss compared to TokenSkip, indicating better coherence between tokens in the compressed CoT. Moreover, R1-Compress achieves lower loss than R1-Compress $_{random}$, demonstrating that the introduction of the search mechanism improves inter-chunk coherence. Enhanced coherence contributes to greater stability during training and enables the fine-tuned model to produce more semantically precise outputs.

To complement the quantitative results, we also perform a qualitative analysis by visualizing token-level loss across several representative examples in Figure 4. We observe that TokenSkip frequently exhibits high-loss regions, particularly around intermediate conclusions and reflective reasoning steps. R1-Compress $_{random}$ and R1-Compress achieves lower loss compared to R1-Compress $_{random}$ by leveraging the search mechanism to identify chunks that are more coherent within the given context.



(a) Token loss comparison on case 1

(b) Token loss comparison on case 2

Figure 4: Token-level loss visualization.

5.5 Ablation Study

Chunk Size. Table 5 reports the results of using different chunk size constraints during chunk segmentation. For a clearer ablation, we compare variants without the search mechanism, i.e., R1-Compress $_{random}$. The results show that smaller chunk sizes yield higher-quality compressed CoT, as finer-grained chunks better preserve local information and reduce the compression difficulty for LLMs. In the limit where the chunk size becomes unbounded, the method effectively reduces to C3oT.

Table 5: Ablation study on chunk size.

Chunk Size	MATH500	AIME24
1000	79.0 (4188)	21.66 (13074)
500	**81.2** (**3880**)	**24.00** (**11369**)

Search Model. We investigate the impact of different models used in the search phase of reasoning compression. Specifically, we compare Qwen2.5-14B-Instruct (Qwen) and DeepSeek-R1-Distill-Qwen-14B (DeepSeek-Distill) as the search model. As shown in Table 6, we observe that both models, when used as the search model, improve accuracy compared to the variant without search. Specifically, DeepSeek-Distill tends to favor longer responses, resulting in a larger gain in accuracy, while Qwen prefers shorter responses, leading to a lower valid token length. Overall, both search models contribute to improved performance, and by selecting shorter yet coherent chunks, the search process ultimately leads to reduced total token usage.

Table 6: Ablation study on search model. "w/o" denotes the absence of a search model, "w/ Qwen" uses Qwen2.5-14B-Instruct as the search model, and "w/ DeepSeek-Distill" uses DeepSeek-R1-Distill-Qwen-14B as the search model.

Methods	MATH500	
	Accuracy	Token(Valid)
Qwen2.5-14B-Ins		
w/o	81.2	3880 (2033)
w/ Qwen	83.0	3373 (**1874**)
w/ DeepSeek-Distill	**84.8**	**3369** (2074)

6 Conclusion

In this paper, we propose R1-Compress, an effective framework for compressing long Chain-of-Thought (CoT) reasoning by combining inner-chunk compression with an inter-chunk search mechanism. Unlike existing approaches that either compromise critical reasoning behaviors—such as reflection—or lead to incoherent outputs, R1-Compress effectively reduces token length while maintaining high reasoning quality. Experimental results across multiple benchmarks demonstrate the method's ability to preserve performance under significant compression. These findings underscore the potential of chunk-level CoT compression as a practical and scalable solution for enhancing the efficiency and deployability of large-scale reasoning models.

References

[1] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, and T. Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, Mar. 2024.

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

[3] R. Chen, Z. Zhang, J. Hong, S. Kundu, and Z. Wang. Seal: Steerable reasoning calibration of large language models for free. *arXiv preprint arXiv:2504.07986*, 2025.

[4] X. Chen, J. Xu, T. Liang, Z. He, J. Pang, D. Yu, L. Song, Q. Liu, M. Zhou, Z. Zhang, R. Wang, Z. Tu, H. Mi, and D. Yu. Do not think that much for 2+3=? on the overthinking of o1-like llms, 2025.

[5] J. Cheng and B. V. Durme. Compressed chain of thought: Efficient reasoning through dense representations, 2024.

