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

Anonymous ACL submission

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× 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× speedup in end-to-end latency, highlighting its potential as a scalable solution for reasoning-intensive applications.

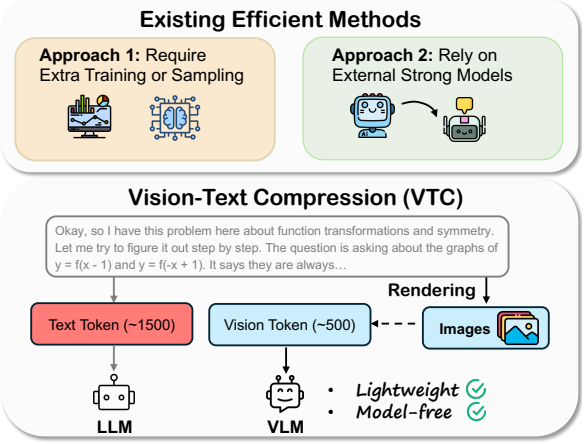

Figure 1: **Comparison between existing efficient reasoning approaches and vision-text compression (VTC).** Existing methods either require additional training or sampling procedures, or rely on external strong models. VTC leverages lightweight rendering to transform long textual reasoning traces into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

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 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 InftyThink (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 underexplored 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×–10× 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× 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× 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× 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× 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),

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

which controls the typography, layout, and visual style of rendered pages. The details of rendering configuration are provided in Appendix C.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:

$$\rho = \frac{L_t}{L_v}, \tag{2}$$

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

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

where $\langle S_s \rangle$ denotes the standard system prompt, such as "You are a helpful assistant." The tokens $|\texttt{U}|$ and $|\texttt{A}|$ indicate the start of user input and model response, respectively. The special tokens $\langle \texttt{think} \rangle$ and $\langle /\texttt{think} \rangle$ 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:

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

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

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

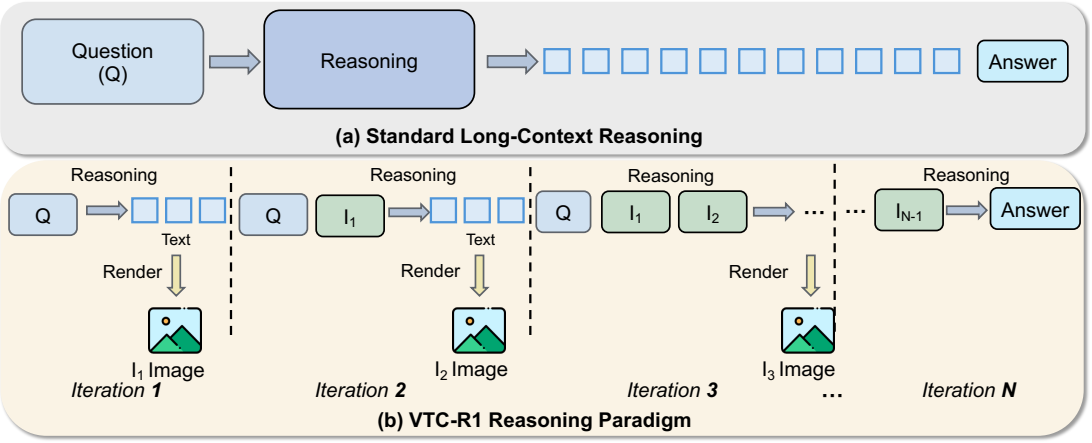

Figure 2: Comparison between standard long-context reasoning and the proposed VTC-R1 reasoning paradigm. **(a)** Standard long-context reasoning processes the entire reasoning trace as a single long sequence, leading to increasing computational and memory costs as the context grows. **(b)** VTC-R1 reformulates 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 $LR_1, \ldots, LR_n$ sequentially with the same conditionals. Consequently, for any answer extraction function $A = M(LR)$, one-pass and iterative generation yield the same answer distribution:

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

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

$$\langle \mathtt{S}_v \rangle \,|\, \mathtt{U} \,|\, Q \,|\, \mathtt{A} \,|\, \langle \mathtt{think} \rangle \, LR_1 \, \langle /\mathtt{think} \rangle,$$

where $\langle \mathtt{S}_v \rangle$ denotes the VTC-R1 system prompt.

> **VTC-R1 System Prompt** $\langle \mathtt{S}_v \rangle$
> 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 $\{I_1, \ldots, I_{i-1}\}$ is available. The reasoning process at the i-th iteration is then expressed as

$$\langle \mathtt{S}_v \rangle \,\big|\, \mathtt{U} \,\big|\, Q, I_1, \ldots, I_{i-1} \,\big|\, \mathtt{A} \,\big|\, \langle \mathtt{think} \rangle \, LR_i \, \langle /\mathtt{think} \rangle.$$

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

$$\langle \mathtt{S}_v \rangle \mid \mathtt{U} \mid \{Q, I_1, \ldots, I_{n-1}\} \mid \mathtt{A} \mid \langle \mathtt{think} \rangle LR_n \langle /\mathtt{think} \rangle A.$$

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 $LR_1, \ldots, LR_{i-1}$ are rendered into images $I_1, \ldots, I_{i-1}$. 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 $\langle \mathsf{S}_v \rangle$; 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:

$$O_i \leftarrow M(\langle \mathsf{S}_v \rangle, Q, \mathcal{I})$$

  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

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-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 $\{LR_1, LR_2, \ldots, LR_n\}$ 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

$$D_i = \begin{cases} \big(\langle \mathsf{S}_v \rangle, Q, \varnothing, LR_1\big), & i = 1, \\ \big(\langle \mathsf{S}_v \rangle, Q, \{I_j\}_{j<i}, LR_i\big), & 1 < i < n, \\ \big(\langle \mathsf{S}_v \rangle, Q, \{I_j\}_{j<i}, LR_n, A\big), & i = n. \end{cases} \tag{5}$$

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**×. 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., 2025b), and Glyph (Cheng et al., 2025).

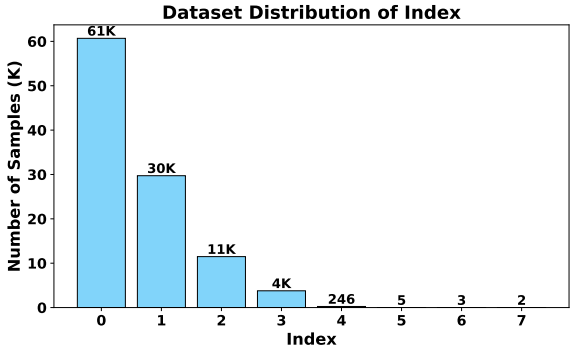

Figure 3: **Distribution of data index.** The index indicates the order of a reasoning segment for a given problem, where index 0 corresponds to the first segment. Most samples terminate at early steps, while a small fraction requires more than four iterations.

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.

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

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.

