# Thought Manipulation: External Thought Can Be Efficient for Large Reasoning Models

Anonymous ACL submission

## Abstract

Recent advancements in large reasoning models (LRMs) have demonstrated the effectiveness of scaling test-time computation to enhance reasoning capabilities in multiple tasks. However, LRMs typically suffer from "overthinking" problems, where models generate significantly redundant reasoning steps while bringing limited performance gains. Existing work relies on fine-tuning to mitigate overthinking, which requires additional data, unconventional training setups, risky safety misalignment, and poor generalization.

Through empirical analysis, we reveal an important characteristic of LRM behaviors that placing external CoTs generated by smaller models between the thinking token (`<think>` and `</think>`) can effectively manipulate the model to generate fewer thoughts. Building on these insights, we propose a simple yet efficient pipeline, ThoughtMani, to enable LRMs to bypass unnecessary intermediate steps and reduce computational costs significantly. We conduct extensive experiments to validate the utility and efficiency of ThoughtMani. For instance, when applied to QwQ-32B on the LiveBench/Code dataset, ThoughtMani keeps the original performance and reduces output token counts by approximately 30%, with little overhead from the CoT generator. Furthermore, we find that ThoughtMani enhances safety alignment by an average of 10%. Since model vendors typically serve models of different sizes simultaneously, ThoughtMani provides an effective way to construct more efficient and accessible LRMs for real-world applications.

## 1 Introduction

Recent advancements in large reasoning models (LRMs) have demonstrated the great potential of incorporating long-thinking processes in enhancing reasoning capabilities for complex tasks (Plaat et al., 2024; Xu et al., 2025). By leveraging reinforcement learning (RL), LRMs are trained to generate step-by-step chain-of-thought (CoT) reasoning, breaking down problems into smaller components and performing multiple checks before arriving at a final response (OpenAI, 2025a; Team et al., 2025a; Shao et al., 2024). Models like DeepSeek-R1 (DeepSeek-AI et al., 2025) and QwQ (Team, 2025) exemplify the effectiveness of this method, showcasing substantial improvements in reasoning accuracy.

Despite the improved performance, scaling up CoT often requires exponentially larger computational resources (Sui et al., 2025). Models like QwQ typically consume 5 to 10 times more tokens to reach conclusions compared to standard approaches. Previous studies introduce "overthinking" to describe the phenomenon that unnecessary steps may lead to inefficiencies, particularly in simple questions (Chen et al., 2024). This inefficiency not only undermines the utility of LRMs in time-sensitive scenarios but also leads to additional computational resource consumption, further degrading overall system performance.

Existing work has explored fine-tuning-based techniques to mitigate inefficiencies related to overthinking (Chen et al., 2024; Sui et al., 2025). They rely on constructing datasets that consist of different reasoning compression patterns, either skipping less critical tokens (Xia et al., 2025) or introducing task arithmetic (Ma et al., 2025) to manipulate the parameter. However, these fine-tuning-based methods require additional data collection, leading to increased costs. Additionally, fine-tuning may introduce safety misalignment (Liu et al., 2024b; Yi et al., 2024; Gong et al.)

To solve the problem, we delve into the inherent characteristics of when an LRM enters and exits its "thinking" state. Specifically, we examine how LRMs behave when presented with human-provided thoughts framed within explicit thinking tokens (`<think>` and `</think>`) during the prompt phase. Through empirical analysis, we uncover the following distinct patterns in the behavior of LRMs trained via different methods. For RL-based LRMs, these models continue generating thoughts until they "perceive" that sufficient reasoning has been conducted, irrespective of encountering the closing `</think>` token. For Distillation-based LRMs, they terminate the reasoning process immediately upon encountering a `</think>` token, regardless of the quantity or quality of thoughts generated.