[6] DeepSeek. Deepseek-r1-lite-preview: Unleashing supercharged reasoning power. `https://api-docs.deepseek.com/news/news1120`, 2024. Accessed: 2024-12-29.

[7] R. Gong, Y. Liu, W. Qu, M. Du, Y. He, Y. Ma, Y. Chen, X. Liu, Y. Wen, X. Li, R. Wang, X. Zhu, B. Hooi, and J. Zhang. Efficient reasoning via chain of unconscious thought, 2025.

[8] S. Hao, S. Sukhbaatar, D. Su, X. Li, Z. Hu, J. Weston, and Y. Tian. Training large language models to reason in a continuous latent space, 2024.

[9] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

[10] B. Hou, Y. Zhang, J. Ji, Y. Liu, K. Qian, J. Andreas, and S. Chang. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning, 2025.

[11] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim. A survey on large language models for code generation, 2024.

[12] Y. Kang, X. Sun, L. Chen, and W. Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness, 2024.

[13] Z.-Z. Li, X. Liang, Z. Tang, L. Ji, P. Wang, H. Xu, X. W, H. Huang, W. Deng, Y. Gong, Z. Guo, X. Liu, F. Yin, and C.-L. Liu. Tl;dr: Too long, do re-weighting for efficient llm reasoning compression, 2025.

[14] Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P.-J. Wang, X. Chen, Y. Zhang, F. Yin, J. Dong, Z. Guo, L. Song, and C.-L. Liu. From system 1 to system 2: A survey of reasoning large language models, 2025.

[15] Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P.-J. Wang, X. Chen, Y. Zhang, F. Yin, J. Dong, Z. Li, B.-L. Bi, L.-R. Mei, J. Fang, Z. Guo, L. Song, and C.-L. Liu. From system 1 to system 2: A survey of reasoning large language models, 2025.

[16] K. Lin, C. Snell, Y. Wang, C. Packer, S. Wooders, I. Stoica, and J. E. Gonzalez. Sleep-time compute: Beyond inference scaling at test-time, 2025.

[17] Y. Liu, J. Wu, Y. He, H. Gao, H. Chen, B. Bi, J. Zhang, Z. Huang, and B. Hooi. Efficient inference for large reasoning models: A survey. *arXiv preprint arXiv:2503.23077*, 2025.

[18] H. Luo, H. He, Y. Wang, J. Yang, R. Liu, N. Tan, X. Cao, D. Tao, and L. Shen. Adar1: From long-cot to hybrid-cot via bi-level adaptive reasoning optimization, 2025.

[19] H. Luo, L. Shen, H. He, Y. Wang, S. Liu, W. Li, N. Tan, X. Cao, and D. Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning, 2025.

[20] W. Ma, J. He, C. Snell, T. Griggs, S. Min, and M. Zaharia. Reasoning models can be effective without thinking, 2025.

[21] X. Ma, G. Wan, R. Yu, G. Fang, and X. Wang. Cot-valve: Length-compressible chain-of-thought tuning, 2025.

[22] MAA. American invitational mathematics examination - aime. In *American Invitational Mathematics Examination - AIME 2024*, February 2024.

[23] T. Munkhbat, N. Ho, S. H. Kim, Y. Yang, Y. Kim, and S.-Y. Yun. Self-training elicits concise reasoning in large language models, 2025.

[24] Y. Ning, W. Li, J. Fang, N. Tan, and H. Liu. Not all thoughts are generated equal: Efficient llm reasoning via multi-turn reinforcement learning, 2025.

[25] OpenAI. Learning to reason with llms. `https://openai.com/index/learning-to-reason-with-llms/`, 2024. [Accessed 19-09-2024].

[26] J. Pan, X. Li, L. Lian, C. Snell, Y. Zhou, A. Yala, T. Darrell, K. Keutzer, and A. Suhr. Learning adaptive parallel reasoning with language models, 2025.

[27] Z. Qiao, Y. Deng, J. Zeng, D. Wang, L. Wei, F. Meng, J. Zhou, J. Ren, and Y. Zhang. Concise: Confidence-guided compression in step-by-step efficient reasoning, 2025.