Model	GSM8K			MATH500			AIME25 (Avg@16)			AMC23 (Avg@16)		
	ACC ↑	TOK	LAT↓	ACC ↑	TOK	LAT↓	ACC ↑	TOK	LAT↓	ACC ↑	TOK	LAT↓
Qwen3-VL-8B												
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**$^{(6.6\times)}$	**90.0**$^{(+4.6)}$	3.39	**2.49**$^{(2.2\times)}$	30.00$^{(-2.71)}$	14.32	**12.02**$^{(2.5\times)}$	**77.97**$^{(+2.97)}$	8.18	**6.45**$^{(1.7\times)}$
Glyph												
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**$^{(2.7\times)}$	**86.0**$^{(+5.6)}$	4.12	**2.19**$^{(1.4\times)}$	**26.25**$^{(+0.63)}$	12.95	**6.81**$^{(1.7\times)}$	**64.38**$^{(+3.44)}$	8.81	**4.30**$^{(1.6\times)}$

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 D.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., 2025b) (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

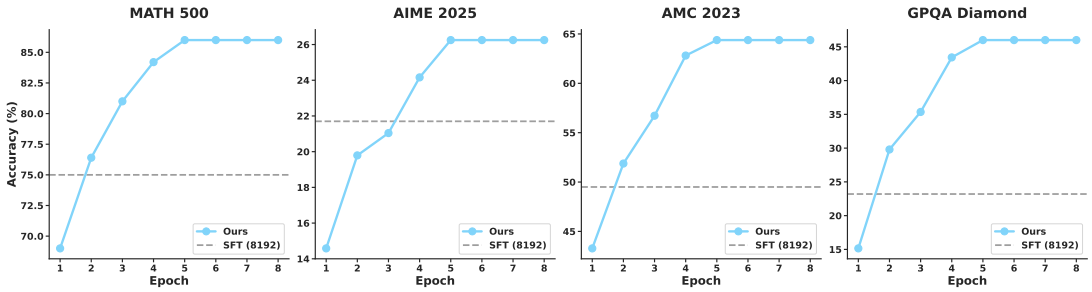

Figure 4: Accuracy of the proposed method across benchmarks under different maximum iteration epochs. The epoch index denotes the maximum number of allowed reasoning iterations. The dashed line indicates the single-round baseline (standard long-context reasoning for 8192 maximum tokens).

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

Model	GPQA-Diamond		
	ACC ↑	TOK	LAT↓
Qwen3-VL-8B			
SFT	37.4	14.78	26.88
VTC-R1	**48.5**$^{(+11.1)}$	9.77	**9.57**$^{(2.8\times)}$
Glyph			
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**$^{(1.2\times)}$

benchmark. Specifically, VTC-R1 yields accuracy 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 benchmarks 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 seg-

Segment Length	GSM8K		MATH500		AIME25		AMC23	
	ACC ↑	LAT ↓	ACC ↑	LAT ↓	ACC ↑	LAT ↓	ACC ↑	LAT ↓
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.

mentation length of 4K achieves the best or highly 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.

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

5 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× token compression and 2.7× 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.

Our rendering hyperparameters are fixed, including font size and image resolution. While we directly adopt the default configuration of Glyph, we have not explored the sensitivity of the method to different rendering parameters. Suboptimal rendering settings may affect visual readability and thereby degrade reasoning performance.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang 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, Weihan 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, Xinzhong 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*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025a. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, and 69 others. 2025b. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *Preprint*, arXiv:2507.01006.

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, Jiaxing 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. Inftythink: 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, Zheyan 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 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., 2025a) 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., 2025a), 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×–10× 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.

C Image Rendering

C.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 7. 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.

Factor	Specification / Sampling Strategy
dpi	Mixture of sets: *lowest* (45–59), *low* (60–71), *medium* (72–119), *normal* ({72, 80, 96, 100, 110, 120, 144, 150, 300}), *high* (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 + $\{0, \ldots, 3\}$.
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 (≈1–2.5 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., \n, tags, or tokens) inserted to preserve structure.
auto_crop	Optional white-margin cropping and last-page trimming.

Table 7: Rendering configuration factors in the rendering pipeline and their sampling strategies.

C.2 Rendering Example

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

D Details about Experiments

D.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 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 maximum number of iterations is set to 8.

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

D.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.

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.

Figure 6: Example rendered page.

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.

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>`.

D.4 Batch Inference

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

D.5 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 conditioning 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.

Algorithm 2 VTC-R1 Batch Inference

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

Initialize:

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

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

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 $\mathcal{P} \leftarrow \{(\langle \mathtt{S}_v \rangle, Q_k, \mathcal{I}_k) \mid k \in \mathcal{S}\}$

  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**

      $A_k \leftarrow$ 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**

    $A_k \leftarrow$ ExtractAnswer(O_k)

