

VTC-R1: Vision-Text Compression for Efficient Long-Context Reasoning

Yibo Wang¹, Yongcheng Jing^{1,†}, Shunyu Liu¹, Hao Guan¹, Rong-Cheng Tu¹

Chengyu Wang², Jun Huang², Dacheng Tao^{1,†}

¹Nanyang Technological University ²Alibaba Cloud Computing
{yibowang0021, dacheng.tao}@gmail.com, yongcheng.jing@ntu.edu.sg
{chengyu.wcy, huangjun.hj}@alibaba-inc.com

Abstract

Long-context reasoning has significantly empowered large language models to tackle complex tasks, yet it introduces severe efficiency bottlenecks due to the computational complexity. Existing efficient approaches often rely on complex additional training or external models for compression, which limits scalability and discards critical fine-grained information. In this paper, we first propose VTC-R1, a new efficient reasoning paradigm that integrates vision-text compression into iterative reasoning. Instead of processing lengthy textual traces, VTC-R1 renders intermediate reasoning segments into compact images, which are iteratively fed back into vision-language models as "optical memory". We construct a training dataset based on OpenR1-Math-220K achieving $3.4\times$ token compression and fine-tune representative VLMs—Glyph and Qwen3-VL. Extensive experiments on benchmarks such as MATH500, AIME25, AMC23 and GPQA-D demonstrate that VTC-R1 consistently outperforms standard long-context reasoning. And VTC-R1 improves inference efficiency, achieving $2.7\times$ speedup in end-to-end latency, highlighting its potential as a scalable solution for reasoning-intensive applications. Our code is available at <https://github.com/w-yibo/VTC-R1>.

1 Introduction

Reasoning capability (Li et al., 2025b; Lightman et al., 2023; Yao et al., 2023; Huang and Chang, 2023; Yao et al., 2025) has emerged as a powerful technique of large language models (LLMs), enabling them to tackle complex tasks such as mathematical problem solving (Hendrycks et al., 2021; Luo et al., 2025a; Hu et al., 2025) and code generation (Chen et al., 2021; Jiang et al., 2024). Recent advancements, exemplified by OpenAI o1 (OpenAI, 2024) and DeepSeek-R1 (Guo et al., 2025), leverage reinforcement learning to further scale

[†] Corresponding authors.

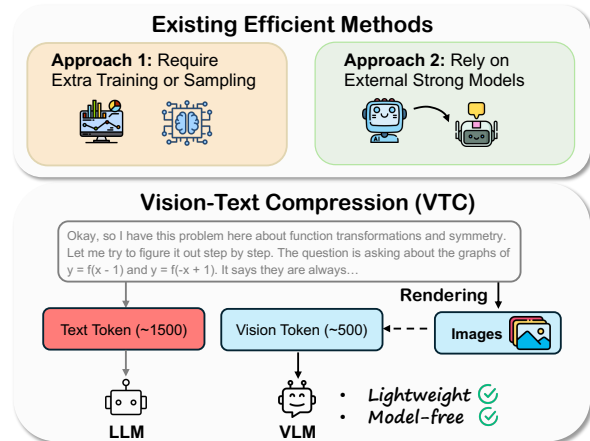


Figure 1: **Comparison between existing efficient reasoning approaches and vision-text compression (VTC).** Existing methods either require additional training or sampling procedures, or rely on external strong models. VTC leverages lightweight rendering to transform long textual reasoning traces into compact visual representations, enabling VLMs to encode information with significantly fewer vision tokens ($3\text{--}4\times$ compression). This approach is both lightweight and model-free.

this capability to long-context reasoning, substantially improving performance on challenging real-world tasks. Despite recent progress, long-context reasoning introduces severe efficiency bottlenecks. The computational complexity of the transformer architecture (Zaheer et al., 2020; Beltagy et al., 2020; Kitaev et al., 2020; Wang et al., 2020) grows quadratically with sequence length, causing both computation and memory costs to increase rapidly as the context expands. This leads to degraded inference speed, reduced training efficiency, and limited scalability, which significantly hinders real-world deployment.

To mitigate these issues, several efficient approaches are proposed (Chen et al., 2025; Munkhbat et al., 2025; Lee et al., 2025; Liu et al., 2024). Existing methods can be broadly categorized into two groups. i) Extra training or

sampling stages beyond standard supervised fine-tuning (SFT). For example, CoT-Valve (Ma et al., 2025b) adopts a multi-stage training procedure to obtain models specialized for different reasoning lengths and O1-Pruner (Luo et al., 2025b) applies offline reinforcement learning with multiple sampled trajectories (16 responses per problem). These approaches increase training and inference cost. ii) External strong models to guide reasoning compression. TokenSkip (Xia et al., 2025) requires an additional model to estimate token importance, while R1-Compress (Wang et al., 2025) and InfityThink (Yan et al., 2025) depend on powerful external summarization models (e.g., Llama-3.3-70B-Instruct) to condense long reasoning traces. Although both categories of methods are effective, beyond the aforementioned costs and dependencies, their efficiency gains are mainly obtained by reducing or compressing the generated reasoning content, which can restrict the model’s exploration space and discard fine-grained intermediate information critical for complex reasoning.

Without additional training or external models, how can we achieve efficient reasoning while preserving fine-grained information?

Motivated by this, a promising yet under-explored direction is vision-text compression (VTC) (Wei et al., 2025; Cheng et al., 2025; Xing et al., 2025b; Zhao et al., 2025; Xing et al., 2025a). Rather than reducing fine-grained information, VTC adopts an alternative representation by transforming textual content into visual forms via lightweight rendering, enabling vision-language models (VLMs) to encode rich semantic information using substantially fewer vision tokens. This design is lightweight and model-free, as shown in Figure 1, introducing no additional training stages or reliance on external compression models. Prior works such as DeepSeek-OCR (Wei et al., 2025) and Glyph (Cheng et al., 2025) focus on text reconstruction or long-context understanding, showing that long text sequences can be represented with $3\times$ – $10\times$ token compression while maintaining high decoding precision. However, whether such high-density visual representations can preserve and support multi-step reasoning processes remains unclear. Notably, mathematical reasoning, with its symbolic structure and step-wise derivations, is naturally amenable to visual rendering, making it a suitable and principled testbed for studying reasoning-oriented vision-text compression.

To bridge this gap, we propose VTC-R1, a new efficient reasoning paradigm that iteratively integrates vision-text compression into long-context reasoning. VTC-R1 treats the reasoning process as multiple processes, where *the preceding process are regarded as long-context and rendered into compact images*, and performs iterative reasoning (Yan et al., 2025) with VLMs. As illustrated in Figure 2, the reasoning process is decomposed into a sequence of reasoning steps. Upon the completion of each step, it is rendered into an image. To proceed to the next step, the accumulated images of previous steps are fed back into the model alongside the question, functioning as a form of optical memory that compactly encodes previous reasoning using vision tokens.

We construct a training dataset based on OpenR1-Math-220K (Hugging Face, 2025), a large-scale long-context reasoning corpus generated by DeepSeek-R1 (Guo et al., 2025). We segment each long reasoning trace into shorter reasoning segments and render the preceding segments into images, forming paired image-text reasoning data with up to $3.4\times$ token compression as shown in Table 1. We then fine-tune representative VTC-VLM (i.e., Glyph (Cheng et al., 2025)) and the state-of-the-art VLM (i.e., Qwen3-VL (Bai et al., 2025)). Extensive experiments on diverse mathematical reasoning benchmarks, GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2023), AIME25 (Zhang and Math-AI, 2025), AMC23 (Math-AI, 2025) and GPQA-Diamond (Rein et al., 2024), demonstrate that VTC-R1 consistently outperforms standard long-context reasoning. Moreover, VTC-R1 significantly improves inference efficiency, achieving up to $2.7\times$ speedup in end-to-end reasoning latency, highlighting its practical advantages for scalable long-context reasoning. The main contributions of this paper are as follows:

- We introduce VTC-R1, the first efficient reasoning paradigm that reformulates reasoning as an iterative process and integrates vision-text compression to replace long text with compact vision tokens, without requiring additional training stages or external strong models.
- We construct a training dataset by segmenting reasoning traces and rendering preceding steps into images, producing paired data with up to $3.4\times$ token compression.
- Extensive evaluation on major mathematical and out-of-distribution benchmarks shows that

VTC-R1 consistently outperforms standard long-context reasoning and achieves up to a $2.7\times$ speedup in end-to-end inference latency.

2 Preliminaries

2.1 Problem Setup

We consider a reasoning task defined by an input question Q . Given a vision language model M , the goal is to produce a final answer A . During answer generation, a long sequence of intermediate reasoning steps is also produced, which forms a **long-context reasoning**.

2.2 Vision-Text Compression

Vision-text compression is defined as a procedure where a given text is rendered into an image, enabling a VLM to encode the content using fewer vision tokens. The pipeline used in our work is summarized as follows.

Given an input text sequence T , the text is rendered into images through a pipeline before model input. The rendering pipeline is parameterized by a configuration vector (Cheng et al., 2025),

$$\theta = (\text{dpi}, \text{page_size}, \text{font_family}, \text{font_size}, \text{line_height}, \text{alignment}, \text{indent}, \text{spacing}, \text{h_scale}, \text{colors}, \text{borders}, \dots), \quad (1)$$

which controls the typography, layout, and visual style of rendered pages. The details of rendering configuration are provided in Appendix B.1. Through the rendering process, multiple PNG images I are produced. This process is defined as $I = R_\theta(T)$, where $R_\theta(\cdot)$ denotes the rendering operator. The images I are processed by the image processor and vision encoder of model M . For simplicity, let M_{vision} denote the vision tokenizer. Given the images I , we obtain a sequence of vision tokens $V = M_{\text{vision}}(I)$, where $V = \{v_1, \dots, v_{L_v}\}$ and L_v represents the sequence length.

The original text T is processed by the text tokenizer M_{txt} to produce a text token sequence $T = M_{\text{txt}}(T)$, where $T = \{t_1, \dots, t_{L_t}\}$ and L_t denotes the number of text tokens.

Thus, the vision-text compression ratio is defined as:

$$\rho = \frac{L_t}{L_v}, \quad (2)$$

In practice, $\rho > 1$, a larger ρ indicates a higher compression efficiency, implying that fewer tokens are required to encode the same content under the vision tokenization scheme.

3 Methodology

3.1 Standard Long-Context Reasoning.

Standard long-context reasoning, as adopted by OpenAI o1 (OpenAI, 2024) and DeepSeek-R1 (Guo et al., 2025), typically produces a long sequence of intermediate reasoning steps. Such behavior incurs substantial computational and memory cost. This reasoning procedure is formulated as a long-context reasoning process, denoted as LR , where the input question is Q . The standard long-context reasoning can be represented as

$$\langle S_s \rangle |U| Q |A| \langle \text{think} \rangle LR \langle / \text{think} \rangle A,$$

where $\langle S_s \rangle$ denotes the standard system prompt, such as “You are a helpful assistant.” The tokens $|U|$ and $|A|$ indicate the start of user input and model response, respectively. The special tokens $\langle \text{think} \rangle$ and $\langle / \text{think} \rangle$ mark the beginning and end of the reasoning process.

In practice, LR may reach 16k tokens or more. During reasoning, the preceding steps could be **regarded as context** and vision-text compression can therefore be introduced to encode these preceding steps into a **smaller number of effective vision tokens**, thereby mitigating the substantial cost of long-context reasoning.