[28] X. Qu, Y. Li, Z. Su, W. Sun, J. Yan, D. Liu, G. Cui, D. Liu, S. Liang, J. He, P. Li, W. Wei, J. Shao, C. Lu, Y. Zhang, X.-S. Hua, B. Zhou, and Y. Cheng. A survey of efficient reasoning for large reasoning models: Language, multimodality, and beyond, 2025.

[29] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report, 2025.

[30] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023.

[31] Y. Sui, Y.-N. Chuang, G. Wang, J. Zhang, T. Zhang, J. Yuan, H. Liu, A. Wen, S. Zhong, H. Chen, and X. Hu. Stop overthinking: A survey on efficient reasoning for large language models, 2025.

[32] R. Wang, H. Wang, B. Xue, J. Pang, S. Liu, Y. Chen, J. Qiu, D. F. Wong, H. Ji, and K.-F. Wong. Harnessing the reasoning economy: A survey of efficient reasoning for large language models, 2025.

[33] Y. Wang, S. Wu, Y. Zhang, S. Yan, Z. Liu, J. Luo, and H. Fei. Multimodal chain-of-thought reasoning: A comprehensive survey, 2025.

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[35] H. Xia, Y. Li, C. T. Leong, W. Wang, and W. Li. Tokenskip: Controllable chain-of-thought compression in llms, 2025.

[36] C. Yang, Q. Si, Y. Duan, Z. Zhu, C. Zhu, Z. Lin, L. Cao, and W. Wang. Dynamic early exit in reasoning models, 2025.

[37] J. Yang, K. Lin, and X. Yu. Think when you need: Self-adaptive chain-of-thought learning, 2025.

[38] W. Yang, X. Yue, V. Chaudhary, and X. Han. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time, 2025.

[39] H. Yao, J. Huang, W. Wu, J. Zhang, Y. Wang, S. Liu, Y. Wang, Y. Song, H. Feng, L. Shen, and D. Tao. Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search, 2024.

[40] H. Yao, Q. Yin, J. Zhang, M. Yang, Y. Wang, W. Wu, F. Su, L. Shen, M. Qiu, D. Tao, et al. R1-sharevl: Incentivizing reasoning capability of multimodal large language models via share-grpo. *arXiv preprint arXiv:2505.16673*, 2025.

[41] H. Yao, R. Zhang, J. Huang, J. Zhang, Y. Wang, B. Fang, R. Zhu, Y. Jing, S. Liu, G. Li, et al. A survey on agentic multimodal large language models. *arXiv preprint arXiv:2510.10991*, 2025.

[42] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[43] J. Zhang, Y. Zhu, M. Sun, Y. Luo, S. Qiao, L. Du, D. Zheng, H. Chen, and N. Zhang. Lightthinker: Thinking step-by-step compression, 2025.

[44] R. Zhuang, B. Wang, and S. Sun. Accelerating chain-of-thought reasoning: When goal-gradient importance meets dynamic skipping, 2025.

A More Details about Experiments

A.1 Implementation Details.

We perform compression using LLaMA3.1-70B-Instruct on 4 × 80GB GPUs. For model training, we conduct full-parameter fine-tuning of Qwen2.5-14B-Instruct on 8 × 80GB GPUs and Qwen2.5-32B-Instruct on 16 × 80GB GPUs. All fine-tuning procedures run for more than 2–4 hours. The hyperparameters used for full fine-tuning are summarized in Table 7.

Table 7: Hyperparameters for the Qwen2.5-14B-Instruct and Qwen2.5-32B-Instruct.

Hyperparameter	Qwen2.5-14B-Instruct	Qwen2.5-32B-Instruct
cutoff_len	8192	8192
batch_size	8	2
learning_rate	1.0e-5	1.0e-5
num_train_epochs	4.0	4.0
lr_scheduler_type	cosine	cosine
warmup_ratio	0.1	0.1

A.2 Benchmark.

MATH500: A challenging math dataset comprising 500 problems from high school math competitions.

AIME24: A benchmark dataset consisting of 30 challenging mathematical problems from the 2024 American Invitational Mathematics Examination.