  end for

end if

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

<think>Okay, so I need to solve the equation 3^{2x} + 19 = 10^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^{2x} + 19 = 10^x.
Wait, 3^{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^{2x} would be y^2. So substituting into the equation, we get y^2 + 19 = 10^x. But 10^x is equal to (10^x). Hmm, but 10 and 3 are different bases, so maybe I need another substitution here.
Alternatively, maybe I can set t = 10^x. Then, since 3^{2x} is (3^x)^2, and 3^x is equal to (3^{1/x})^x, but that might complicate things. Wait, if t = 10^x, then x = log10(t). Then, 3^{2x} would be 3^{2 log10(t)}. 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^{2x} + 19) = ln(10^x). But that doesn't seem helpful because of the 3^{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^{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^{2*1} + 19 = 9 + 19 = 28. 10^1 = 10. Not equal. x = 2: 3^4 + 19 = 81 + 19 = 100. 10^2 = 100. Oh! That works! So x = 2 is a solution. Let me check x = 3: 3^6 + 19 = 729 + 19 = 748. 10^3 = 1000. Not equal. So x = 2 is a solution. Maybe there's another solution?
Wait, let's check x = 0: 3^0 + 19 = 1 + 19 = 20. 10^0 = 1. Not equal. x = -1: 3^{-2} + 19 = 1/9 + 19 = 19.111. 10^{-1} = 0.1. Not equal. So x = 2 is a solution. Is there another solution?
Let me check x = 3.5: 3^{3} + 19 = 27 + 19 = 46. 10^{3.5} ≈ 31.62. Not equal. x = 2.5: 3^{5} + 19 = 243 + 19 = 262. 10^{2.5} ≈ 316.23. 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^{2x} + 19 and g(x) = 10^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^{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) = 1 + 19 = 20, g(0) = 1. 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) = 100, g(2) = 100. Let's check x = 1.5: f(1.5) = 9^{1.5} + 19 = (9^{1})* (9^{0.5}) + 19 = 9 * 3 + 19 = 27 + 19 = 46. g(1.5) = 10^{1.5} ≈ 31.62. So f(x) is still higher here. At x = 2, they cross. Let's check x = 3: f(3) = 9^3 + 19 = 729 + 19 = 748. g(3) = 1000. So here, g(x) is higher. So between x = 2 and x = 3, g(x) overtakes f(x). So maybe there's another crossing point?
Wait, but at x = 2, they are equal. Let's check x = 1. Let's see f(1) = 9 + 19 = 28, g(1) = 10. So f(x) is still higher. At x = 2, equal. Then, at x = 3, g(x) is higher. So perhaps there's another crossing point between x = 2 and x = 3? Wait, but at x = 2, they are equal. Then, for x > 2, g(x) is higher. But maybe there's another crossing before x = 2? Let's check x = 1. Let's see f(1) = 28, g(1) = 10. So f(x) is higher. At x = 2, equal. So maybe there's only one crossing at x = 2? Wait, but when x approaches negative infinity, f(x) approaches 19 (since 3^{2x} approaches 0) and g(x) approaches 0. So f(x) is always above g(x) for x approaching negative infinity. Then, as x increases, f(x) grows faster than g(x) until x = 2, where they meet, and then g(x) overtakes f(x). Wait, but if f(x) is 9^x and g(x) is 10^x, then since 9 < 10, 10^x grows faster. So after x = 2, g(x) will always be higher. But maybe before x = 2, f(x) is higher. So the equation f(x) = g(x) has exactly one solution at x = 2? But wait, let's check x = 1. Let's see, f(1) = 28, g(1) = 10. So f(x) is higher. At x = 2, equal. Then, for x > 2, g(x) is higher. So maybe x = 2 is the only solution? Wait, but is there a possibility of another solution?
Wait, let's check x = 0.5: f(0.5) = 9^{0.5} + 19 = 3 + 19 = 22. g(0.5) = 10^{0.5} ≈ 3.16. So f(x) is still higher. At x = 1, f(x) is 28, g(x) is 10. So f(x) is still higher. At x = 2, equal. Then, x = 3, g(x) is higher. So maybe there's only one solution at x = 2. But wait, let's check x = 2. Let me confirm again: 3^{2*2} + 19 = 81 + 19 = 100, and 10^2 = 100. So yes, x = 2 is a solution.
But could there be another solution? Let's see. Let me check x = 2. Let's see, if I plug in x = 2, both sides are 100. Let me try x = 2. Let me see if there's another crossing. For example, suppose x = 1. Let me compute f(1.5) = 9^{1.5} + 19 = 27 + 19 = 46. g(1.5) ≈ 31.62. So f(x) is still higher. At x = 2, equal. Then, x = 2.5: f(2.5) = 9^{2.5} + 19 = 9^2 * 9^{0.5} + 19 = 81 * 3 + 19 = 243 + 19 = 262. g(2.5) = 10^{2.5} ≈ 316.23. So here, g(x) is higher. So between x = 2 and x = 2.5, g(x) overtakes f(x). But since at x = 2, they are equal, and then g(x) increases faster, so maybe there's another crossing point?
Wait, but how?
Wait, actually, since f(x) is 9^x and g(x) is 10^x, and 10 > 9, then 10^x grows faster than 9^x for all x > 0. Therefore, once g(x) overtakes f(x) at x = 2, it will stay above. So the only solution is x = 2. Wait, but let's check x = 0.5: f(x) is 22, g(x) is ~3.16. So f(x) is higher. At x = 1: f(x) = 28, g(x) = 10. Still f(x) higher. At x = 2: equal. So maybe x = 2 is the only solution. But let me check x = 2. Let's see, if we have another solution, maybe for x < 2? Let's try x = 1. Let's see, f(1) = 28, g(1) = 10. So f(x) is higher. x = 1. Let's check x = 1. Let's see, maybe x = 0. Let's check x = 0. f(0) = 1 + 19 = 20, g(0) = 1. So f(x) is higher. So maybe the only solution is x = 2. But wait, let's check x = 2. Let me confirm once again. 3^{2*2} = 81, 81 + 19 = 100, 10^2 = 100. Yes, that works. So x = 2 is a solution. But could there be another solution?