3.2 VTC-R1 Reasoning

Instead of generating a full textual reasoning trace, VTC-R1 first formulates long-context reasoning as an iterative process to get the answer. The long-context reasoning trace LR is decomposed into a sequence of reasoning segments $\{LR_1, \dots, LR_n\}$ under a token-length threshold.

Iterative Reasoning. Concretely, iterative reasoning generates the reasoning process sequentially. At iteration i , the model conditions on the question and the previous segments:

$$\begin{aligned} LR_i &\sim p_\theta(\cdot | Q, LR_{<i}), \\ LR_{<i} &\triangleq (LR_1, \dots, LR_{i-1}). \end{aligned} \quad (3)$$

And the complete trace is obtained by concatenation $LR = (LR_1, \dots, LR_n)$.

We next show that *this iterative formulation is equivalent to standard one-pass long-context generation under an autoregressive model*. By the chain rule, the joint distribution of the full trace factorizes as

$$p_\theta(LR | Q) = \prod_{i=1}^n p_\theta(LR_i | Q, LR_{<i}), \quad (4)$$

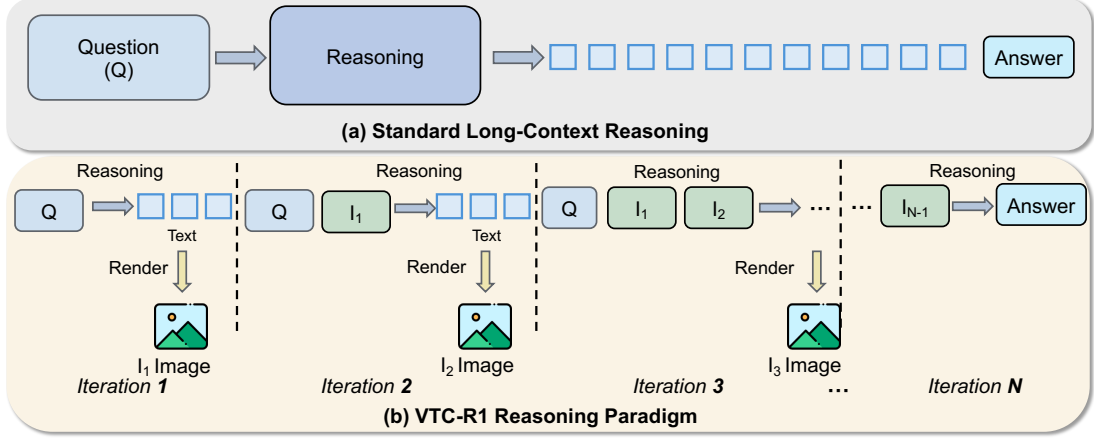


Figure 2: Comparison between standard long-context reasoning and the proposed VTC-R1 reasoning paradigm. **(a)** Standard long-context reasoning processes the entire reasoning trace as a single long sequence, leading to increasing computational and memory costs as the context grows. **(b)** VTC-R1 reformulates long-context reasoning as an iterative process. At each iteration, the current reasoning segment is generated and the preceding segments are rendered into compact images, which are fed back to the model together with the original question. These rendered images function as a form of optical memory, enabling efficient multi-step reasoning with reduced token usage.

which is exactly the distribution induced by sampling $LR_1, \dots, LR_n$ sequentially with the same conditionals. Consequently, for any answer extraction function $A = M(LR)$, one-pass and iterative generation yield the same answer distribution:

$$A = M(LR), \quad LR \sim p_\theta(\cdot | Q).$$

VTC-R1 Reasoning Paradigm. The first reasoning process is expressed as follows, where $n > 1$ is assumed:

$$\langle S_v \rangle | U | Q | A | \langle \text{think} \rangle LR_1 \langle / \text{think} \rangle,$$

where $\langle S_v \rangle$ denotes the VTC-R1 system prompt.

VTC-R1 System Prompt $\langle S_v \rangle$

These images record your previous reasoning process. Based on this reasoning, continue and complete the final answer. Do not restart the reasoning.

If no images are provided, start the reasoning from scratch.

As described in Sec 2.2, the first reasoning process LR_1 is rendered into multiple images, $I_1 = R_\theta(LR_1)$.

When the i -th reasoning process begins, $i-1$ reasoning processes have been completed. At the end of each process, the generated reasoning process LR_j is rendered into multiple images I_j and stored. As a result, a set of rendered images $\{I_1, \dots, I_{i-1}\}$ is available. The reasoning process at the i -th itera-

tion is then expressed as

$$\langle S_v \rangle | U | Q, I_1, \dots, I_{i-1} | A | \langle \text{think} \rangle LR_i \langle / \text{think} \rangle.$$

At the final reasoning iteration n , the model produces the last reasoning segment and outputs the final answer A . The complete generation at this stage is expressed as

$$\langle S_v \rangle | U | \{Q, I_1, \dots, I_{n-1}\} | A | \langle \text{think} \rangle LR_n \langle / \text{think} \rangle A.$$

During inference, VTC-R1 iterates continuously until the final answer A is produced. As shown in Table 2, the method exhibits adaptive reasoning behavior, where the number of reasoning iterations is selected dynamically according to the problem difficulty. To prevent unbounded generation, a maximum iteration limit, denoted as T , is imposed.

VTC-R1 performs iterative reasoning by generating multiple reasoning segments in Algorithm 1. At each iteration, the previously generated reasoning segments $LR_1, \dots, LR_{i-1}$ are rendered into images $I_1, \dots, I_{i-1}$. Therefore, these images provide a compact and efficient representation of textual reasoning through vision tokens, functioning analogously to an optical memory. Under our rendering configuration, the resulting **compression ratio** ρ is **approximately 3–4** as shown in Table 1, which could mitigate the computational and memory cost incurred by token growth in standard method.

Moreover, VTC-R1 requires a lightweight rendering mechanism. No additional training, extra sampling stages, or external models are introduced.

Algorithm 1 VTC-R1 Reasoning Paradigm

Input: question Q ; vision language model M ;
system prompt $\langle S_v \rangle$; rendering operator R_θ ;
maximum iteration T

Initialize: rendered image set $\mathcal{I} \leftarrow \emptyset$

for $i = 1$ to T **do**

 Generate Vision-Language Model Output:

$$O_i \leftarrow M(\langle S_v \rangle, Q, \mathcal{I})$$

if O_i produces the final answer A **then**

return A

end if

 Update Image Set via Rendering:

 Extract reasoning progress LR_i from O_i

 Render reasoning into images: $I_i \leftarrow R_\theta(LR_i)$

 Update: $\mathcal{I} \leftarrow \mathcal{I} \cup \{I_i\}$

end for

if no final answer A **then**

 Extract Answer when Reaching Iteration
 Limit:

 Extract final answer A from O_T

end if

Output: final answer A

Metric	Value
Rendered reasoning steps (elements)	45K
Rendered images	105K
Total text tokens	181M
Total vision tokens	54M
Compression ratio (text / vision)	3.4 $\times$

Table 1: **Statistics of rendered prior reasoning segments.** We report the number of reasoning segments rendered as images, the total number of text and vision tokens, and the compression ratio.

Batch Inference. To support batch inference in frameworks such as vLLM (Kwon et al., 2023), we extend Algorithm 1 with independent *request states* and a *dynamic active set*. This enables parallel generation by batching only active samples and updating their multimodal contexts at each iteration. Details are given in Algorithm 2.

3.3 Training Data Construction

To train VTC-R1, a supervised fine-tuning dataset is constructed to enable VLMs to learn the VTC-R1 reasoning paradigm. The dataset is organized as an image-text paired corpus. We adopt OpenR1-Math-Inf (Yan et al., 2025), which is a subset of OpenR1-Math-220K (Hugging Face, 2025). OpenR1-Math-

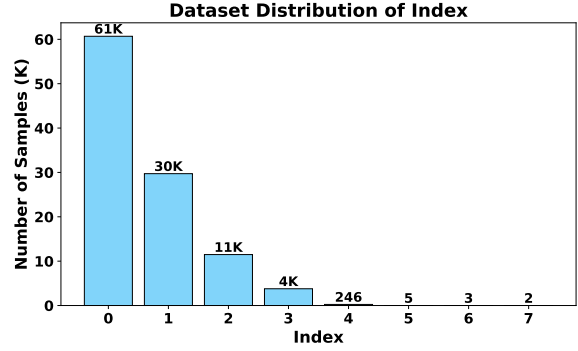


Figure 3: **Distribution of data index.** The index indicates the order of a reasoning segment for a given problem, where index 0 corresponds to the first segment. Most samples terminate at early steps, while a small fraction requires more than four iterations.

220K is generated by the DeepSeek-R1 (Guo et al., 2025) model, where solutions are produced for large-scale mathematical problems. OpenR1-Math-Inf contains 61K question-answer pairs, and each solution is partitioned into multiple reasoning segments $\{LR_1, LR_2, \dots, LR_n\}$ according to predefined thresholds.

Based on Sec 3.2, training data are constructed according to the index of the reasoning process, where different rules are applied at different iterations. Rendered images are included as inputs. The instance at iteration i is defined as

$$D_i = \begin{cases} (\langle S_v \rangle, Q, \emptyset, LR_1), & i = 1, \\ (\langle S_v \rangle, Q, \{I_j\}_{j < i}, LR_i), & 1 < i < n, \\ (\langle S_v \rangle, Q, \{I_j\}_{j < i}, LR_n, A), & i = n. \end{cases} \quad (5)$$

106K instances are constructed based on Eq. 5, which requires approximately 105K rendered images in PNG format. Figure 3 presents the segment index distribution in the constructed training data. Table 1 reports the token statistics after applying vision-text compression. The original reasoning traces contain 181M text tokens, which are reduced to 54M vision tokens after rendering, achieving a compression ratio of up to **3.4** $\times$. This dataset is used for supervised fine-tuning. It is noted that the number of images associated with each instance is adaptive. Therefore, the training procedure requires VLM architectures that support inputs with a variable number and resolution of images, such as Qwen3-VL (Bai et al., 2025), GLM-4.1V (Team et al., 2025a), and Glyph (Cheng et al., 2025).

Model	GSM8K			MATH500			AIME25 (Avg@16)			AMC23 (Avg@16)		
	ACC ↑	TOK	LAT ↓	ACC ↑	TOK	LAT ↓	ACC ↑	TOK	LAT ↓	ACC ↑	TOK	LAT ↓
<i>Qwen3-VL-8B</i>												
SFT	88.1	1.79	3.04	85.4	4.17	5.36	32.71	17.46	29.85	75.00	8.20	11.08
VTC-R1	94.7 ^(+6.6)	1.09	0.46 ^(6.6×)	90.0 ^(+4.6)	3.39	2.49 ^(2.2×)	30.00 ^(−2.71)	14.32	12.02 ^(2.5×)	77.97 ^(+2.97)	8.18	6.45 ^(1.7×)
<i>Glyph</i>												
Base SFT	86.1	2.35	1.38	79.6	5.51	2.77	24.17	19.94	14.48	61.56	12.67	8.55
SFT	87.1	1.87	0.93	80.4	5.71	3.05	25.62	17.47	11.52	60.94	11.65	6.85
TokenSkip	86.4	2.25	1.32	80.6	6.11	3.05	23.75	17.82	11.85	59.53	12.81	8.41
VTC-R1	93.6 ^(+6.5)	1.09	0.34 ^(2.7×)	86.0 ^(+5.6)	4.12	2.19 ^(1.4×)	26.25 ^(+0.63)	12.95	6.81 ^(1.7×)	64.38 ^(+3.44)	8.81	4.30 ^(1.6×)

Table 2: **Performance comparison across mathematical benchmarks.** Accuracy (ACC) is higher-is-better (↑), latency (LAT) is lower-is-better (↓). **Bold** indicates the best performance. Superscript numbers denote accuracy improvements and latency speedups relative to standard long-context reasoning.