GPQA-Diamond: A high-difficulty subset of the GPQA benchmark, with 198 complex graduate-level multiple-choice questions across various scientific domains.

A.3 Baseline.

CoT-Valve: We adopt the Short-Long-Short CoT compression strategy proposed by CoT-Valve, which aligns with our experimental setting. We use an untrained model as the short model and a model fine-tuned on Long-CoT as the long model. By applying model merging, we obtain a Short-Long-Short model, following the setup introduced in CoT-Valve. Specifically, we perform linear interpolation with weights of (0.9, 0.1) and (0.8, 0.2) for the short and long models, respectively, to create different variants of the Short-Long-Short model. These merged models are then used to sample and construct the MixChain of Short-Long-Short CoT dataset.

TokenSkip: This baseline directly applies token-level compression to Long-CoT to generate shortened CoT. We follow its setting to measure the token importance by LLMLingua-2 compressor. The control ratios that we use are 0.9, 0.8, 0.7, 0.6.

C3oT: In Table 1, we adopt the prompt template provided in the original implementation and use the same LLM (LLaMA3.1-70B-Instruct) as the compressor to ensure a fair comparison.

A.4 Metric.

Accuracy. For MATH500 and GPQA-Diamond, we report **pass@1** accuracy, where the model is evaluated with a single response. For AIME24, due to its small size (30), we report the accuracy **avg@10**, calculated as the average accuracy over 10 independent runs.

Token (Token Length): The average token length of all model-generated responses, used to evaluate the overall compression effectiveness.

Valid (Valid Token Length): The average token length of responses that are answered correctly. This metric is introduced to better analyze the relationship between output length and successful reasoning.

A.5 Training Dataset.

We use the OpenR1-Math-220k dataset, a large-scale benchmark for mathematical reasoning. It consists of 220k math problems, each accompanied by two to four reasoning traces generated by DeepSeek R1 for problems sourced from NuminaMath 1.5. All traces have been verified using Math Verify. We randomly sample 5,000 examples from it.

A.6 Filter Strategy.

Each response from DeepSeek-R1 is first segmented into multiple chunks using our chunk segmentation strategy. To ensure efficient downstream compression, we filter out samples with more than 30 chunks, reducing the initial 5k samples to 3.8k. We further refine the dataset by verifying `has_vaild_answer`: Whether the original R1 response contains an extractable answer and `has_same_answer`: Whether the answer extracted from the compressed CoT matches the original one. Additionally, we remove samples with excessively low or high compression ratios. After this filtering process, a total of 2,513 samples are retained for training.

B More Discussion

B.1 Necessity of Chunk.

C3oT compresses Long-CoT directly via LLMs. However, due to the extremely long context of Long-CoT, LLMs often struggle to follow instructions faithfully and preserve critical information. Specifically, we use the advanced model LLaMA3.1-70B-Instruct as the compressor. As shown in Table 8, after compressing 3,620 Long-CoT samples, only 442 resulting CoT retain answers consistent with the original responses. This outcome highlights the limitations of direct instance-level compression and underscores the necessity of our proposed chunk-level approach.

Table 8: Filtering statistics of C3oT-compressed data based on answer consistency. `has_same_answer` is introduce in Appendix A.6.

Stage	Sample Count
Before `has_same_answer` filter	3,620
After `has_same_answer` filter	442

B.2 Main Results.

As shown in Table 2, our method achieves a substantial reduction of over 1,000 tokens on the GPQA-Diamond benchmark, with only a minimal performance drop (approximately 2%) compared to the Long-CoT baseline. The strong performance on this out-of-distribution benchmark suggests that models may inherit overly verbose reasoning patterns from Long-CoT supervision, which are then reflected in other tasks. This observation highlights the practical significance of compressing Long-CoT.

Additionally, we observe that on AIME24, the 32B model exhibits a noticeable reduction in valid token length, while the overall token length remains nearly unchanged. This is because AIME24 is a highly challenging task, and when the model produces incorrect answers, it tends to generate longer responses. Thus, although valid reasoning becomes more concise, the total output length does not decrease accordingly.