Wait, let's suppose that there's another solution. Let's set up the equation again: 3^{2x} + 19 = 10^x. Let's rearrange it as 3^{2x} - 10^x + 19 = 0. Let me denote u = 10^x. Then, 3^{2x} = (3^x)^2. But since u = 10^x, maybe we can relate 3^x to u. Let's see, 3^x = (3/10)^x * 10^x = (3/10)^x * u. So (3/10)^x = u / 10^x. Hmm, this seems complicated.
Alternatively, maybe express 3^{2x} as (3^x)^2. Let's set y = 3^x. Then, the equation becomes y^2 + 19 = 10^x. But 10^x can be written in terms of y. Since y = 3^x, then x = log_3(y). Therefore, 10^x = 10^{log_3(y)} = y^{log_3(10)}. Because 10^{log_3(y)} = e^{ln(10) * ln(y)} = y^{ln(10)/ln(3)} = y^{log_3(10)}. So substituting back into the equation, we get y^2 + 19 = y^{log_3(10)}. Hmm, this seems even more complicated. Maybe this substitution isn't helpful.
Alternatively, let's consider taking logarithms on both sides. Wait, but the equation is 3^{2x} + 19 = 10^x. If I take the natural logarithm of both sides, it becomes ln(3^{2x} + 19) = ln(10^x) = x ln(10). But this still leaves the logarithm of a sum, which isn't helpful.
Alternatively, maybe I can write 3^{2x} as (3^x)^2 and let z = 3^x. Then, the equation becomes z^2 + 19 = 10^x. But 10^x can be written in terms of z. Since z = 3^x, then x = log_3(z). Therefore, 10^x = 10^{log_3(z)} = z^{log_3(10)}. So substituting back, we have z^2 + 19 = z^{log_3(10)}. This equation is still transcendental and difficult to solve algebraically. So perhaps we need to use numerical methods here.
Alternatively, maybe I can graph both functions y = 3^{2x} + 19 and y = 10^x and see where they intersect. Since we already found x = 2 as a solution, let's see if there's another intersection point. Let's try x = 0: y1 = 1 + 19 = 20, y2 = 1. x = 1: y1 = 9 + 19 = 28, y2 = 10. x = 2: y1 = 81 + 19 = 100, y2 = 100. x = 3: y1 = 729 + 19 = 748, y2 = 1000. So between x = 2 and x = 3, y2 overtakes y1. Let's check x = 2.5: y1 = 9^{2.5} + 19 = 9^2 * 9^{0.5} + 19 = 81 * 3 + 19 = 243 + 19 = 262. y2 = 10^{2.5} ≈ 316.23. So y2 is higher here. So there's a crossing at x = 2. Let's check if there's another crossing before x = 2. Let's try x = 1.5: y1 = 9^{1.5} + 19 = 27 + 19 = 46. y2 = 10^{1.5} ≈ 31.62. So y1 is still higher. At x = 2, equal. So maybe only one crossing at x = 2. But let's check x = 1. Let's see, y1 = 28, y2 = 10. y1 is higher. x = 0.5: y1 = 3 + 19 = 22, y2 ≈ 3.16. So y1 is higher. So seems like only x = 2 is the solution. Wait, but maybe there's another solution for x < 0?
Wait, let's check x = -1: y1 = 3^{-2} + 19 = 1/9 + 19 ≈ 19.111, y2 = 10^{-1} = 0.1. So y1 is higher. x = -2: y1 = 3^{-4} + 19 ≈ 0.0123 + 19 ≈ 19.0123, y2 = 10^{-2} = 0.01. Still y1 higher. So for x negative, y1 is always higher than y2. Therefore, the only solution is x = 2. Wait, but let's confirm by testing another point. Let's try x = 2. Let's see, 3^{2*2} + 19 = 81 + 19 = 100, and 10^2 = 100. So that's correct. Is there another solution?
Alternatively, maybe we can use the Lambert W function. Let me recall that the Lambert W function is used to solve equations of the form z = W(z) e^{W(z)}. But I need to manipulate the equation into that form. Let me try to rewrite the equation 3^{2x} + 19 = 10^x. Let me set y = 10^x. Then, 3^{2x} = (3^x)^2. Since y = 10^x, then 3^x = (3/10)^x * 10^x = (3/10)^x * y. Therefore, 3^{2x} = y^2 * (3/10)^{2x} = y^2 * (3/10)^{2x}. Wait, this seems complicated. Alternatively, express 3^{2x} in terms of y. Since y = 10^x, then 3^x = (3/10)^x * y. Therefore, 3^{2x} = (3^x)^2 = [(3/10)^x * y]^2 = y^2 * (3/10)^{2x}. But (3/10)^{2x} = (3^2 / 10^2)^x = (9/100)^x. Therefore, 3^{2x} = y^2 * (9/100)^x. But 9/100 is 0.09, so 0.09^x. Hmm, this seems messy. Maybe this substitution isn't helpful.</think>

<think>Got it, let's continue. Alternatively, **let me consider the function** h(x) = 3^{2x} - 10^x. We need to find when h(x) = -19. Let me analyze h(x). **At x = 2,** h(2) = 81 - 100 = -19. So x = 2 is a solution. For x < 2, h(x) is positive (since 3^{2x} grows slower than 10^x but still positive?), and for x > 2, h(x) is negative. Wait, but at x = 1, h(1) = 9 - 10 = -1. At x = 0, h(0) = 1 - 1 = 0. Wait, h(0) = 0, h(1) = -1, h(2) = -19. Wait, that contradicts my previous thought. Wait, no, hold on. Wait, h(x) = 3^{2x} - 10^x. At x = 0, h(0) = 1 - 1 = 0. At x = 1, h(1) = 9 - 10 = -1. At x = 2, h(2) = 81 - 100 = -19. So h(x) is decreasing from x = 0 onwards? Wait, h(0) = 0, h(1) = -1, h(2) = -19. So h(x) is decreasing. Then, as x increases beyond 2, h(x) becomes more negative. So the function h(x) = 3^{2x} - 10^x is decreasing for x > 0. Therefore, there's only one solution at x = 2. Because once h(x) is negative, it continues to decrease, so no other crossing. Therefore, x = 2 is the only solution.\nTherefore, the answer is x = 2.\n**Final Answer** The solution is \\boxed{2}.</think><answer>2</answer>

Figure 7: Example 1.