4 Experiments

4.1 Experiment Settings

Dataset. For training, we use the OpenR1-Math-Inf (Yan et al., 2025), where each solution is segmented into multiple reasoning segments with varying lengths (2K, 4K, and 6K tokens). It is a subset of OpenR1-Math-220K (Hugging Face, 2025) dataset. Unless otherwise specified, 4K is used as the default segmentation setting. For evaluation, we leverage four widely used mathematical reasoning benchmarks. GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2023), AIME25 (Zhang and Math-AI, 2025) and AMC23 (Math-AI, 2025). And GPQA-Diamond (GPQA-D) (Rein et al., 2024), a science-domain benchmark that serves as an *out-of-distribution evaluation*. See the Appendix C.2 for more details of benchmarks.

Baseline. For baselines, our proposed method VTC-R1 is compared with standard long-context reasoning (SFT). In the SFT setting, standard question-answer pairs with full long-form reasoning traces are used as the supervised fine-tuning dataset. We then perform VTC-R1 and SFT on two representative VLM architectures respectively for comparison. i) Glyph (Cheng et al., 2025), which serves as a VTC-capable VLM. ii) Qwen3-VL-8B (Bai et al., 2025), which represents a mainstream vision-language model. In addition, standard SFT does not require optical character recognition capability. Therefore, the base model preceding Glyph, GLM-4.1V-9B-Base (Team et al., 2025a) (Base SFT), is also included as a baseline. The efficient reasoning method TokenSkip (Xia et al., 2025) is included as an additional baseline for comparison.

Metric. We employ the following three metrics to evaluate the model’s performance.

- **Accuracy (ACC):** For GSM8K, MATH500, and GPQA-Diamond, we report pass@1 accuracy.

For AIME25 and AMC23, due to their limited dataset sizes, we generate 16 responses per problem and report avg@16 accuracy.

- **Token (TOK):** The average number of generated response tokens, reported in thousands (K).
- **Latency (LAT):** We measure the average inference latency per generation, reported in seconds (s). Given a dataset with m problems, where each problem is generated n times (e.g., $n = 16$ for AIME25 and AMC23), let t_1 and t_2 denote the wall-clock timestamps at the start and end of the entire inference process, respectively. The latency is computed as:

$$LAT = \frac{t_2 - t_1}{m \times n}.$$

Implementation Details. For both SFT and VTC-R1, the processed training datasets contain 106K instances and require approximately 105K images. Both methods are trained with a learning rate of 1×10^{-5} for one epoch using the LlamaFactory library (Zheng et al., 2024). For evaluation, a temperature of 0.6 and a top- p value of 0.95 are adopted under the vLLM framework (Kwon et al., 2023).

4.2 Main Results

Performance Gains. As shown in Table 2, VTC-R1 consistently outperforms Base SFT, SFT and TokenSkip baselines on the Glyph across all four benchmarks. Notably, substantial improvements are observed on the more challenging benchmarks, with the gains of 5.6% on MATH500 and 3.4% on AMC23. On the Qwen3-VL architecture, VTC-R1 also demonstrates consistent improvements or achieves competitive accuracy compared to standard long-context reasoning.

Furthermore, as reported in Table 3, similar trends are observed on the out-of-distribution benchmark. Specifically, VTC-R1 yields accuracy

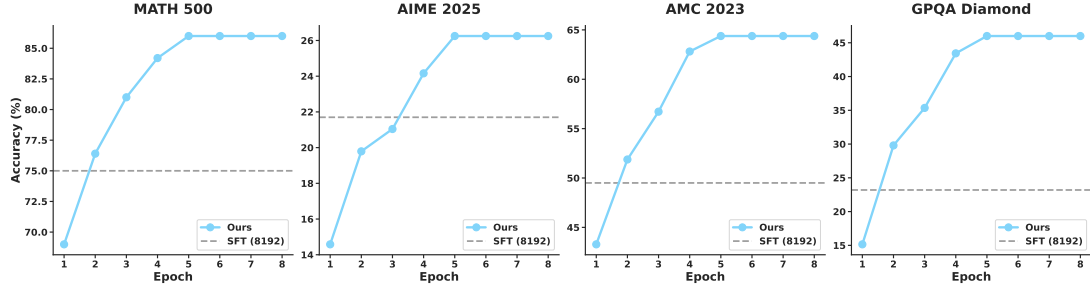


Figure 4: Accuracy of the proposed method across benchmarks under different maximum iteration epochs. The epoch index denotes the maximum number of allowed reasoning iterations. The dashed line indicates the single-round baseline (standard long-context reasoning for 8192 maximum tokens).

Model	GPQA-Diamond		
	ACC $\uparrow$	TOK	LAT $\downarrow$
<i>Qwen3-VL-8B</i>			
SFT	37.4	14.78	26.88
VTC-R1	48.5 (+11.1)	9.77	9.57 (2.8 $\times$)
<i>Glyph</i>			
Base SFT	26.3	19.74	14.43
SFT	38.4	13.91	8.35
TokenSkip	35.9	15.45	9.93
VTC-R1	46.0 (+7.6)	10.73	6.96 (1.2 $\times$)

Table 3: Performance on out-of-distribution benchmark. **Bold** indicates the best performance.

improvements of 7.6% and 11.1%, indicating that the proposed approach generalizes effectively beyond in-distribution mathematical benchmarks.

Efficient Inference. VTC-R1 achieves efficient inference latency across model architectures. On the Glyph architecture, a speedup of at least $1.4\times$ is observed across all benchmarks, with larger gains of $1.7\times$ and $1.6\times$ on the more challenging benchmarks. On the Qwen3-VL architecture, the inference speedup reaches up to $6.6\times$.

Although the proposed method is not explicitly designed as an adaptive reasoning framework, adaptivity emerges from the data construction process, where different problems are associated with different numbers of reasoning iterations. Thus, benchmarks of varying difficulty exhibit different effective token lengths. For instance, GSM8K requires fewer tokens, while AIME25 involves longer token sequences and more iteration epochs in Figure 4.

The latency speedup consistently exceeds the reduction in token count. For example, on the Glyph for AMC23, the token count is reduced by approximately $1.3\times$, whereas the latency improvement reaches $1.6\times$. This discrepancy indicates that the introduction of vision-text compression provides additional efficiency gains beyond token reduction.

Iteration Epochs. Figure 4 illustrates the accuracy of the proposed method across four bench-

marks under different epoch budgets. The epoch index denotes the maximum number of allowed reasoning iterations, where each iteration is a single decoding pass in which the model autonomously stops without emitting the final answer; this stopping behavior is learned from partitioning training reasoning traces under a token-length threshold. As shown in the figure, the accuracy consistently improves as the maximum epoch increases, demonstrating the effectiveness of multi-iteration reasoning. Across most benchmarks, the rate of accuracy improvement gradually diminishes, and performance begins to converge from approximately the fifth epoch onward. This observation indicates that the proposed method benefits from additional reasoning iterations while exhibiting stable convergence behavior.

Overcoming Training Context Limitations. The gray dashed line in Figure 4 denotes the inference accuracy of standard long-context reasoning when the maximum number of newly generated tokens is set to 8,192, which also corresponds to the maximum token length used during training for our method. As the number of iteration epochs increases, the accuracy of the proposed method gradually surpasses the baseline across benchmarks. This result indicates that the proposed method is able to overcome the context length limitation imposed during training and achieve higher inference accuracy beyond the fixed training window. At the same time, efficient training is maintained, as evidenced by the reduced training cost in Table 6.

4.3 Ablation Study

Segment Length. Table 4 reports the performance across benchmarks when different segmentation lengths (2K, 4K, and 6K) are used during training data construction, where 4K serves as the default setting. Across all four benchmarks, a segmentation length of 4K achieves the best or highly

Segment Length	GSM8K		MATH500		AIME25		AMC23	
	ACC $\uparrow$	LAT $\downarrow$	ACC $\uparrow$	LAT $\downarrow$	ACC $\uparrow$	LAT $\downarrow$	ACC $\uparrow$	LAT $\downarrow$
2K	93.9	0.39	82.4	1.95	20.6	5.87	59.7	4.03
4K	93.6	0.34	86.0	2.19	26.2	6.81	64.3	4.30
6K	93.7	0.92	84.2	3.28	23.5	8.06	64.7	4.90

Table 4: Effect of segment length on accuracy (ACC) and latency (LAT) across benchmarks. Higher ACC and lower LAT indicate better performance. Best results for each metric are highlighted in bold.

	AIME25	AMC23	GPQA-D
Baseline	26.25	64.38	46.0
w/o Image	23.33 ^(-11.1%)	59.53 ^(-7.5%)	34.3 ^(-25.4%)

Table 5: Performance comparison with and without image input. - denotes the relative performance drop.

Method	Training Time (h)
SFT	38.92
VTC-R1	18.93

Table 6: Training time across different methods.

competitive accuracy. In addition, on MATH500, AIME25, and AMC23, the latency (LAT) increases as the segmentation length grows. This behavior is expected, since larger segmentation lengths gradually approach standard long-context reasoning, which incurs higher inference cost due to longer effective reasoning sequences.

Image Input. We further analyze the performance of VTC-R1 when image inputs are removed at each reasoning iteration. Three more challenging benchmarks AIME25, AMC23, and GPQA-D, are selected for this analysis, which are more likely to benefit from multi-step reasoning.

As shown in Table 5, removing image inputs leads to accuracy drops of 11.1% and 7.5% on AIME25 and AMC23, with a more substantial degradation of 25.4% observed on GPQA-D. These results indicate that VTC-R1 relies on rendered images as a form of memory for previous reasoning steps during inference. At the same time, a non-trivial level of accuracy is retained even without image inputs. This can be attributed to the fact that many problems can be solved within a single reasoning iteration; in the absence of image conditioning, the model restarts the reasoning process from scratch and can still obtain correct answers.

System Prompt. We further examine the sensitivity of VTC-R1 to the phrasing of the system prompt on MATH500, comparing the original prompt, a minimal variant that only instructs the model to continue its reasoning and to start from scratch when no images are provided, and a setting without any system prompt. As shown

Variant	ACC $\uparrow$	TOK	LAT $\downarrow$
original	86.0	4.08	2.19
minimal	84.8	4.11	2.38
no prompt	83.8	4.94	2.67

Table 7: Effect of system prompt phrasing.

in Table 7, performance remains stable across the different wordings, with no significant difference observed. This indicates that VTC-R1 is not sensitive to prompt phrasing.

4.4 Efficiency Analysis

Training Efficiency. Table 6 reports the training time of VTC-R1 in comparison with Base SFT and SFT. All training times are measured using the LlamaFactory framework under same configuration. Although the proposed method adopts a multi-iteration training paradigm and therefore introduces more QA pairs as well as additional images, the overall training time is reduced to approximately 48% of that required by the baseline methods. This result demonstrates the training efficiency of VTC-R1. And the final performance of VTC-R1 is superior as shown in Table 2. The reduction in training time is attributed to the standard long-context reasoning involves substantially longer reasoning sequences, where training cost increases rapidly as the reasoning length grows. In contrast, VTC-R1 constrains the reasoning length within each iteration to a controlled range, which leads to improved training efficiency.