C Prompt Template

The compression prompt used for LLMs is provided in Table 9. The templates for dataset construction, mathematical evaluation and GPQA evaluation are shown in Table 10.

Table 9: Compression prompt for LLMs.

Here is an reasoning piece excerpt from some math problem solving process (it is incomplete, but this doesn't matter.): {step}
Instructions:
You need to simplify the wording of given reasoning piece to get a concise reasoning piece.
Notice:
1. Avoid omitting any reasoning steps. You should keep all the reflection, analysing, checking steps and even steps making mistakes. (Especially steps contains word "wait", "hmm")
2. Directly give me the simplified content without any additional words.
3. Do not add additional steps or continue the reasoning process.
4. Follow the format of given reasoning piece.
Output format: `<start>` (simplified content) `<end>`

Table 10: Template for Dataset construction and Evaluation.

Dataset construction template:
{Question} Let's think step by step and output the final answer within boxed{{}}, {Compressed_CoT}.
MATH template:
{Question} Let's think step by step and output the final answer within boxed{{}}.
GPQA template:
{Question} Answer the following multiple choice question. The last line of your response should be of the following format: 'Answer: $LETTER' (without quotes) where LETTER is one of ABCD. Let's think step by step before answering.
A) A
B) B
C) C
D) D

D Case Study

Case Example 1

Problem: In the school, there are 50 teachers, of whom 29 drink coffee, 28 drink tea, and 16 do not drink either coffee or tea. How many teachers drink only coffee, and how many drink only tea?

Long-CoT: (Partial)
\nOkay, let's see. There's a problem about teachers who drink coffee or tea, and some who don't drink either. I need to find out how many drink only coffee and how many drink only tea. Hmm, this sounds like a problem where I can use sets and maybe a Venn diagram. Let me start by recalling some concepts.\n\nFirst, the total number of teachers is 50. Out of these, 29 drink coffee, 28 drink tea, and 16 don't drink either. **Wait**, so the number of teachers who drink at least one of the beverages should be the total minus those who don't drink either. Let me calculate that.\n\nTotal teachers = 50\nTeachers who don't drink coffee or tea = 16\nSo, teachers who drink coffee or tea or both = 50 - 16 = 34.\n\nOkay, so 34 teachers are in the union of coffee drinkers and tea drinkers. Now, I remember that the principle of inclusion-exclusion says that the number of people in either set is equal to the number in the first set plus the number in the second set minus the number in both sets. In formula terms:\n\nNumber of coffee drinkers + Number of tea drinkers - Number of both = Total in union\n\nPlugging in the numbers I have:\n\n29 (coffee) + 28 (tea) - Both = 34\n\nSo, 29 + 28 = 57. 1173 characters

R1-Compress: (Partial)
I need to find out how many teachers drink only coffee and how many drink only tea. This sounds like a problem where I can use sets and maybe a Venn diagram. The total number of teachers is 50. Out of these, 29 drink coffee, 28 drink tea, and 16 don't drink either. **Hmm**, the number of teachers who drink at least one of the beverages should be the total minus those

who don't drink either. Let me calculate that.\n\nTotal teachers = 50\nTeachers who don't drink coffee or tea = 16\nSo, teachers who drink coffee or tea or both = 50 - 16 = 34.\n\nNow, I recall the principle of inclusion-exclusion: \nNumber of coffee drinkers + Number of tea drinkers - Number of both = Total in union\n\nPlugging in the numbers:\n\n29 (coffee) + 28 (tea) - Both = 34\n\nSo, 29 + 28 = 57. 758 characters

As shown in Case 1, R1-Compress is able to successfully compress nearly 30% of the content, while preserving the key computational steps and effectively retaining the reflection component (with "Wait" transformed into "Hmm"). Further analysis shows that the compressed content primarily consists of unnecessary discussions and redundant explanations.

Case Example 2

Problem: Two is 10% of x and 20% of y. What is $x - y$?