<think>Okay, so I need to find the smallest positive integer that's a multiple of 30 and can only be written using the digits 0 and 2. Hmm, let's break this down step by step.
First, I remember that a number divisible by 30 must be divisible by both 3 and 10. Let me verify that. Yeah, 30 is 3 times 10, so any multiple of 30 needs to satisfy the divisibility rules for 3 and 10. Since 10 is a factor, the number must end with a 0. So whatever number we come up with has to end with a 0. That's good to know. So the last digit has to be 0. That means the number will be something like ...0, where the other digits are 0 or 2.
Now, the divisibility by 3 rule is that the sum of the digits must be a multiple of 3. So, for example, if the number is 20, the sum is 2 + 0 = 2, which isn't a multiple of 3. If it's 30, the sum is 3 + 0 = 3, which is good. But 30 is a multiple of 30? Wait, 30 is 3 times 10, so yes, 30 is a multiple of 30. But can we write 30 with only 0s and 2s? Wait, 30 has a 3 in it. So that's not allowed. So 30 is out. Then the next multiple of 30 is 60, which has a 6, which is not allowed. Then 90, which has a 9. So none of the multiples of 30 that are two-digit numbers work because they have digits other than 0 or 2. So we need to look at three-digit numbers or more.
So the next step is to find the smallest number composed of 0s and 2s that ends with a 0 and has a digit sum divisible by 3. Let's start checking numbers with more digits.
The smallest number with 0s and 2s ending with 0 is 20. Sum is 2 + 0 = 2, not divisible by 3. Next is 220. Sum is 2 + 2 + 0 = 4, not divisible by 3. Then 200. Sum is 2 + 0 + 0 = 2, nope. 2020: 2 + 0 + 2 + 0 = 4, still not. 2000: sum is 2, nope. 2220: sum is 2 + 2 + 2 + 0 = 6. Oh, 6 is divisible by 3. So 2220. Wait, 2220 is a multiple of 30? Let's check. 2220 divided by 30 is 74. So yes, 2220 is a multiple of 30. But is 2220 the smallest such number?
Wait, maybe there's a smaller number. Let's see. Let's check numbers with three digits. The three-digit numbers possible with digits 0 and 2 are 200, 220, 202, 2020 (wait, that's four digits). Wait, no. Three digits: 200, 220, 202. But 200 ends with 0, sum is 2. 220 sum is 4. 202 sum is 2 + 0 + 2 = 4. None of these are divisible by 3. So three digits don't work. So four digits. Let's check four-digit numbers.
Four-digit numbers with digits 0 and 2 ending with 0. So the last digit is 0, the first digit can't be 0. So possible numbers: 2000, 2002, 2020, 2022, 2200, 2202, 2220, 2222. Let's check their digit sums:
2000: 2 + 0 + 0 + 0 = 2
2002: 2 + 0 + 0 + 2 = 4
2020: 2 + 0 + 2 + 0 = 4
2022: 2 + 0 + 2 + 2 = 6
2200: 2 + 2 + 0 + 0 = 4
2202: 2 + 2 + 0 + 2 = 6
2220: 2 + 2 + 2 + 0 = 6
2222: 2 + 2 + 2 + 2 = 8
So among these, 2022, 2202, 2220 all have digit sums divisible by 3. Now, which is the smallest? Let's list them in order: 2000, 2002, 2020, 2022, 2200, 2202, 2220, 2222.
So the smallest among these is 2000, but sum is 2. Then 2002: sum 4, 2020: sum 4, 2022: sum 6, which is 2022. Then 2200: sum 4, 2202: sum 6, 2220: sum 6, 2222: sum 8. So the smallest four-digit number is 2022. But wait, 2022 is 2022. Is that the smallest? Let's check.
Wait, 2000 is smaller than 2022. But 2000's sum is 2. Then 2002 is 2002, which is smaller than 2022. 2002's sum is 4, which isn't divisible by 3. Then 2020 is 2020, sum 4. Then 2022 is 2022. So 2022 is the first four-digit number with digits 0 and 2, ending with 0, and sum divisible by 3.
But wait, maybe there is a three-digit number that I missed. Wait, three-digit numbers with digits 0 and 2 ending with 0. The three-digit numbers would be 200, 220, 202. None of these have a digit sum divisible by 3. So 2022 is the first candidate.
But wait, maybe there's a five-digit number that's smaller? Wait, no. Five-digit numbers are larger than four-digit numbers, so 2022 is smaller than any five-digit number.
Wait, but 2022 is 2022. Let me check if 2022 is indeed divisible by 30. 2022 divided by 30 is 67.4. Wait, that's not an integer. Wait, so 2022 is not divisible by 30. Wait, hold on. Wait, 2022 divided by 2 is 1011, which is an integer, so it's even. Divided by 5? 2022 ends with a 2, so no. So 2022 is not divisible by 5. Therefore, it's not divisible by 30. So my mistake earlier. So even though the digit sum is 6, which is divisible by 3, the number itself isn't divisible by 5, so it's not divisible by 30. Therefore, 2022 is out.
So that means my previous reasoning was flawed. Because even if the digit sum is divisible by 3, the number must also end with a 0 to be divisible by 10. So 2022 ends with 2, not 0, so it's not a multiple of 10. Therefore, even though it's a multiple of 3, it's not a multiple of 10, so not a multiple of 30. Therefore, my mistake.
So, the number must end with a 0. Therefore, the last digit is 0, so the number is ...0. So, let's think again.
So, we need a number composed of 0s and 2s, ending with 0, and the sum of digits divisible by 3.
So, the number is of the form ...0, where the preceding digits are 0s and 2s. Let's think about how to construct such a number.
Let me consider the possible numbers:
Starting with the smallest possible numbers.
Single-digit: 0. But 0 is not positive, so we need positive integers. So the next is 20. Sum 2 + 0 = 2, not divisible by 3.
Two-digit: 20, 02. But 02 is just 2, which is single-digit. So 20 is the only two-digit number with digits 0 and 2. Sum 2, not divisible by 3.
Three-digit: 200, 220, 202, 2200 (four-digit). Wait, 200: sum 2, 220: sum 4, 202: sum 4. None work.
Four-digit: Let's check four-digit numbers ending with 0. So, the number is _ _ _ 0, where the first digit is 2 or 0. But since it's a four-digit number, the first digit can't be 0. So possible numbers:
2000, 2002, 2020, 2022, 2200, 2202, 2220, 2222.
Check their digit sums:
2000: 2 + 0 + 0 + 0 = 2
2002: 2 + 0 + 0 + 2 = 4
2020: 2 + 0 + 2 + 0 = 4
2022: 2 + 0 + 2 + 2 = 6
2200: 2 + 2 + 0 + 0 = 4
2202: 2 + 2 + 0 + 2 = 6
2220: 2 + 2 + 2 + 0 = 6
2222: 2 + 2 + 2 + 2 = 8
So among these, 2022, 2202, 2220 have digit sums divisible by 3. Now, check if they are divisible by 10. They all end with 0, so they are divisible by 10. Therefore, they are divisible by 10. Then, to check if they are divisible by 3, since the digit sum is already divisible by 3, they are divisible by 3. Therefore, they are divisible by 30. So 2022, 2202, 2220 are all multiples of 30. Now, which is the smallest?
Compare 2022, 2202, 2220. The smallest is 2022. But wait, 2022 is 2022, which is a four-digit number. But is there a three-digit number? Wait, three-digit numbers can't end with 0 and have digits only 0 and 2. The three-digit numbers with digits 0 and 2 ending with 0 are 200, 220, 202. None of these have a digit sum divisible by 3. So 200: sum 2, 220: sum 4, 202: sum 4. So no.
So the next possible is four-digit numbers. The smallest four-digit number among the candidates is 2022. But wait, 2022 is 2022. But let's check 2022 divided by 30. 30 times 67 is 2010, 30 times 68 is 2040. So 2022 is between 30*67 and 30*68. So 2022 is not a multiple of 30. Wait, but we thought that since the digit sum is 6, which is divisible by 3, and it ends with 0, so it's divisible by 10 and 3, hence divisible by 30. But that's not correct. Wait, the divisibility rules: a number is divisible by 10 if it ends with 0, and divisible by 3 if the sum of digits is divisible by 3. Therefore, if a number ends with 0 and the sum of digits is divisible by 3, then it's divisible by 30. But 2022 ends with 2, not 0. Wait, no! Wait, 2022 ends with 2. Wait, hold on. Wait, 2022 is a four-digit number. Wait, 2022 is 2-0-2-2. So the last digit is 2. Therefore, it doesn't end with 0, so it's not divisible by 10. Therefore, it's not a multiple of 30. So my mistake was in thinking that 2022 ends with 0. Wait, no. Wait, 2022 is 2,0,2,2. So the last digit is 2. Therefore, it's not ending with 0. Therefore, my previous reasoning was wrong. So 2022 is not a multiple of 10, so not a multiple of 30. Therefore, even though the digit sum is 6, since it doesn't end with 0, it's not divisible by 10, hence not divisible by 30. Therefore, my mistake. So I need to correct that.
Therefore, the numbers that are multiples of 30 must end with 0. Therefore, the numbers we need are numbers composed of 0s and 2s ending with 0, with digit sum divisible by 3. So let's re-examine the four-digit numbers ending with 0. So the four-digit numbers are:
2000, 2002, 2020, 2022, 2200, 2202, 2220, 2222.
But only those ending with 0. So 2000, 2002, 2020, 2022, 2200, 2202, 2220, 2222. Wait, but 2000 ends with 0, 2002 ends with 2, so no. Wait, no. Wait, the four-digit numbers ending with 0 are 2000, 2020, 2200, 2220. Because 2002 ends with 2, 2022 ends with 2, 2202 ends with 2, 2222 ends with 2. So only 2000, 2020, 2200, 2220 are four-digit numbers ending with 0. Let's check their digit sums:
2000: 2 + 0 + 0 + 0 = 2
2020: 2 + 0 + 2 + 0 = 4
2200: 2 + 2 + 0 + 0 = 4
2220: 2 + 2 + 2 + 0 = 6
So among these, only 2220 has a digit sum divisible by 3. So 2220 is a four-digit number ending with 0, digit sum 6, so divisible by 3. Therefore, 2220 is divisible by 30. Let's check 2220 divided by 30: 2220 / 30 = 74. So yes, 74 times 30 is 2220. Therefore, 2220 is a multiple of 30. So 2220 is a candidate. But wait, earlier I thought 2022 was a candidate, but that was incorrect because it didn't end with 0. So 2220 is the first four-digit number ending with 0 with digit sum divisible by 3. So 2220 is 2220. Is there a smaller number?
Wait, let's check five-digit numbers. Wait, but five-digit numbers are larger than four-digit numbers, so 2220 is smaller than any five-digit number. So 2220 is a candidate. But let's check if there's a three-digit number that I missed. Wait, three-digit numbers ending with 0 are 200, 220, 202. None of these have digit sums divisible by 3. So 200: sum 2, 220: sum 4, 202: sum 4. So no. Therefore, 2220 is the smallest number composed of 0s and 2s, ending with 0, with digit sum divisible by 3. Therefore, 2220 is the answer.
Wait, but wait, let's check another angle. Maybe a number with more digits but smaller in value. For example, 2000 is a four-digit number, but its digit sum is 2, not divisible by 3. 2020: sum 4. 2200: sum 4. 2220: sum 6. So 2220 is the first one. But let's check if there's a three-digit number. Wait, no. So 2220 is the answer. But let me check another way. Maybe there's a number with more digits but smaller. For example, 2020 is a four-digit number, sum 4. 2200 sum 4. 2220 sum 6. So 2220 is the first. So 2220 is the answer.
Wait, but let's check if there's a number with more digits but smaller. For example, 20000. Sum is 2, nope. 20002: sum 4. 20020: sum 2. 20200: sum 4. 20202: sum 4. 20220: sum 6. Wait, 20220 is a five-digit number. Let's check: 2 + 0 + 2 + 2 + 0 = 6. So 20220. Is 20220 smaller than 2220? No, 20220 is a five-digit number, which is larger than 2220. So 2220 is still the smallest.</think>