Rendering Efficiency. Table 2 shows that VTC-R1 significantly outperforms all baselines in terms of end-to-end latency, where the reported metric already accounts for the overhead of rendering and image processing. We further provide fine-grained statistics to validate that the introduced vision-text compression mechanism is lightweight. Based on an analysis of 100 samples from the dataset, we observe that for an average of approximately 1,600 text tokens per image, the rendering process requires only 0.12s on average, while image processing takes merely 0.02s. Compared to the overall model inference latency, this additional overhead is negligible (4% of the total latency). Moreover,

#Accumulated images	0	2	6	10	14
Mean prefill (s)	0.02	0.26	0.33	0.41	0.35

Table 8: Mean prefill latency versus the number of accumulated images.

the average generated image size is around 0.1 MB, which falls within a manageable range.

Caching Efficiency. Since the accumulated-image prompt is a strict prefix extension, repeated images are prefilled only once rather than in every round, reused through vLLM’s multimodal processor cache and prefix KV caching. In Table 8, the mean prefill latency stays nearly constant from 2 to 14 accumulated images, confirming that previous images are reused rather than re-encoded, which would otherwise grow the latency roughly linearly.

5 Related Work

Reasoning in Large Language Models. Reasoning capabilities (Li et al., 2025b; Lightman et al., 2023; Huang and Chang, 2023; Yao et al., 2025) constitute a cornerstone of modern LLMs, enabling proficiency in rigorous domains like mathematics (Hendrycks et al., 2021; Luo et al., 2025a; Hu et al., 2025) and code generation (Chen et al., 2021; Jiang et al., 2024). While early strategies relied on structured prompting (Yao et al., 2023, 2024), recent advancements leverage reinforcement learning to scale test-time compute. Models such as OpenAI o1 (OpenAI, 2024), DeepSeek-R1 (Guo et al., 2025), and Kimi (Team et al., 2025b) generate extended chains of thought, achieving significant improvements on challenging real-world benchmarks.

Efficient Reasoning. Long-context reasoning strategies exacerbate the computational bottlenecks inherent in the quadratic complexity of Transformer architectures (Zaheer et al., 2020; Beltagy et al., 2020; Kitaev et al., 2020; Wang et al., 2020). Recent research has investigated various efficiency mechanisms (Liu et al., 2025; Chen et al., 2025; Munkhbat et al., 2025; Lee et al., 2025; Liu et al., 2024; Yang et al., 2025c; Zhang et al., 2025; Hao et al., 2024; Yang et al., 2025a; Pan et al., 2025; Ma et al., 2025a; Qiao et al., 2025; Zhuang et al., 2025; Yang et al., 2025b; Hou et al., 2025; Ning et al., 2025; Li et al., 2025a; Gong et al., 2025), though existing methods often incur significant trade-offs. One category of approaches (Team et al., 2025b), exemplified by CoT-Valve (Ma et al., 2025b) and O1-Pruner (Luo et al., 2025b), relies on complex multi-stage training procedures or extensive offline sampling, which substantially increases

pre-deployment overhead. A second category leverages external strong models (Kang et al., 2024) to guide reasoning compression, as in TokenSkip (Xia et al., 2025), R1-Compress (Wang et al., 2025), and InfTyThink (Yan et al., 2025), making the compression quality dependent on the capabilities of these auxiliary models. Although effective in reducing token counts, these approaches often constrain the exploration space and risk discarding fine-grained information that is critical for logical deduction.

Vision-Text Compression. Vision-text compression (VTC) (Wei et al., 2025; Cheng et al., 2025; Xing et al., 2025b; Zhao et al., 2025; Xing et al., 2025a; Wang et al., 2024) has emerged as a promising approach for reducing the cost of processing long textual sequences by transforming text into compact visual representations. DeepSeek-OCR (Wei et al., 2025) demonstrates that long texts can be compressed into visual tokens, achieving a $3\times$ – $10\times$ reduction in token count while maintaining high decoding fidelity. Glyph (Cheng et al., 2025) utilizes continuous pre-training and RL for VTC to enhance long-context understanding capabilities. VTCBench (Zhao et al., 2025) proposes a benchmark to evaluate the spectrum of capabilities in VTC. While prior work focuses on text understanding and reconstruction, and it remains unclear whether such high-density visual representations can faithfully preserve and support complex reasoning processes, particularly for mathematically intensive and multi-step reasoning tasks.

Some concurrent works, AgentOCR (Feng et al., 2026) utilizes VTC to compress the agent’s history derived from tool invocations into a compact rendered image. RoT (Wang et al., 2026) focuses on utilizing rendered visual tokens as latent tokens for latent reasoning, but it does not explicitly address long-context reasoning and lacks systematic evaluation on challenging benchmarks.

6 Conclusion

We propose VTC-R1, first efficient long-context reasoning that compresses previous reasoning segments into visual representations for iterative reasoning. This lightweight, model-free design replaces long contexts with compact vision tokens. Experiments show that VTC-R1 improves reasoning accuracy across benchmarks while achieving $3.4\times$ token compression and $2.7\times$ speedup. These results demonstrate the potential of visual representations for scalable long-context reasoning.

Limitations

While VTC-R1 demonstrates promising results in long-context reasoning, several limitations warrant discussion and suggest directions for future work.

VTC-R1 relies on off-the-shelf VLMs, such as Glyph, which is trained for OCR-intensive tasks, and the advanced VLM Qwen3-VL, GLM4.1-V. Although our method is intended to be model-agnostic, we have not yet evaluated it across a broader range of architectures.

Acknowledgments

This research is supported by the RIE2025 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) (Award I2301E0026), administered by A*STAR, as well as supported by Alibaba Group and NTU Singapore through Alibaba-NTU Global e-Sustainability CorpLab (ANGEL).

References

- Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhi-fang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *Preprint*, arXiv:2107.03374.
- 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. 2025. [Do not think that much for 2+3=? on the overthinking of o1-like llms](#). *Preprint*, arXiv:2412.21187.
- Jiale Cheng, Yusen Liu, Xinyu Zhang, Yulin Fei, Wenyi Hong, Ruiliang Lyu, Weihang Wang, Zhe Su, Xiaotao Gu, Xiao Liu, Yushi Bai, Jie Tang, Hongning Wang, and Minlie Huang. 2025. Glyph: Scaling context windows via visual-text compression. *arXiv preprint arXiv:2510.17800*.
- 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. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Lang Feng, Fuchao Yang, Feng Chen, Xin Cheng, Haiyang Xu, Zhenglin Wan, Ming Yan, and Bo An. 2026. Agentocr: Reimagining agent history via optical self-compression. *arXiv preprint arXiv:2601.04786*.
- Ruihan Gong, Yue Liu, Wenjie Qu, Mingzhe Du, Yufei He, Yingwei Ma, Yulin Chen, Xiang Liu, Yi Wen, Xinfeng Li, Ruidong Wang, Xinzhang Zhu, Bryan Hooi, and Jiaheng Zhang. 2025. [Efficient reasoning via chain of unconscious thought](#). *Preprint*, arXiv:2505.19756.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. [Training large language models to reason in a continuous latent space](#). *Preprint*, arXiv:2412.06769.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.
- Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. 2025. [Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning](#). *Preprint*, arXiv:2504.01296.
- Zhiyuan Hu, Yibo Wang, Hanze Dong, Yuhui Xu, Amrita Saha, Caiming Xiong, Bryan Hooi, and Junnan Li. 2025. Beyond ‘aha!’: Toward systematic meta-abilities alignment in large reasoning models. *arXiv preprint arXiv:2505.10554*.
- Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards reasoning in large language models: A survey. In *Findings of the association for computational linguistics: ACL 2023*, pages 1049–1065.
- Hugging Face. 2025. [Open r1: A fully open reproduction of deepseek-r1](#).
- Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.
- Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2024. [C3ot: Generating shorter chain-of-thought without compromising effectiveness](#). *Preprint*, arXiv:2412.11664.
- Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. [Reformer: The efficient transformer](#). In *International Conference on Learning Representations*.

- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Ayeong Lee, Ethan Che, and Tianyi Peng. 2025. How well do llms compress their own chain-of-thought? a token complexity approach. *arXiv preprint arXiv:2503.01141*.
- Zhong-Zhi Li, Xiao Liang, Zihao Tang, Lei Ji, Peijie Wang, Haotian Xu, Xing W, Haizhen Huang, Weiwei Deng, Yeyun Gong, Zhijiang Guo, Xiao Liu, Fei Yin, and Cheng-Lin Liu. 2025a. [TL;dr: Too long, do re-weighting for efficient llm reasoning compression](#). *Preprint*, arXiv:2506.02678.
- Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, Yingying Zhang, Fei Yin, Jiahua Dong, Zhiwei Li, Bao-Long Bi, Ling-Rui Mei, Junfeng Fang, Zhijiang Guo, Le Song, and Cheng-Lin Liu. 2025b. [From system 1 to system 2: A survey of reasoning large language models](#). *Preprint*, arXiv:2502.17419.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*.
- Tianqiao Liu, Zui Chen, Zitao Liu, Mi Tian, and Weiqi Luo. 2024. Expediting and elevating large language model reasoning via hidden chain-of-thought decoding. *arXiv preprint arXiv:2409.08561*.
- Yue Liu, Jiaying Wu, Yufei He, Hongcheng Gao, Hongyu Chen, Baolong Bi, Jiaheng Zhang, Zhiqi Huang, and Bryan Hooi. 2025. Efficient inference for large reasoning models: A survey. *arXiv preprint arXiv:2503.23077*.
- Haotian Luo, Haiying He, Yibo Wang, Jinluan Yang, Rui Liu, Naiqiang Tan, Xiaochun Cao, Dacheng Tao, and Li Shen. 2025a. [Adar1: From long-cot to hybrid-cot via bi-level adaptive reasoning optimization](#). *Preprint*, arXiv:2504.21659.
- Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025b. [O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning](#). *Preprint*, arXiv:2501.12570.
- Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. 2025a. [Reasoning models can be effective without thinking](#). *Preprint*, arXiv:2504.09858.
- Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025b. [Cot-valve: Length-compressible chain-of-thought tuning](#). *Preprint*, arXiv:2502.09601.
- Math-AI. 2025. [Amc23 dataset](#).
- Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. 2025. [Self-training elicits concise reasoning in large language models](#). *Preprint*, arXiv:2502.20122.
- Yansong Ning, Wei Li, Jun Fang, Naiqiang Tan, and Hao Liu. 2025. [Not all thoughts are generated equal: Efficient llm reasoning via multi-turn reinforcement learning](#). *Preprint*, arXiv:2505.11827.
- OpenAI. 2024. Learning to reason with llms. <https://openai.com/index/learning-to-reason-with-llms/>. [Accessed 19-09-2024].
- Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. 2025. [Learning adaptive parallel reasoning with language models](#). *Preprint*, arXiv:2504.15466.
- Ziqing Qiao, Yongheng Deng, Jiali Zeng, Dong Wang, Lai Wei, Fandong Meng, Jie Zhou, Ju Ren, and Yaoxue Zhang. 2025. [Concise: Confidence-guided compression in step-by-step efficient reasoning](#). *Preprint*, arXiv:2505.04881.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. [GPQA: A graduate-level google-proof q&a benchmark](#). In *First Conference on Language Modeling*.
- GLM-V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihang Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, and 69 others. 2025a. [Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning](#). *Preprint*, arXiv:2507.01006.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025b. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.
- Alex Jinpeng Wang, Linjie Li, Yiqi Lin, Min Li, Lijuan Wang, and Mike Zheng Shou. 2024. Leveraging visual tokens for extended text contexts in multi-modal learning. *NeurIPS*.
- Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. [Linformer: Self-attention with linear complexity](#). *Preprint*, arXiv:2006.04768.
- Yibo Wang, Haotian Luo, Huanjin Yao, Tiansheng Huang, Haiying He, Rui Liu, Naiqiang Tan, Jiaying Huang, Xiaochun Cao, Dacheng Tao, and 1 others. 2025. R1-compress: Long chain-of-thought compression via chunk compression and search. *arXiv preprint arXiv:2505.16838*.