(A) 1 (B) 2 (C) 5 (D) 10 (E) 20

Long-CoT: (Partial)

\nOkay, let's see here. The problem says that two is 10% of x and 20% of y. Then we need to find x minus y. Hmm, percentages can sometimes trip me up, but let me take it step by step.\n\nFirst, let's parse the problem. Two is 10% of x. So, in math terms, that would be 2 = 10% of x. Similarly, two is also 20% of y, so 2 = 20% of y. Then, after finding x and y, subtract them to get x - y. The answer choices are numbers from 1 to 20, so it's probably a straightforward calculation once we figure out x and y.\n\nAlright, starting with the first equation: 2 is 10% of x. To translate percentage to decimal, 10% is 0.1. So, 2 = 0.1 * x. 630 characters

.....

Wait, but let me make sure I didn't make any mistakes here. It's easy to mix up percentages. Let me go through the steps again just to confirm.\n\nFirst equation: 10% of x is 2. 10% is 0.1. So, 0.1x = 2. Solving for x: x = 2 0.1 = 20. Correct.\n\nSecond equation: 20% of y is 2. 20% is 0.2. So, 0.2y = 2. Solving for y: y = 2 0.2 = 10. Correct.\n\nSubtracting them: 20 - 10 = 10. Answer choice D. Hmm, seems solid. But the answer choices include 10, which is D, but I wonder if there's a trick here. Sometimes problems might involve something like the percentage of x equals the percentage of y but in different terms. Wait, no, in this case, it's straightforward: two is both 10% of x and 20% of y. So, x and y are different variables, each solved independently. Then subtract. Yeah, that's right.\n\nAlternatively, maybe someone might get confused and think it's the same number, but no, the problem clearly states two is 10% of x and 20% of y. So, x and y have different percentages leading to the same value, 2. Hence, solving for x and y gives different numbers, and their difference is 10.\n\nI don't see any mistakes here. Therefore, the answer should be D) 10.\n\n**Final Answer**\n$\boxed{D}$\n 1193 characters

R1-Compress: (Partial)

Okay, let's see here. Two is 10% of x and 20% of y. We need to find x minus y. Hmm, percentages can trip me up, so I'll take it step by step.\n\nFirst, let's parse the problem. Two is 10% of x, so 2 = 10% of x. Similarly, two is 20% of y, so 2 = 20% of y. Then, we need to find x and y, and subtract them to get x - y.\n\nStarting with the first equation: 2 is 10% of x. 10% is 0.1, so 2 = 0.1 * x. 394 characters

.....

Wait, let me recheck my steps. First, 10% of x is 2, so 0.1x = 2 and x = 2 0.1 = 20. Correct.\n\nNext, 20% of y is 2, so 0.2y = 2 and y = 2 0.2 = 10. Correct.\n\nSubtracting them: 20 - 10 = 10. Answer choice D. Hmm, seems solid. But I wonder if there's a trick. Wait, no, two is both 10% of x and 20% of y, so x and y are different variables. Alternatively, maybe someone might get confused and think it's the same number, but the problem clearly states two is 10% of x and 20% of y, so x and y have different percentages leading to the same value, 2. Hence, solving for x and y gives different numbers, and their difference is 10.\n\nI don't see any mistakes. So, the answer is D) 10.\n\n**Final Answer**\n$\boxed{D}$ 713 characters

As shown in Case 2, R1-Compress is able to successfully compress 236 characters during the problem-solving stage, while retaining the key steps and calculations. Furthermore, it effectively preserves the reflection and verification phases, compressing approximately 500 characters in this stage.

Overall, R1-Compress achieves high-quality compressed CoT by reducing redundant expressions and calculations, while maintaining the essential computations and reflective verification.

E Limitation

Due to the inherent randomness in LLM-based compression—stemming from variations in prompts, different LLMs, and sampling parameters—the quality of each compressed instance cannot be fully guaranteed. Although we generate multiple candidate chunks and apply a search strategy to select high-quality compressed CoT, it is still possible to obtain outputs with contextual incoherence. This work provides insights into Long-CoT compression from the perspective of reflection and, through case studies, reveals that the removed tokens are primarily associated with redundant expressions and repetitive computational steps. Nonetheless, the compression of Long-CoT remains an open problem and warrants further investigation.