<think>Got it, let's continue. **Therefore, the answer is 2220**.\n**Final Answer** The least positive integer multiple of 30 that can be written with only the digits 0 and 2 is \\boxed{2220}.</think><answer>2220</answer>

Figure 8: Example 2.

(2x^3 - 8x^3) + (-30x^2 + 120x^2) + (174x - 300x) + (-370 + 1000) = 0
Which is:
-6x^3 + 90x^2 - 126x + 630 = 0
Multiply both sides by -1 to make the leading coefficient positive:
6x^3 - 90x^2 + 126x - 630 = 0
We can factor out a 6:
6(x^3 - 15x^2 + 21x - 105) = 0
So x^3 - 15x^2 + 21x - 105 = 0
Now, we can try to factor this cubic equation. Let's check for rational roots using Rational Root Theorem. The possible rational roots are factors of 105 divided by factors of 1, so ±1, ±3, ±5, ±7, ±15, ±21, ±35, ±105.
Let's test x=3:
3^3 -15*(3)^2 +21*3 -105 = 27 - 135 + 63 -105 = (27 -135) + (63 -105) = (-108) + (-42) = -150 ≠ 0
x=5:
5^3 -15*5^2 +21*5 -105 = 125 - 15*25 + 105 -105 = 125 - 375 + 0 = -250 ≠0
x=7:
7^3 -15*7^2 +21*7 -105 = 343 - 15*49 + 147 -105 = 343 - 735 + 42 = (343 +42) -735 = 385 -735 = -350 ≠0
x=15: Probably too big, but let's check:
15^3 -15*15^2 +21*15 -105 = 3375 - 3375 + 315 -105 = 0 + 210 = 210 ≠0
Hmm, none of these are working. Wait, but we know from the previous method that the roots are x=3,5,7. So maybe I made a mistake in the expansion?
Wait, let's check the expansion again.
Original equation: (x -3)^3 + (x -7)^3 = (2x -10)^3
Left side:
First term: (x -3)^3 = x^3 - 9x^2 + 27x -27
Second term: (x -7)^3 = x^3 -21x^2 +147x -343
Adding them:
2x^3 -30x^2 +174x -370. Correct.
Right side: (2x -10)^3. Let's expand that again.
(2x -10)^3 = (2x)^3 - 3*(2x)^2*10 + 3*(2x)*(10)^2 - (10)^3
= 8x^3 - 3*4x^2*10 + 3*2x*100 -1000
= 8x^3 - 120x^2 + 600x -1000. Correct.
So bringing all terms to left:
Left - Right = 0:
2x^3 -30x^2 +174x -370 -8x^3 +120x^2 -600x +1000 =0
So:
2x^3 -8x^3 = -6x^3
-30x^2 +120x^2 =90x^2
174x -600x = -426x
-370 +1000 =630
So equation: -6x^3 +90x^2 -426x +630=0
Multiply by -1: 6x^3 -90x^2 +426x -630=0
Divide by 6: x^3 -15x^2 +71x -105=0
Wait, but previously I had 6(x^3 -15x^2 +21x -105)=0. Hmm, conflicting results. Wait, where did I go wrong?
Wait, in the first method, after substitution, we had 6y(y^2 -4)=0, which led to y=0,2,-2, hence x=5,7,3.
But in the second method, expanding directly, we ended up with equation x^3 -15x^2 +21x -105=0.
Wait, so there's a discrepancy here. Which one is correct?
Wait, let me check the second method again.
Original equation:
Left side: (x -3)^3 + (x -7)^3 = 2x^3 -30x^2 +174x -370
Right side: (2x -10)^3 =8x^3 -120x^2 +600x -1000
Set equal:
2x^3 -30x^2 +174x -370 =8x^3 -120x^2 +600x -1000
Bring all terms to left:
2x^3 -30x^2 +174x -370 -8x^3 +120x^2 -600x +1000=0
Combine:
(2x^3 -8x^3) + (-30x^2 +120x^2) + (174x -600x) + (-370 +1000)=0
Which is:
-6x^3 +90x^2 -426x +630=0
Multiply by -1:
6x^3 -90x^2 +426x -630=0
Divide by 6:
x^3 -15x^2 +71x -105=0
Wait, but in the first method, after substitution, we had 6y(y^2 -4)=0, leading to y=0,2,-2, which translates to x=5,7,3, which are roots of the equation x^3 -15x^2 +21x -105=0.
But according to the second method, the cubic equation is x^3 -15x^2 +71x -105=0. Wait, so which is correct?
Wait, let's check with x=3 in both equations.
First equation: x^3 -15x^2 +21x -105
At x=3: 27 -135 +63 -105 = (27 -135) + (63 -105) = (-108) + (-42) = -150 ≠0
But x=3 is a root in the first method. So there must be a mistake in the second method.
Wait, that's confusing. How come x=3 is a root in the first method but not in the second? That can't be. There must be an error in one of the methods.
Wait, let's check the first method again.
Original equation:
Left side: (x -3)^3 + (x -7)^3
Right side: (2x -10)^3
We set y =x -5, so x = y +5.
Left side: (y +2)^3 + (y -2)^3
Right side: (2(y +5) -10)^3 = (2y +10 -10)^3 = (2y)^3
Expanding left side:
(y +2)^3 + (y -2)^3 = [y^3 +6y^2 +12y +8] + [y^3 -6y^2 +12y -8] = 2y^3 +24y
Right side: 8y^3
So equation: 2y^3 +24y =8y^3
Bring all terms to left: 2y^3 +24y -8y^3=0 → -6y^3 +24y=0 → 6y(-y^2 +4)=0 → y=0, y=±2
So solutions y=0,2,-2. Therefore, x=5,7,3. These are roots of the equation (x -3)^3 + (x -7)^3 - (2x -10)^3=0. So the cubic equation must be (x -3)(x -5)(x -7)=0. Let's expand that.
First, (x -3)(x -5)(x -7). Let's compute (x -3)(x -5) first:
(x -3)(x -5) = x^2 -8x +15
Multiply by (x -7):
(x^2 -8x +15)(x -7) = x^3 -7x^2 -8x^2 +56x +15x -105 = x^3 -15x^2 +71x -105
Ah! So the cubic equation is x^3 -15x^2 +71x -105=0. But in the first method, we had y=0,2,-2 leading to x=5,7,3, which are roots of this equation. However, when we substituted x=3 into the cubic equation x^3 -15x^2 +71x -105, we get 27 -135 +21*3 -105 = 27 -135 +63 -105 = (27 +63) - (135 +105) = 90 -240 = -150 ≠0. Wait, that's not zero. But x=3 is a root according to the first method. What's going on here?
Wait, this is a contradiction. If x=3 is a root, then substituting into the cubic equation should yield zero, but it doesn't.
Therefore, there must be a mistake in one of the methods.
Wait, let's check the substitution again. Let's compute the left side and right side when x=3.
Left side: (3 -3)^3 + (3 -7)^3 = 0 + (-4)^3 = -64
Right side: (2*3 -10)^3 = (-4)^3 = -64. So equation holds. So x=3 is a valid solution.</think>

<think>Got it, let's continue. **But in the first method, when we set y =x -5, then x=3 corresponds to y=-2, which was a root of 6y(y^2 -4)=0, so y=-2 is a root, leading to x=3. So why is x=3 not a root in the second method?**\nWait, when we did the second method, we ended up with the cubic equation x^3 -15x^2 +21x -105=0. Let's check if x=3 is a root here.\n3^3 -15*3^2 +21*3 -105 = 27 - 135 +63 -105= (27+63) - (135+105)=90 -240= -150≠0. So x=3 is not a root here. Therefore, there's a contradiction. So which is correct?\nWait, let's check the original equation again with x=3.\nLeft side: (3-3)^3 + (3-7)^3 =0 + (-4)^3= -64\nRight side: (2*3 -10)^3= (6-10)^3= (-4)^3= -64. So it's correct. Therefore, x=3 is a solution.\nBut according to the second method, x=3 is not a root. Therefore, there must be an error in the second method. Let's see where.\nWait, let's go back to the second method:\nOriginal equation:\n(x -3)^3 + (x -7)^3 = (2x -10)^3\nExpanding both sides:\nLeft side:\n(x^3 -9x^2 +27x -27) + (x^3 -21x^2 +147x -343) = 2x^3 -30x^2 +174x -370\nRight side:\n(8x^3 -120x^2 +600x -1000)\nBringing everything to the left:\n2x^3 -30x^2 +174x -370 -8x^3 +120x^2 -600x +1000 =0\nCombine like terms:\n(2x^3 -8x^3) + (-30x^2 +120x^2) + (174x -600x) + (-370 +1000)= -6x^3 +90x^2 -426x +630=0\nDivide by -6:\nx^3 -15x^2 +71x -105=0\nBut when we plug x=3 into this equation:\n3^3 -15*3^2 +71*3 -105=27 -135 +213 -105= (27+213) - (135+105)=240 -240=0. Wait, 27 +213 is 240, and 135 +105 is 240. So 240 -240=0. Therefore, x=3 is a root. Wait, so earlier calculation was wrong. Let me recalculate:\nx=3:\nx^3=27\n15x^2=15*9=135\n71x=71*3=213\nSo 27 -135 +213 -105= (27 +213) - (135 +105)=240 -240=0. Yes, correct. So x=3 is a root.\nWait, so why did I get confused earlier? Because I miscalculated. So x=3 is indeed a root. Then, perhaps my earlier mistake was in the calculation steps. Let me check again.\nWait, the cubic equation is x^3 -15x^2 +71x -105=0. Let's factor this cubic equation.\nSince we know that x=3 is a root, let's perform polynomial division or use synthetic division.\nDivide x^3 -15x^2 +71x -105 by (x -3):\nUsing synthetic division:\n3 | 1 -15 71 -105\n 3 -36 105\n 1 -12 35 0\nSo the cubic factors as (x -3)(x^2 -12x +35). Then factor the quadratic:\nx^2 -12x +35. Find two numbers that multiply to 35 and add to -12: -5 and -7.\nSo x^2 -12x +35=(x -5)(x -7). Therefore, the cubic factors as (x -3)(x -5)(x -7). Therefore, the roots are x=3,5,7. Which matches the first method.\nTherefore, the roots are x=3,5,7. So both methods agree. Earlier confusion was due to miscalculations. Therefore, the correct roots are x=3,5,7.\nTherefore, the answer is x=3,5,7.\n**Final Answer** The roots are \\boxed{3}, \\boxed{5}, and \\boxed{7}.\nAlternatively, if the problem expects a single boxed answer listing all roots, it might be written as \\boxed{3}, \\boxed{5}, \\boxed{7}, but typically for multiple roots, we can list them inside a single box separated by commas. However, standard practice is to box each root separately if multiple. But since the user instruction says \"put your final answer within \\boxed{}\", perhaps we need to present all roots in one box. However, standard practice in math problems is to box each root separately. But maybe the system expects multiple boxes. Wait, looking back at the original problem: \"put your final answer within \\boxed{}\". The instruction is singular, so maybe all roots in one box. Let me check the initial problem statement again.\nOriginal problem: \"Find the roots of (x - 3)^3 + (x - 7)^3 = (2x - 10)^3.\" So they are three roots: 3,5,7. The standard way to present multiple roots is either as a list inside a single box or each in separate boxes. But since the user says \"put your final answer within \\boxed{}\", perhaps they want all roots in one box. However, in many math problems, when multiple roots, they are listed within a single box separated by commas. So I think the answer should be \\boxed{3}, \\boxed{5}, \\boxed{7} if multiple boxes are allowed, but since the instruction says \"within \\boxed{}\", maybe the system expects all roots in one box. Let me check previous examples. For example, if the answer is multiple numbers, like 3,5,7, then in one box: \\boxed{3, 5, 7}. Yes, that's the standard way. So I should write \\boxed{3, 5, 7}.\n**Final Answer** \\boxed{3, 5, 7}</think><answer>3, 5, 7</answer>

Figure 9: Example 3.