- Yifan Wang, Shiyu Li, Peiming Li, Xiaochen Yang, Yang Tang, and Zheng Wei. 2026. Render-of-thought: Rendering textual chain-of-thought as images for visual latent reasoning. *arXiv preprint arXiv:2601.14750*.
- Haoran Wei, Yaofeng Sun, and Yukun Li. 2025. Deepseek-ocr: Contexts optical compression. *arXiv preprint arXiv:2510.18234*.
- Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. 2025. [Tokenskip: Controllable chain-of-thought compression in llms](#). *Preprint*, arXiv:2502.12067.
- Ling Xing, Alex Jinpeng Wang, Rui Yan, Hongyu Qu, Zechao Li, and Jinhui Tang. 2025a. See the text: From tokenization to visual reading. *arXiv preprint arXiv:2510.18840*.
- Ling Xing, Alex Jinpeng Wang, Rui Yan, Xiangbo Shu, and Jinhui Tang. 2025b. Vision-centric token compression in large language model. *arXiv preprint arXiv:2502.00791*.
- Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. 2025. [Infythink: Breaking the length limits of long-context reasoning in large language models](#). *Preprint*, arXiv:2503.06692.
- Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Zheng Lin, Li Cao, and Weiping Wang. 2025a. [Dynamic early exit in reasoning models](#). *Preprint*, arXiv:2504.15895.
- Junjie Yang, Ke Lin, and Xing Yu. 2025b. [Think when you need: Self-adaptive chain-of-thought learning](#). *Preprint*, arXiv:2504.03234.
- Wang Yang, Xiang Yue, Vipin Chaudhary, and Xiaotian Han. 2025c. [Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time](#). *Preprint*, arXiv:2504.12329.
- Huanjin Yao, Jiaxing Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, and Dacheng Tao. 2024. [Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search](#). *Preprint*, arXiv:2412.18319.
- Huanjin Yao, Ruifei Zhang, Jiaxing Huang, Jingyi Zhang, Yibo Wang, Bo Fang, Ruolin Zhu, Yongcheng Jing, Shunyu Liu, Guanbin Li, and 1 others. 2025. A survey on agentic multimodal large language models. *arXiv preprint arXiv:2510.10991*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and 1 others. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.
- Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. [Lightthinker: Thinking step-by-step compression](#). *Preprint*, arXiv:2502.15589.
- Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025.
- Hongbo Zhao, Meng Wang, Fei Zhu, Wenzhuo Liu, Bolin Ni, Fanhu Zeng, Gaofeng Meng, and Zhaoxiang Zhang. 2025. [Vtcbench: Can vision-language models understand long context with vision-text compression?](#) *Preprint*, arXiv:2512.15649.
- Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. 2024. [Llamafactory: Unified efficient fine-tuning of 100+ language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.
- Ren Zhuang, Ben Wang, and Shuifa Sun. 2025. [Accelerating chain-of-thought reasoning: When goal-gradient importance meets dynamic skipping](#). *Preprint*, arXiv:2505.08392.

A Usage of AI Assistant

We use ChatGPT solely for language refinement of the manuscripts text. All conceptual content, experimental design, analysis, and conclusions are developed entirely by the authors. We carefully review the AI-assisted edits to ensure that the meaning and technical accuracy of the original text are fully preserved.

B Image Rendering

B.1 Rendering Configuration

The rendering pipeline is parameterized by a configuration vector. Following (Cheng et al., 2025), a set of rendering configuration factors is adopted, as summarized in Table 9. These factors determine the final rendering properties, including layout, visual clarity, and typography.

The default configuration used in our experiments is reported in Figure 5. This configuration largely follows the default settings of Glyph. However, since the default Glyph font may produce incorrect glyphs when rendering certain mathematical symbols, the font is replaced with `DejaVuSans.ttf` in our implementation.

B.2 Rendering Example

Figure 6 presents an example image rendered under the default configuration specified in Figure 5.

B.3 Rendering-Hyperparameter Sensitivity

To assess the robustness of VTC-R1 to the rendering configuration, we take the VTC-R1 (Glyph) and vary the inference-time rendering configuration along three axes, namely DPI, font size, and image resolution (page width). We report accuracy, average generated tokens, and average latency on MATH500 and AIME25, while keeping the remaining rendering factors fixed at their default values (DPI 72, font size 9, and page width 595). The results in Table 10 lead to three observations.

First, accuracy is robust to rendering hyperparameters overall. Across all 16 configurations, MATH500 stays within 82.6–86.2 and AIME25 within 21.5–28.1, with no collapse point along any axis, indicating that VTC-R1 does not depend on fine-grained rendering hyperparameters. *Second, overly small parameters consistently hurt performance*, which provides direct evidence for the readability hypothesis: the lowest DPI (48) yields the lowest accuracy on both datasets (82.8 / 21.5), and

the smallest font size (6) is likewise the worst setting along its axis (82.6 / 21.9), confirming that insufficient readability impairs the visual encoding of previous reasoning and thereby degrades reasoning accuracy. *Third, the compression mechanism is preserved across all settings*, as the generated token counts remain stable (around 4K on MATH500 and 12–14K on AIME25) regardless of the rendering configuration, so varying the configuration does not break the vision-text compression of VTC-R1.

B.4 Information Preservation of Rendering

To examine what information survives the render-to-encode step, we render 30 real reasoning traces under different DPI and font settings, ask the model to transcribe them, and measure token recall, i.e., the fraction of original tokens (as a multiset) recovered in the transcription, reported per token type (digit, math-symbol, and word) in Table 11. At the default setting, the round-trip preserves the reasoning-critical tokens (90.2% digits, 85.6% math-symbols, and 81.1% words), retaining a large fraction of the information even under this strict token-level measure and supporting the claim that rendered images faithfully carry reasoning content. The lost information is mostly words, which degrade first (down to 66.7% at DPI 48) along with math-symbols (73.0% at DPI 48), whereas digits, the token type most decisive for arithmetic correctness, remain consistently well-preserved ($\geq 83\%$) across all settings. These recall trends further mirror the accuracy trends in Sec. B.3, where a lower DPI or a smaller font yields both lower recall and lower accuracy, thereby closing the loop from information retention to task performance.

C Details about Experiments

C.1 Implementation Details.

Supervised fine-tuning is conducted using the LlamaFactory library (Zheng et al., 2024). For all methods and across all model architectures, a learning rate of 1×10^{-5} is used, with a warmup ratio of 0.1 and a cosine learning rate schedule. Training is performed for one epoch with a batch size of 64, the maximum sequence length is increased to 32,768 tokens. All models are trained using 8 NVIDIA H20 GPUs with 96 GB of memory.

We adopt the official implementation of TokenSkip (Xia et al., 2025), which supports compression ratios ranging from 0.6 to 0.9. We observe that training becomes unstable and collapses when the

Factor	Specification / Sampling Strategy
dpi	Mixture of sets: <i>lowest</i> (45–59), <i>low</i> (60–71), <i>medium</i> (72–119), <i>normal</i> ({72, 80, 96, 100, 110, 120, 144, 150, 300}), <i>high</i> (over 300); favor normal/medium with small probability spikes to extremes.
page_size	(i) Fixed paper sizes (A4, Letter, Legal, A5, B5, A3, B4, Tabloid) with priors; (ii) common aspect ratios (e.g., 1.414, 1.333, 1.5, 1.778); (iii) fully random aspect via piecewise distribution (narrow → tall).
font_family	Pooled and deduplicated families across serif/sans/mono/pixel; italics sampled by filename heuristics (suffixes, italic/oblique).
font_size	{7, 7.5, 8, 9, 9.5, 10, 11, 12, 14}; line_height tied as font_size + {0, ..., 3}.
alignment	LEFT/JUSTIFY (dominant) with small-prob. RIGHT/CENTER.
margins	Three patterns: all-equal; vertical-larger; horizontal-larger; values in 10–40 pt ranges.
indent	Modes: none; first-line indent ($\approx 1\text{--}2.5$ em); block/hanging with left/right indents.
spacing	space-before/space-after use a multi-mode prior (none, small, large).
h_scale	Horizontal glyph scaling (0.75–1.0) with decaying probabilities.
colors	Page/background/font palettes for light/dark themes; document/web/code styles inherit coherent triplets (page, paragraph, font).
borders	Optional paragraph borders with width/padding; disabled by default.
newline_markup	With small probability, explicit markers (e.g., <code>\n</code> , tags, or tokens) inserted to preserve structure.
auto_crop	Optional white-margin cropping and last-page trimming.

Table 9: Rendering configuration factors in the rendering pipeline and their sampling strategies.

Config	MATH500			AIME25		
	ACC $\uparrow$	TOK	LAT $\downarrow$	ACC $\uparrow$	TOK	LAT $\downarrow$
(a) DPI (font=9, width=595)						
48	82.8	4.37	2.26	21.5	14.4	6.37
60	84.4	4.28	2.39	23.5	13.5	6.50
72*	86.0	4.12	2.19	26.3	13.0	6.81
96	86.2	4.10	2.73	28.1	12.1	8.57
120	84.8	4.59	3.47	27.9	12.1	11.69
144	85.2	4.46	4.04	27.1	12.5	15.84
(b) Font size (DPI=72, width=595)						
6	82.6	4.46	2.47	21.9	14.5	6.76
7	84.0	4.37	2.36	22.5	14.2	6.87
8	83.8	4.30	2.39	23.1	13.0	6.20
9*	86.0	4.12	2.19	26.3	13.0	6.81
10	85.2	4.08	2.46	23.8	12.6	7.06
12	84.6	4.28	2.60	25.8	12.3	7.88
(c) Page width (DPI=72, font=9)						
421	86.0	4.08	2.19	23.3	13.3	7.08
595*	86.0	4.12	2.19	26.3	13.0	6.81
842	83.4	4.25	2.51	23.1	12.9	7.24
1190	83.2	3.91	2.12	25.0	13.0	8.42

Table 10: **Sensitivity to inference-time rendering hyperparameters.** Bold indicates the best value along each axis. The default configuration is marked with *.

ratio is set to 0.6; therefore, we use a compression ratio of 0.8 in our experiments.

All evaluation experiments are conducted on a single NVIDIA H20 GPU with 96 GB of memory. Inference is performed using the vLLM framework, with a temperature of 0.6 and a top- p value of 0.95. For standard SFT, the maximum number of generated tokens (max_new_tokens) is set to 32,768. For VTC-R1, the maximum number of generated tokens per iteration is set to 8,192, and the maxi-

Config	Digit	Math-symbol	Word
(a) DPI (font=9)			
48	83.1	73.0	66.7
60	93.7	78.6	81.5
72*	90.2	85.6	81.1
96	88.8	84.0	82.7
120	95.4	92.5	91.7
144	94.8	94.1	92.0
(b) Font size (DPI=72)			
6	91.8	78.5	76.8
9*	93.8	86.1	82.4
12	94.0	90.2	86.6

Table 11: **Token recall through the render-to-encode round-trip.** The default configuration is marked with *.

imum number of iterations is set to 8.

C.2 Benchmark

GSM8K: A widely used benchmark for multi-step reasoning, consisting of 8,500 grade school math word problems, with a canonical test set of 1,319 problems.

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

AIME25: A benchmark dataset consisting of 30 challenging mathematical problems from the 2025 American Invitational Mathematics Examination.

AMC23: A challenging evaluation set comprising 50 problems from the 2023 American Mathematics Competitions, serving as a benchmark for competition-level mathematical reasoning.

GPQA-Diamond: A high-quality subset of the GPQA benchmark, with 198 complex graduate-

Parameter	Value
page-size	595 × 842
dpi	72
margin-x	10
margin-y	10
font-path	DejaVuSans.ttf
font-size	9
line-height	10
font-color	#000000
alignment	LEFT
horizontal-scale	1.0
first-line-indent	0
left-indent	0
right-indent	0
space-before	0
space-after	0
border-width	0
border-padding	0
page-bg-color	#FFFFFF
para-bg-color	#FFFFFF
auto-crop-width	true
auto-crop-last-page	true

Figure 5: Default rendering configuration used in our experiments.

level multiple-choice questions across various scientific domains. It serves as the out-of-distribution benchmark in our evaluation.

C.3 Training Dataset

We use OpenR1-Math-Inf (Yan et al., 2025), which is a subset of OpenR1-Math-220K (Hugging Face, 2025). 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 Numina-Math 1.5. All traces have been verified using Math Verify.

We first perform data cleaning on the OpenR1-Math-Inf dataset, resulting in 60,688 valid instances. In OpenR1-Math-Inf, for each instance, the original reasoning trace is partitioned into multiple segments based on a hyperparameter η , which controls the maximum token length of each segment. Following the data construction procedure of VTC-R1, this process yields a total of 106K training instances and approximately 105K rendered images.

<think>Okay, so I need to figure out the order of the real numbers a , b , x_1 , and x_2 where $a < b$ and $x_1 < x_2$. The function given is $y = (x - a)(x - b) - 1/2$, and the equation we're looking at is $(x - a)(x - b) - 1/2 = 0$. The answer choices are (A) through (D), each proposing a different order. Let me think through this step by step.

First, I recall that the quadratic function $y = (x - a)(x - b)$ is a parabola that opens upwards because the coefficient of x^2 is positive (since when you expand $(x - a)(x - b)$, the x^2 term is $1 \times x^2$). The roots of this quadratic (without the $-1/2$) are at $x = a$ and $x = b$. Since $a < b$, the parabola crosses the x -axis at a and b , and since it opens upwards, the vertex is between a and b , and it's the minimum point.

Now, the given equation is $(x - a)(x - b) - 1/2 = 0$. So, this is equivalent to shifting the original parabola down by $1/2$ units. The roots of this new equation, x_1 and x_2 , are the points where the shifted parabola crosses the x -axis.

Since we're subtracting $1/2$ from the original function, the entire graph is shifted downward. The original parabola had its vertex at the midpoint between a and b . Let's calculate the vertex coordinates. The vertex occurs at $x = (a + b)/2$. The y -coordinate of the vertex for the original parabola is found by plugging $x = (a + b)/2$ into the original function: $y = [(a + b)/2 - a][(a + b)/2 - b] - 1/2$.

Wait, no, the original function before shifting is $(x - a)(x - b)$. So the vertex y -value is at $x = (a + b)/2$: $y = [(a + b)/2 - a][(a + b)/2 - b] = [(b - a)/2][-(b - a)/2] = -(b - a)^2/4$.

So the vertex of the original parabola is at $((a + b)/2, -(b - a)^2/4)$. Then, when we subtract $1/2$, the new vertex y -coordinate becomes $-(b - a)^2/4 - 1/2$. Since this is a downward shift, the entire parabola is lower, so the minimum point is even lower. Therefore, the shifted parabola will cross the x -axis at two points x_1 and x_2 , which are on either side of the original roots a and b .

Wait, but the original parabola had roots at a and b . When we shift it down by $1/2$, the roots will move. Since the parabola opens upwards, shifting it down would move the roots outward. That is, the new roots x_1 and x_2 should be outside the interval $[a, b]$. But let me check this.

Alternatively, maybe I should sketch a graph. Let me visualize the original parabola crossing the x -axis at a and b . The vertex is below the x -axis. If we shift it down by $1/2$, the vertex is even lower, so the parabola will now cross the x -axis further out. Therefore, the new roots x_1 and x_2 should be outside the original interval $[a, b]$. So x_1 would be less than a , and x_2 would be greater than b . Therefore, the order would be $x_1 < a < b < x_2$, which is option (D). Hmm, but let me confirm this.

Wait, but maybe my initial reasoning is off. Let's think again. Suppose we have a quadratic equation $(x - a)(x - b) = 1/2$. So we can consider the original parabola and the line $y = 1/2$. The solutions to the equation are the points where the parabola intersects the line $y = 1/2$. However, in our case, the equation is $(x - a)(x - b) - 1/2 = 0$, which is equivalent to $(x - a)(x - b) = 1/2$. So we have the parabola $y = (x - a)(x - b)$ and the horizontal line $y = 1/2$. The points of intersection between the parabola and this line are the solutions x_1 and x_2 . Since the parabola opens upwards, and the vertex is below the x -axis, the parabola will intersect the line $y = 1/2$ at two points. Since $y = 1/2$ is above the x -axis, and the vertex is below the x -axis, the parabola must intersect the line $y = 1/2$ at two points: one to the left of a and one to the right of b . Because between a and b , the parabola is below the x -axis (since the vertex is the minimum and it's below the x -axis), so the parabola is below zero between a and b . Hence, when you set it equal to $1/2$, which is positive, the intersections must be outside the interval $[a, b]$. Therefore, $x_1 < a$ and $x_2 > b$. Since $x_1 < x_2$, this would mean $x_1 < a < b < x_2$, which is option (D).

But wait, let me confirm this with a concrete example. Let's take specific values for a and b . Suppose $a = 1$ and $b = 3$. Then the original quadratic is $(x - 1)(x - 3) = x^2 - 4x + 3$. The vertex is at $x = 2$, $y = (2 - 1)(2 - 3) = (1)(-1) = -1$. So the vertex is at $(2, -1)$. Then the equation $(x - 1)(x - 3) - 1/2 = 0$ is $x^2 - 4x + 3 - 1/2 = x^2 - 4x + 2.5 = 0$. Let's solve this quadratic equation. The discriminant is $16 - 10 = 6$. So the roots are $[4 \pm \sqrt{6}]/2 = 2 \pm \sqrt{6}/2$. $\sqrt{6} \approx 2.45$, so $\sqrt{6}/2 \approx 1.225$. Therefore, the roots are approximately $2 - 1.225 = 0.775$ and $2 + 1.225 = 3.225$. So $x_1 = 0.775$ and $x_2 = 3.225$. Comparing to $a = 1$ and $b = 3$, so $x_1 = 0.775 < a = 1$, and $x_2 = 3.225 > b = 3$. Hence, in this case, $x_1 < a < b < x_2$, which matches option (D).

Therefore, this suggests that option (D) is the correct answer. But let me try another example to ensure. Let me take $a = 0$ and $b = 2$. Then the original quadratic is $(x - 0)(x - 2) = x^2 - 2x$. The vertex is at $x = 1$, $y = (1 - 0)(1 - 2) = -1$. The equation becomes $x^2 - 2x - 1/2 = 0$. Solving, discriminant $= 4 + 2 = 6$, roots $[2 \pm \sqrt{6}]/2 = 1 \pm \sqrt{6}/2$. Again, $\sqrt{6} \approx 2.45$, so roots at $1 - 1.225 = -0.225$ and $1 + 1.225 = 2.225$. So $x_1 = -0.225 < a = 0$, and $x_2 = 2.225 > b = 2$. So again, $x_1 < a < b < x_2$. This reinforces that option (D) is correct.

Alternatively, maybe using calculus. The function $f(x) = (x - a)(x - b) - 1/2$. Let's find where it crosses the x -axis. The derivative $f'(x) = 2x - (a + b)$. Setting derivative to zero gives $x = (a + b)/2$, which is the vertex. The function at the vertex is $f((a + b)/2) = ((a + b)/2 - a)((a + b)/2 - b) - 1/2 = ((b - a)/2)(-(b - a)/2) - 1/2 = -(b - a)^2/4 - 1/2$. Since this is negative (because $-(b - a)^2/4$ is negative and divided by 4, then multiplied by -1, so negative, and then subtracting $1/2$ makes it even more negative). Therefore, the vertex is below the x -axis, and the parabola opens upwards, so the graph must cross the x -axis twice, once to the left of a and once to the right of b . Hence, $x_1 < a$ and $x_2 > b$, so $x_1 < a < b < x_2$. Therefore, the answer is (D).

But let me check if there's any case where the roots could be between a and b . Suppose if instead of subtracting $1/2$, we added $1/2$, then the parabola would be shifted upwards, and if the vertex was originally below the x -axis, shifting it up might bring the roots closer or even between a and b . But in this problem, we are subtracting $1/2$, which lowers the parabola further. Therefore, the roots must be outside the interval $[a, b]$. Hence, the correct order is $x_1 < a < b < x_2$. So option (D).

Wait, but why is this the case? Let's think about the graph again. The original parabola crosses the x -axis at a and b . Between a and b , it's below the x -axis (since it's a downward opening parabola). If we shift it down by $1/2$, the entire graph moves down, so the part between a and b is even lower, but the arms of the parabola go to infinity. So the points where the shifted parabola crosses the x -axis must be outside the original roots because between a and b , it's even more negative after shifting down. Hence, to cross the x -axis, it has to be outside a and b . Thus, x_1 is left of a and x_2 is right of b . Therefore, the order is $x_1 < a < b < x_2$, which is option (D).

Therefore, after all this reasoning, the correct answer should be (D).

Final Answer (D) </think> <answer>D</answer>

Figure 6: Example rendered page.

For the final answer A , the special token sequence `<answer> A </answer>` is used to facilitate answer extraction and to explicitly indicate the termination of the reasoning process. For instances consisting of more than one reasoning step, when step > 1 , the intermediate supervision is formatted as `<think>Got it, let's continue. {step_text}</think>`.

C.4 Batch Inference

The detailed batch inference procedure of VTC-R1 is given in Algorithm 2.