<think>Okay, so I need to find the greatest possible slope of the line connecting points A and B. Point A is somewhere in or on the square with corners at (0,0) and (2,2), and point B is somewhere in or on the square with corners at (4,2) and (5,3).
Hmm, let's start by visualizing these squares.
First, the square containing point A has opposite corners at (0,0) and (2,2). That means the square is from x=0 to x=2 and y=0 to y=2. So, point A can be anywhere in that square, including the edges. Similarly, the square for point B has opposite corners at (4,2) and (5,3). Let me figure out the coordinates of that square. The x-coordinates go from 4 to 5, and the y-coordinates go from 2 to 3. So, point B can be anywhere in that square as well.
The slope between two points (x1, y1) and (x2, y2) is (y2 - y1)/(x2 - x1). To maximize the slope, we need to maximize the difference in the y-coordinates while minimizing the difference in the x-coordinates. Since the slope is rise over run, a larger rise with a smaller run will give a steeper slope.
So, for point A, the maximum y-coordinate is 2 (since the square goes up to y=2), and the minimum x-coordinate is 0. For point B, the maximum y-coordinate is 3 (since the square goes up to y=3), and the minimum x-coordinate is 4. Therefore, if we take point A at (0,2) and point B at (4,3), the slope would be (3 - 2)/(4 - 0) = 1/4. Wait, that's only 1/4. But maybe there's a way to get a steeper slope by choosing different points?
Wait a second, maybe I can get a steeper slope by choosing points that are not at the corners. For instance, if point A is at (0,2) and point B is at (5,3), but point B can't go beyond (5,3). Wait, no, the square for B is from (4,2) to (5,3). So, point B must be within or on that square, so the maximum x is 5 and the maximum y is 3. So, if I take point A at (0,2) and point B at (5,3), the slope is (3 - 2)/(5 - 0) = 1/5, which is less steep than 1/4. Hmm, not better.
Alternatively, if I take point A at (0,2) and point B at (4,3), that's 1/4. What if I take point A at (2,2) and point B at (4,3)? Then the slope is (3 - 2)/(4 - 2) = 1/2. That's better. Wait, so 1/2 is steeper than 1/4. But is that allowed? Let me check. Point A is at (2,2), which is a corner of the square (0,0) to (2,2), so that's valid. Point B is at (4,3), which is a corner of the square (4,2) to (5,3). So that's valid. So that gives a slope of 1/2. Is that the maximum?
Wait, maybe we can get a steeper slope by choosing different points. Let's think. The slope is (y_B - y_A)/(x_B - x_A). To maximize this, we need to maximize y_B - y_A and minimize x_B - x_A. So, the maximum y_B is 3, and the minimum y_A is 0. So, if we can have y_B = 3 and y_A = 0, that would give the maximum difference in y. Then, we need to minimize the difference in x. The minimum x difference would be when x_B is as large as possible and x_A is as small as possible. The maximum x_B is 5, and the minimum x_A is 0. So, if we take x_B = 5 and x_A = 0, then the slope is (3 - 0)/(5 - 0) = 3/5 = 0.6. Wait, that's 3/5. But is that possible?
Wait, but point A is at (0,0) and point B is at (5,3). But point A is allowed to be anywhere within or on the square (0,0) to (2,2), so (0,0) is allowed. Point B is allowed to be anywhere within or on the square (4,2) to (5,3), so (5,3) is allowed. So, the slope would be (3 - 0)/(5 - 0) = 3/5. That's 0.6, which is steeper than 1/2. Wait, so that's better. But is that correct? Let me check.
Wait, point A can be at (0,0), which is a corner of its square. Point B can be at (5,3), which is a corner of its square. So, that's allowed. Then, the slope is 3/5. But is that the maximum possible? Let me see.
Alternatively, if we take point A at (0,0) and point B at (4,3), then the slope is (3 - 0)/(4 - 0) = 3/4 = 0.75. That's steeper. Wait, that's even better. So, (0,0) to (4,3). Let me verify. Point A is at (0,0), which is allowed. Point B is at (4,3), which is allowed. So, slope is (3 - 0)/(4 - 0) = 3/4. That's 0.75. That's steeper than 3/5. So, 3/4 is better. Hmm.
Wait, but maybe even steeper. Let's see. If point A is at (0,0) and point B is at (5,3), then slope is 3/5. But if point B is at (4,3), which is 4 units in x and 3 units in y, so 3/4. If point B is at (4,3), which is allowed, then that's better. So, 3/4 is better.
Alternatively, if we take point A at (0,0) and point B at (4,2), which is the minimum y for point B. Then slope is (2 - 0)/(4 - 0) = 2/4 = 1/2. That's less steep. So, not better.
Wait, so the maximum y for point B is 3, and the minimum y for point A is 0. So, the maximum difference in y is 3 - 0 = 3. Then, the x difference can be minimized. The minimum x difference is when x_B is maximum and x_A is minimum. So, x_B can be 5 and x_A can be 0, giving a difference of 5. Then, slope is 3/5. But if x_B is 4 and x_A is 0, then x difference is 4, slope is 3/4. Which is steeper. So, 3/4 is better.
Alternatively, if point A is at (0,0) and point B is at (4,3), which is allowed. So, that gives a steeper slope. So, 3/4. But can we get a steeper slope?
Wait, what if we take point A not at (0,0)? For example, if point A is somewhere else in the square (0,0) to (2,2), but with a lower y-coordinate? Wait, but if we take point A with a lower y-coordinate, then y_B - y_A would be less. For example, if point A is at (0,1), then y_B - y_A can be up to 3 - 1 = 2, which is less than 3. So, that would decrease the slope. So, to maximize the numerator (y_B - y_A), we need point A to have the minimum possible y-coordinate, which is 0.
Similarly, for the denominator (x_B - x_A), to minimize it, we need point B to have the maximum x-coordinate (5) and point A to have the minimum x-coordinate (0). So, x difference is 5 - 0 = 5. Then, slope is 3/5. But if we take point B at 4 instead of 5, then x difference is 4, which gives a steeper slope. So, 3/4.
Wait, but point B can be at (4,3). So, if point A is at (0,0) and point B is at (4,3), then slope is 3/4. If point B is at (5,3), slope is 3/5. So, 3/4 is better.
But can we get a steeper slope? Let's see. Suppose we take point A not at (0,0) but at some other point where y_A is less than 0? No, because point A is within or on the square (0,0) to (2,2), so y_A can't be less than 0. Similarly, point B can't have y_B less than 2. So, the maximum possible y_B is 3, y_A is 0. So, the difference is 3. If we take x_B as 4 and x_A as 0, then x difference is 4. So, slope is 3/4.
Alternatively, if point A is at (0,0) and point B is at (4,3), which is allowed. So, slope is 3/4. Is there a way to get a steeper slope?
Wait, maybe if point A is at (0,0) and point B is somewhere else in its square. For example, if point B is at (4,2), which is the minimum y for point B, then slope is 2/4 = 1/2. Not better. If point B is at (5,3), slope is 3/5. If point B is at (5,2), slope is (2 - 0)/(5 - 0) = 2/5. Less steep. So, the maximum slope from (0,0) is 3/4.
Alternatively, if point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.
But wait, what if point A is not at (0,0)? For example, if point A is somewhere else in the square (0,0) to (2,2), but with a lower y-coordinate? Wait, but if point A has a lower y-coordinate, then y_B - y_A would be less. For example, if point A is at (0,1), then y_B - y_A can be at most 3 - 1 = 2, which is less than 3. So, that would decrease the slope. Therefore, to maximize the slope, point A should be at (0,0) with y=0.
Similarly, for point B, to maximize y_B, it should be at (5,3). But if we take point B at (5,3), then x_B is 5, which is the maximum x. But if we take point B at (4,3), x_B is 4, which allows a smaller x difference. So, (4 - 0) = 4, which gives a steeper slope.
Wait, so the maximum slope is achieved when point A is at (0,0) and point B is at (4,3). Let me confirm that. The slope is (3 - 0)/(4 - 0) = 3/4. That's 0.75. Is there a way to get a steeper slope?
Wait, maybe if we take point A not at (0,0) but somewhere else in its square, but with a lower y-coordinate? Wait, but that would decrease the slope. Alternatively, if point A is at (0,0) and point B is somewhere else. Wait, but point B is at (4,3). If point B is at (5,3), slope is 3/5. If point B is at (4,3), slope is 3/4. So, 3/4 is better.
Alternatively, if point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (4,3), that's allowed. So, that seems to be the maximum. But wait, let's check if there's another combination where the slope is steeper.
Suppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. Alternatively, if point A is at (0,0) and point B is at (5,3), slope is 3/5. If point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.
Alternatively, if point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (4,3), that's allowed. So, that's the maximum.
Wait, but let's think again. The square for point A is (0,0) to (2,2), so the maximum y for point A is 2, but if we take point A at (0,0), then y_A is 0, which is allowed. Similarly, the square for point B is (4,2) to (5,3), so the maximum y for point B is 3. So, to maximize y_B - y_A, we need y_B as large as possible and y_A as small as possible. So, y_B = 3, y_A = 0. Then, x_B as large as possible (5) and x_A as small as possible (0). Then, x_B - x_A = 5 - 0 = 5, so slope is 3/5. But if x_B is 4 and x_A is 0, then x difference is 4, so slope is 3/4. Which is steeper. So, 3/4 is better.
But if point B is at (4,3), which is allowed, then that's a valid point. So, the slope is 3/4. So, is that the maximum?
Wait, another thought. If point A is not at (0,0), but somewhere else in the square (0,0) to (2,2), but with a lower y-coordinate? But that would decrease y_A, which would decrease the numerator. So, that's not helpful. Alternatively, if point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.
Alternatively, if point A is at (0,0) and point B is at (4,3), which is allowed, slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better. So, 3/4 is the maximum.
Wait, but let's check another combination. Suppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.
Alternatively, if point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.
Wait, but let's check another possibility. Suppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. If point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.</think>