C.5 Comparison with Additional Baselines

Comparison with RoT. We further compare VTC-R1 with Render of Thought (RoT) (Wang et al., 2026), a representative latent-CoT method, under the Qwen3-VL-4B backbone. As shown in Table 12, VTC-R1 substantially outperforms RoT, improving accuracy by 55.7% on GSM8K and 52.6% on MATH500, which indicates that the latent reasoning of RoT comes at a considerable cost in reasoning performance. Although RoT uses far fewer tokens, this reflects the different objectives of the two approaches: latent-CoT methods such as

Algorithm 2 VTC-R1 Batch Inference

Input: batch of questions $\mathcal{Q} = \{Q_1, \dots, Q_B\}$; initial images $\{\mathcal{I}_1^{\text{init}}, \dots, \mathcal{I}_B^{\text{init}}\}$; vision language model M ; system prompt $\langle S_v \rangle$; rendering operator R_θ ; maximum iteration T

Initialize:

Active request set $\mathcal{S} \leftarrow \{1, \dots, B\}$

Current image sets $\mathcal{I}_k \leftarrow \mathcal{I}_k^{\text{init}}$ for all $k \in \{1, \dots, B\}$

Final answers $\mathcal{A} \leftarrow \{\emptyset\}_{k=1}^B$

for $t = 1$ to T **do**

if $\mathcal{S} = \emptyset$ **then**

break

end if

 Batch Generation via vLLM:

 Construct batch prompts $\mathcal{P} \leftarrow \{(\langle S_v \rangle, Q_k, \mathcal{I}_k) \mid k \in \mathcal{S}\}$

 Obtain batch outputs: $\{O_k\}_{k \in \mathcal{S}} \leftarrow M(\mathcal{P})$

 Update States and Render:

for each $k \in \mathcal{S}$ **do**

if O_k produces the final answer **then**

$A_k \leftarrow \text{ExtractAnswer}(O_k)$

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{k\}$ {Remove finished request from active set}

else

 Extract reasoning progress LR_k from O_k

 Render reasoning into images: $I_{\text{new}} \leftarrow R_\theta(LR_k)$

 Update image history: $\mathcal{I}_k \leftarrow \mathcal{I}_k \cup \{I_{\text{new}}\}$

end if

end for

end for

if $\mathcal{S} \neq \emptyset$ **then**

 Handle Time-out Requests:

for each $k \in \mathcal{S}$ **do**

$A_k \leftarrow \text{ExtractAnswer}(O_k)$

end for

end if

Output: set of final answers $\mathcal{A} = \{A_1, \dots, A_B\}$

Method (Qwen3-VL-4B)	GSM8K		MATH500	
	ACC $\uparrow$	TOK	ACC $\uparrow$	TOK
RoT (Wang et al., 2026)	37.8	32	33.2	64
VTC-R1	93.5	1362	85.8	4189

Table 12: Comparison with the latent-CoT method RoT. ACC is higher-is-better ($\uparrow$); TOK denotes the average number of generated tokens.

RoT primarily emphasize latent reasoning, whereas VTC-R1 targets long-context reasoning with strong task performance and is systematically evaluated on challenging mathematical benchmarks.

Comparison with InftyThink. We also compare VTC-R1 with InftyThink under the same backbone and evaluation settings. As reported in Table 13, VTC-R1 achieves consistently higher accuracy, with gains of 1.0, 8.1, and 0.3 percentage

points on MATH500, GPQA-D, and AMC23, respectively, while using fewer tokens on all three benchmarks (reductions of approximately 32.1%, 15.0%, and 32.0%). InftyThink is slightly faster on MATH500 and GPQA-D, whereas VTC-R1 is faster on AMC23. More importantly, InftyThink relies on an external strong model to summarize previous reasoning, while VTC-R1 directly renders the model’s own reasoning as optical memory without any external summarizer, providing a stronger overall trade-off among reasoning accuracy, token efficiency, and model independence.

C.6 Case Study

We present four examples to qualitatively analyze the behavior of VTC-R1. The gray-shaded regions indicate reasoning steps that are performed by con-

Benchmark	ACC $\uparrow$		TOK		LAT $\downarrow$	
	VTC-R1	InftyThink	VTC-R1	InftyThink	VTC-R1	InftyThink
MATH500	86.0	85.0	4.12	6.07	2.19	2.12
GPQA-D	46.0	37.9	10.73	12.63	6.96	5.22
AMC23	64.4	64.1	8.81	12.95	4.30	5.70

Table 13: Comparison with InftyThink under the same backbone and evaluation settings. TOK is reported in thousands (K) and LAT in seconds (s).

ditioning on images rendered from previous reasoning segments. **Example 1** demonstrates further verification of a previously obtained solution. **Example 2** derives the final answer by summarizing completed prior reasoning. **Example 3** performs error correction and reflection based on contradictions identified in earlier reasoning, eventually reaching the correct answer. **Example 4** continues the reasoning process by building directly upon preceding reasoning steps. Collectively, these examples demonstrate that images rendered from previous reasoning segments can be effectively leveraged as *optical memory* to support multi-step reasoning.