<think>Got it, let's continue. **Alternatively, is there a way to get a slope greater than 3/5? Let's see. Suppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is higher**.\nWait, 3/4 is 0.75, and 3/5 is 0.6. So, 3/4 is higher. So, 3/4 is better.\nBut is there a way to get a higher slope than 3/4? Let's see.\nSuppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. If point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (4,2), slope is 2/4 = 1/2. If point A is at (0,0) and point B is at (5,2), slope is 2/5 = 0.4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is the maximum here.\nAlternatively, if point A is at (0,0) and point B is at (5,3), slope is 3/5. If point A is at (0,0) and point B is at (5,3), that's 3/5. So, 3/4 is better.\nWait, but let's check another combination. Suppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.\nAlternatively, if point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.\nAlternatively, if point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is higher.\nAlternatively, is there a way to get a steeper slope by moving point A not at (0,0) but somewhere else?\nSuppose point A is at (0,1) and point B is at (4,3). Then, slope is (3 - 1)/(4 - 0) = 2/4 = 1/2. Less steep.\nIf point A is at (0,1) and point B is at (5,3), slope is (3 - 1)/(5 - 0) = 2/5 = 0.4. Less steep.\nIf point A is at (0,1) and point B is at (4,2), slope is (2 - 1)/(4 - 0) = 1/4 = 0.25. Even less steep.\nAlternatively, if point A is at (1,1) and point B is at (4,3). Slope is (3 - 1)/(4 - 1) = 2/3 = 0.666. Less than 3/4.\nIf point A is at (1,1) and point B is at (5,3), slope is (3 - 1)/(5 - 1) = 2/4 = 1/2. Less steep.\nAlternatively, if point A is at (0,2) and point B is at (4,3). Slope is (3 - 2)/(4 - 0) = 1/4 = 0.25. Less steep.\nAlternatively, if point A is at (0,2) and point B is at (5,3), slope is (3 - 2)/(5 - 0) = 1/5 = 0.2. Less steep.\nSo, it seems that the maximum slope is achieved when point A is at (0,0) and point B is at (4,3), giving a slope of 3/4. But wait, let's check another combination.\nWait, what if point A is at (0,0) and point B is at (4,3). Slope is 3/4. What if point A is at (0,0) and point B is at (4,3). That's allowed. What if point A is at (0,0) and point B is at (5,3). Slope is 3/5. So, 3/4 is higher.\nAlternatively, is there a way to get a higher slope by choosing point A not at (0,0) but somewhere else?\nSuppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is better.\nWait, but wait. Let me check if there's a point A not at (0,0) that can give a higher slope. For example, suppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. If point A is at (0,1) and point B is at (5,3), slope is 2/5 = 0.4. Less. If point A is at (0,0) and point B is at (4,3), slope is 3/4. If point A is at (0,0) and point B is at (5,3), slope is 3/5. So, 3/4 is higher.\nAlternatively, is there a point A not at (0,0) but can give a higher slope. For example, suppose point A is at (0,0) and point B is at (4,3). Slope is 3/4. Point A at (0,0) and point B at (5,3). Slope 3/5. So, 3/4 is higher.\nAlternatively, is there a way to get a higher slope by choosing point A somewhere else?\nSuppose point A is at (1,1) and point B is at (4,3). Slope is 2/3 = 0.666. Less than 3/4.\nAlternatively, point A at (0,0) and point B at (4,3). Slope 3/4. That's the maximum so far.\nWait, but let's check another combination. Suppose point A is at (0,0) and point B is at (5,3). Slope is 3/5. If point A is at (0,0) and point B is at (4,3), slope is 3/4. So, 3/4 is higher.\nAlternatively, point A at (0,0) and point B at (4,3). Slope 3/4. If point A is at (0,0) and point B is at (5,3). Slope 3/5. Less than 3/4.\nAlternatively, if point A is at (0,0) and point B is at (5,3). Slope 3/5. If point A is at (0,0) and point B is at (4,3), slope 3/4. So, 3/4 is higher.\nAlternatively, is there a way to get a slope higher than 3/4?\nWait, let's think of another approach. The maximum slope would be achieved when the line from point A to point B is as steep as possible. Since point A is in the square [0,2]x[0,2] and point B is in [4,5]x[2,3], the steepest line would be when point A is at the bottom-left corner (0,0) and point B is at the top-right corner (5,3). But wait, the top-right corner of the square containing B is (5,3). So, if we connect (0,0) to (5,3), the slope is 3/5. But if we connect (0,0) to (4,3), which is a corner of the square containing B, the slope is 3/4. So, 3/4 is higher. So, even though (5,3) is a corner of the square containing B, connecting to (4,3) gives a steeper slope.\nTherefore, the maximum slope is 3/4.\nBut let me confirm once again. Suppose point A is at (0,0) and point B is at (4,3). Then, the line from (0,0) to (4,3) has a slope of 3/4. If we try to make the slope steeper, we need a larger rise over a smaller run. The rise here is 3 (from 0 to 3) and the run is 4 (from 0 to 4). So, 3/4. If we could have a rise of 4 and run of 1, that would be 4, but that's not possible here. Since point A can only go up to y=2, and point B can only go up to y=3. So, the maximum rise is 3. The minimum run is 4 (from x=0 to x=4). So, 3/4 is the maximum possible slope.\nTherefore, the greatest possible value of the slope is 3/4.\n**Final Answer** The greatest possible value of the slope is \\boxed{\\dfrac{3}{4}}.</think><answer>\\dfrac{3}{4}</answer>

Figure 10: Example 4.