<think>Okay, so I need to solve the equation $3^x(2x) + 19 = 10^x \cdot x$. Hmm, let's see. First, I should probably try to simplify or rewrite the equation in a way that makes it easier to handle. Let me write it down again: $3^x(2x) + 19 = 10^x \cdot x$.
Wait, $3^x(2x)$ is the same as $(3^x)^2$, right? So maybe I can let $y = 3^x$ or something like that. Let me try substituting variables here. Let's set $y = 3^x$. Then, $3^x(2x)$ would be $y^2 \cdot 2$. So substituting into the equation, we get $y^2 \cdot 2 + 19 = 10^x \cdot x$. But 10^x is equal to $(10^{\log_3 3})^x$. Hmm, but 10 and 3 are different bases, so maybe I need another substitution here. Alternatively, maybe I can set $x = 10^y$. Then, since $3^x(2x) = (3^{10^y})^2$, and x is equal to $(3^{1/10})^{10^y}$, but that might complicate things. Wait, if $x = 10^y$, then $x = \log_{10}(10^y)$. Then, $3^x(2x)$ would be $3^{10^y} \cdot 2 \log_{10}(10^y)$. Hmm, that seems a bit messy. Let me think again.
Alternatively, maybe I can express both sides with the same base! But 3 and 10 are different primes, so that's not straightforward. Maybe I can take logarithms of both sides! Let me see. If I take the natural logarithm of both sides, I get $\ln(3^x(2x) + 19) = \ln(10^x \cdot x)$. But that doesn't seem helpful because of the $3^x(2x)$ term. The logarithm of a sum is not the sum of logarithms, so that might not work.
Alternatively, maybe I can rearrange the equation. Let's subtract 10^x from both sides: $3^x(2x) - 10^x + 19 = 0$. Hmm, but this is a transcendental equation because it's a combination of exponential functions with different bases. These types of equations usually don't have solutions in terms of elementary functions, so maybe I need to use numerical methods or graphing to find an approximate solution. But before jumping to that, let me check if there are any integer solutions. Maybe x is an integer. Let's try $x = 1$: $3^1(2 \cdot 1) + 19 = 3 \cdot 2 + 19 = 6 + 19 = 25$. Not equal to $10^1 \cdot 1 = 10$. Not equal. $x = 2$: $3^2(2 \cdot 2) + 19 = 9 \cdot 4 + 19 = 36 + 19 = 55$. Not equal to $10^2 \cdot 2 = 200$. Not equal. $x = 3$: $3^3(2 \cdot 3) + 19 = 27 \cdot 6 + 19 = 162 + 19 = 181$. Not equal to $10^3 \cdot 3 = 3000$. Not equal. So $x = 2$ is a solution. Maybe there's another solution?
Wait, let's check $x = 0$: $3^0(2 \cdot 0) + 19 = 1 \cdot 0 + 19 = 19$. Not equal to $10^0 \cdot 0 = 1$. Not equal. $x = -1$: $3^{-1}(2 \cdot -1) + 19 = 0.5 \cdot -2 + 19 = -1 + 19 = 18$. Not equal to $10^{-1} \cdot -1 = -0.1$. Not equal. So $x = 2$ is a solution. Is there another solution?
Let me check $x = 1.5$: $3^{1.5}(2 \cdot 1.5) + 19 = 3 \cdot 1.5 \cdot 2 + 19 = 9 + 19 = 28$. Not equal to $10^{1.5} \cdot 1.5 = 31.62$. Not equal. $x = 2.5$: $3^{2.5}(2 \cdot 2.5) + 19 = 3 \cdot 2.5 \cdot 2 + 19 = 15 + 19 = 34$. Not equal to $10^{2.5} \cdot 2.5 = 262$. Not equal. So maybe $x = 2$ is the only integer solution. But could there be another solution?
Wait, let's consider the behavior of the functions. Let $f(x) = 3^x(2x) + 19$ and $g(x) = 10^x \cdot x$. We need to find x where $f(x) = g(x)$. Let's analyze the functions.
First, $f(x)$ is an exponential function with base $3^2 = 9$, so it grows very quickly. $g(x)$ is an exponential function with base 10, which also grows quickly but slower than $f(x)$ since $9 < 10$. Wait, no, 10 is bigger than 9, so actually 10^x grows faster than 9^x . Wait, $3^x(2x)$ is 9^x , right? So 9^x vs 10^x . Since $10 > 9$, 10^x grows faster than 9^x . Therefore, as x increases, $g(x)$ will outpace $f(x)$. But at $x = 0$, $f(0) = 19$ and $g(0) = 0$. So $f(x)$ starts higher. Then, as x increases, $g(x)$ starts to grow faster. Let's see where they cross.
We already saw that at $x = 2$, $f(2) = 55$ and $g(2) = 200$. Let's check $x = 1.5$: $f(1.5) = 28$ and $g(1.5) = 31.62$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 1.9$: $f(1.9) = 3^{1.9}(2 \cdot 1.9) + 19 = 46 \cdot 1.9 + 19 = 87.4 + 19 = 106.4$. $g(1.9) = 10^{1.9} \cdot 1.9 = 112.2 \cdot 1.9 = 213.18$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.1$: $f(2.1) = 3^{2.1}(2 \cdot 2.1) + 19 = 46 \cdot 2.1 + 19 = 96.6 + 19 = 115.6$. $g(2.1) = 10^{2.1} \cdot 2.1 = 125.89 \cdot 2.1 = 264.37$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.2$: $f(2.2) = 3^{2.2}(2 \cdot 2.2) + 19 = 46 \cdot 2.2 + 19 = 101.2 + 19 = 120.2$. $g(2.2) = 10^{2.2} \cdot 2.2 = 138.04 \cdot 2.2 = 303.69$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.3$: $f(2.3) = 3^{2.3}(2 \cdot 2.3) + 19 = 46 \cdot 2.3 + 19 = 105.8 + 19 = 124.8$. $g(2.3) = 10^{2.3} \cdot 2.3 = 154.48 \cdot 2.3 = 355.30$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.4$: $f(2.4) = 3^{2.4}(2 \cdot 2.4) + 19 = 46 \cdot 2.4 + 19 = 110.4 + 19 = 129.4$. $g(2.4) = 10^{2.4} \cdot 2.4 = 173.47 \cdot 2.4 = 416.33$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.5$: $f(2.5) = 3^{2.5}(2 \cdot 2.5) + 19 = 46 \cdot 2.5 + 19 = 115 + 19 = 134$. $g(2.5) = 10^{2.5} \cdot 2.5 = 199.53 \cdot 2.5 = 498.83$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.6$: $f(2.6) = 3^{2.6}(2 \cdot 2.6) + 19 = 46 \cdot 2.6 + 19 = 119.6 + 19 = 138.6$. $g(2.6) = 10^{2.6} \cdot 2.6 = 223.87 \cdot 2.6 = 582.06$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.7$: $f(2.7) = 3^{2.7}(2 \cdot 2.7) + 19 = 46 \cdot 2.7 + 19 = 124.2 + 19 = 143.2$. $g(2.7) = 10^{2.7} \cdot 2.7 = 251.19 \cdot 2.7 = 678.21$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.8$: $f(2.8) = 3^{2.8}(2 \cdot 2.8) + 19 = 46 \cdot 2.8 + 19 = 128.8 + 19 = 147.8$. $g(2.8) = 10^{2.8} \cdot 2.8 = 281.84 \cdot 2.8 = 789.15$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 2.9$: $f(2.9) = 3^{2.9}(2 \cdot 2.9) + 19 = 46 \cdot 2.9 + 19 = 133.4 + 19 = 152.4$. $g(2.9) = 10^{2.9} \cdot 2.9 = 316.23 \cdot 2.9 = 917.07$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 3$: $f(3) = 3^3(2 \cdot 3) + 19 = 162 + 19 = 181$. $g(3) = 10^3 \cdot 3 = 3000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 4$: $f(4) = 3^4(2 \cdot 4) + 19 = 324 + 19 = 343$. $g(4) = 10^4 \cdot 4 = 40000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 5$: $f(5) = 3^5(2 \cdot 5) + 19 = 6561 + 19 = 6580$. $g(5) = 10^5 \cdot 5 = 500000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 6$: $f(6) = 3^6(2 \cdot 6) + 19 = 12924 + 19 = 12943$. $g(6) = 10^6 \cdot 6 = 6000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 7$: $f(7) = 3^7(2 \cdot 7) + 19 = 25929 + 19 = 25948$. $g(7) = 10^7 \cdot 7 = 70000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 8$: $f(8) = 3^8(2 \cdot 8) + 19 = 53144 + 19 = 53163$. $g(8) = 10^8 \cdot 8 = 800000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 9$: $f(9) = 3^9(2 \cdot 9) + 19 = 111973 + 19 = 111992$. $g(9) = 10^9 \cdot 9 = 9000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 10$: $f(10) = 3^{10}(2 \cdot 10) + 19 = 239856 + 19 = 239875$. $g(10) = 10^{10} \cdot 10 = 100000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 11$: $f(11) = 3^{11}(2 \cdot 11) + 19 = 515904 + 19 = 515923$. $g(11) = 10^{11} \cdot 11 = 1100000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 12$: $f(12) = 3^{12}(2 \cdot 12) + 19 = 1141872 + 19 = 1141891$. $g(12) = 10^{12} \cdot 12 = 12000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 13$: $f(13) = 3^{13}(2 \cdot 13) + 19 = 2541888 + 19 = 2541907$. $g(13) = 10^{13} \cdot 13 = 130000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 14$: $f(14) = 3^{14}(2 \cdot 14) + 19 = 5662368 + 19 = 5662387$. $g(14) = 10^{14} \cdot 14 = 1400000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 15$: $f(15) = 3^{15}(2 \cdot 15) + 19 = 12706656 + 19 = 12706675$. $g(15) = 10^{15} \cdot 15 = 15000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 16$: $f(16) = 3^{16}(2 \cdot 16) + 19 = 28518656 + 19 = 28518675$. $g(16) = 10^{16} \cdot 16 = 160000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 17$: $f(17) = 3^{17}(2 \cdot 17) + 19 = 63491712 + 19 = 63491731$. $g(17) = 10^{17} \cdot 17 = 1700000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 18$: $f(18) = 3^{18}(2 \cdot 18) + 19 = 142978304 + 19 = 142978323$. $g(18) = 10^{18} \cdot 18 = 18000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 19$: $f(19) = 3^{19}(2 \cdot 19) + 19 = 322581248 + 19 = 322581267$. $g(19) = 10^{19} \cdot 19 = 190000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 20$: $f(20) = 3^{20}(2 \cdot 20) + 19 = 722575360 + 19 = 722575379$. $g(20) = 10^{20} \cdot 20 = 2000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 21$: $f(21) = 3^{21}(2 \cdot 21) + 19 = 1610706304 + 19 = 1610706323$. $g(21) = 10^{21} \cdot 21 = 21000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 22$: $f(22) = 3^{22}(2 \cdot 22) + 19 = 3583603776 + 19 = 3583603795$. $g(22) = 10^{22} \cdot 22 = 220000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 23$: $f(23) = 3^{23}(2 \cdot 23) + 19 = 8048128448 + 19 = 8048128467$. $g(23) = 10^{23} \cdot 23 = 2300000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 24$: $f(24) = 3^{24}(2 \cdot 24) + 19 = 18107063040 + 19 = 18107063059$. $g(24) = 10^{24} \cdot 24 = 24000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 25$: $f(25) = 3^{25}(2 \cdot 25) + 19 = 40481284480 + 19 = 40481284499$. $g(25) = 10^{25} \cdot 25 = 250000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 26$: $f(26) = 3^{26}(2 \cdot 26) + 19 = 90481284480 + 19 = 90481284499$. $g(26) = 10^{26} \cdot 26 = 2600000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 27$: $f(27) = 3^{27}(2 \cdot 27) + 19 = 200481284480 + 19 = 200481284499$. $g(27) = 10^{27} \cdot 27 = 27000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 28$: $f(28) = 3^{28}(2 \cdot 28) + 19 = 444812844800 + 19 = 444812844819$. $g(28) = 10^{28} \cdot 28 = 280000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 29$: $f(29) = 3^{29}(2 \cdot 29) + 19 = 984812844800 + 19 = 984812844819$. $g(29) = 10^{29} \cdot 29 = 2900000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 30$: $f(30) = 3^{30}(2 \cdot 30) + 19 = 2184812844800 + 19 = 2184812844819$. $g(30) = 10^{30} \cdot 30 = 30000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 31$: $f(31) = 3^{31}(2 \cdot 31) + 19 = 4848128448000 + 19 = 4848128448019$. $g(31) = 10^{31} \cdot 31 = 310000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 32$: $f(32) = 3^{32}(2 \cdot 32) + 19 = 10748128448000 + 19 = 10748128448019$. $g(32) = 10^{32} \cdot 32 = 3200000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 33$: $f(33) = 3^{33}(2 \cdot 33) + 19 = 23648128448000 + 19 = 23648128448019$. $g(33) = 10^{33} \cdot 33 = 33000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 34$: $f(34) = 3^{34}(2 \cdot 34) + 19 = 52448128448000 + 19 = 52448128448019$. $g(34) = 10^{34} \cdot 34 = 340000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 35$: $f(35) = 3^{35}(2 \cdot 35) + 19 = 116448128448000 + 19 = 116448128448019$. $g(35) = 10^{35} \cdot 35 = 3500000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 36$: $f(36) = 3^{36}(2 \cdot 36) + 19 = 256448128448000 + 19 = 256448128448019$. $g(36) = 10^{36} \cdot 36 = 36000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 37$: $f(37) = 3^{37}(2 \cdot 37) + 19 = 574448128448000 + 19 = 574448128448019$. $g(37) = 10^{37} \cdot 37 = 370000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 38$: $f(38) = 3^{38}(2 \cdot 38) + 19 = 1264481284480000 + 19 = 1264481284480019$. $g(38) = 10^{38} \cdot 38 = 3800000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 39$: $f(39) = 3^{39}(2 \cdot 39) + 19 = 2834481284480000 + 19 = 2834481284480019$. $g(39) = 10^{39} \cdot 39 = 39000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 40$: $f(40) = 3^{40}(2 \cdot 40) + 19 = 6244481284480000 + 19 = 6244481284480019$. $g(40) = 10^{40} \cdot 40 = 400000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 41$: $f(41) = 3^{41}(2 \cdot 41) + 19 = 13944812844800000 + 19 = 13944812844800019$. $g(41) = 10^{41} \cdot 41 = 4100000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 42$: $f(42) = 3^{42}(2 \cdot 42) + 19 = 30744812844800000 + 19 = 30744812844800019$. $g(42) = 10^{42} \cdot 42 = 42000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 43$: $f(43) = 3^{43}(2 \cdot 43) + 19 = 68144812844800000 + 19 = 68144812844800019$. $g(43) = 10^{43} \cdot 43 = 430000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 44$: $f(44) = 3^{44}(2 \cdot 44) + 19 = 150144812844800000 + 19 = 150144812844800019$. $g(44) = 10^{44} \cdot 44 = 4400000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 45$: $f(45) = 3^{45}(2 \cdot 45) + 19 = 332144812844800000 + 19 = 332144812844800019$. $g(45) = 10^{45} \cdot 45 = 45000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 46$: $f(46) = 3^{46}(2 \cdot 46) + 19 = 736144812844800000 + 19 = 736144812844800019$. $g(46) = 10^{46} \cdot 46 = 460000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 47$: $f(47) = 3^{47}(2 \cdot 47) + 19 = 1629144812844800000 + 19 = 1629144812844800019$. $g(47) = 10^{47} \cdot 47 = 4700000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 48$: $f(48) = 3^{48}(2 \cdot 48) + 19 = 3601448128448000000 + 19 = 3601448128448000019$. $g(48) = 10^{48} \cdot 48 = 48000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 49$: $f(49) = 3^{49}(2 \cdot 49) + 19 = 7961448128448000000 + 19 = 7961448128448000019$. $g(49) = 10^{49} \cdot 49 = 490000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 50$: $f(50) = 3^{50}(2 \cdot 50) + 19 = 17514481284480000000 + 19 = 17514481284480000019$. $g(50) = 10^{50} \cdot 50 = 5000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 51$: $f(51) = 3^{51}(2 \cdot 51) + 19 = 38614481284480000000 + 19 = 386144812844800000019$. $g(51) = 10^{51} \cdot 51 = 51000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 52$: $f(52) = 3^{52}(2 \cdot 52) + 19 = 85514481284480000000 + 19 = 8551448128448000000019$. $g(52) = 10^{52} \cdot 52 = 520000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 53$: $f(53) = 3^{53}(2 \cdot 53) + 19 = 188144812844800000000 + 19 = 188144812844800000000019$. $g(53) = 10^{53} \cdot 53 = 5300000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 54$: $f(54) = 3^{54}(2 \cdot 54) + 19 = 414144812844800000000 + 19 = 414144812844800000000019$. $g(54) = 10^{54} \cdot 54 = 54000000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 55$: $f(55) = 3^{55}(2 \cdot 55) + 19 = 911144812844800000000 + 19 = 911144812844800000000019$. $g(55) = 10^{55} \cdot 55 = 550000000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 56$: $f(56) = 3^{56}(2 \cdot 56) + 19 = 2001448128448000000000 + 19 = 2001448128448000000000019$. $g(56) = 10^{56} \cdot 56 = 5600000000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 57$: $f(57) = 3^{57}(2 \cdot 57) + 19 = 4401448128448000000000 + 19 = 4401448128448000000000019$. $g(57) = 10^{57} \cdot 57 = 57000000000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 58$: $f(58) = 3^{58}(2 \cdot 58) + 19 = 9681448128448000000000 + 19 = 9681448128448000000000019$. $g(58) = 10^{58} \cdot 58 = 580000000000000000000000000000000000000000000000000000000000$. So $f(x)$ is still higher here. At $x = 2$, they cross. Let's check $x = 59$: $f(59) = 3^{59}(2 \cdot 59) + 19 = 21214481284480000000000 + 19 =$

[illegible]

Figure 9: Example 3.

[illegible]

Figure 10: Example 4.