Given these insights, we propose ThoughtMani, a training-free method to reduce the computational cost generated due to the "overthinking" problem. By providing a reasoning process generated by a smaller non-reasoning model (CoT generator), e.g., Qwen-2.5-7b-instruct (Yang et al., 2024), and inserting it between thinking tokens, the reasoning model, e.g., QwQ, can directly extract sufficient information from the provided thoughts, thus bypassing unnecessary intermediate steps. Since the CoT generators normally cost much less computational resources than the reasoning models, ThoughtMani can significantly reduce the inference cost.

We comprehensively evaluate ten different compression techniques (replicated three times each) across three LRMs on four diverse datasets. Additionally, we perform ablation studies to analyze the key factors in the proposed method, ensuring a thorough validation of its effectiveness and robustness. For instance, when using Qwen-2.5-7B-Instruct as the CoT generator for QwQ-32B on the GSM-8k dataset, ThoughtMani reduces the output token count by approximately 40% (from 1,791 to 1,075 tokens), with an average additional cost of only 52 tokens from the CoT generator. Additionally, we investigate the safety performance of ThoughtMani and find that it provides an average safety gain of approximately 10% in most cases, whereas other fine-tuning-based methods exhibit a safety drop of 7%. These results demonstrate that our method not only significantly reduces computational overhead but also maintains the reasoning accuracy and enhances the safety alignment of LRMs.

Our contribution can be summarized as follows:

- We reveal a unique pattern of LRMs when external thoughts are given, which sheds light on LRMs' characteristics. Specifically, we uncover how RL-based and distillation-based LRMs differ in their handling of provided CoT, offering insights into their reasoning mechanisms and decision-making processes.

- We propose a training-free inference pipeline, ThoughtMani, to reduce redundant reasoning tokens. By leveraging smaller CoT generators and strategically inserting thoughts within the reasoning process, our approach achieves significant computational savings without compromising performance or requiring additional training resources.

- Extensive experiments on three models and four datasets validate the superiority of ThoughtMani in terms of utility, efficiency, and safety. Our results demonstrate consistent improvements across diverse datasets and tasks, highlighting its practical applicability and robustness in real-world scenarios.

## 2 Related Work

### 2.1 Large Reasoning Model

By scaling up training data size and model size, large language models (LLMs) have developed powerful language understanding and generation capabilities (Zhao et al., 2023), such as GPT-4o (Hurst et al., 2024) and DeepSeekv3 (DeepSeek-AI et al., 2024), enabling rapid and coherent responses to user inputs. However, these models perform poorly when facing complex reasoning and logical analysis tasks (Xu et al., 2025; Zhao et al., 2023), falling far short of human cognitive levels. To address this issue, recent studies focus on improving the capability of language models by utilizing more inference-time computation instead of simply scaling model parameters (Snell et al., 2024). This line of research has already outcome many powerful LRMs such as DeepSeek-R1 (DeepSeek-AI et al., 2025), OpenAI-o1/o3 (OpenAI, 2025a,b), and QwQ (Team, 2025), which shift from fast, intuitive processing to structured, step-by-step reasoning. Deeper reasoning capabilities enable these LRMs to make remarkable improvements when tackling challenging tasks like advanced mathematics and logical reasoning (Team et al., 2025b).

### 2.2 Chain-of-Thought Compression

Despite the improved capabilities, introducing intermediate CoTs brings additional token overhead.

To enable efficient inference without performance degradation, one line of research is to shorten the length of CoT while maintaining its effectiveness. For traditional LLMs, lots of efforts have been put into reducing redundant steps (Ding et al., 2024; Liu et al., 2024a; Ma et al., 2025) or skipping less-important tokens (Han et al., 2024; Xia et al., 2025) Another line is to represent the CoT using latent space compression (Deng et al., 2024; Cheng and Van Durme, 2024), which utilize embeddings instead of tokens to serve as the CoT. With the development of LRMs, the extensive inherent thinking process not only improves the reasoning capability, but also bring computation inefficiency (Chen et al., 2024). Existing work mainly relies on fine-tuning to control model behaviors (Xia et al., 2025; Ma et al., 2025). However, these fine-tuning require additional data and is prohibitively expensive for larger models like (DeepSeek-AI et al., 2025) and introduce unexpected safety misalignment (Liu et al., 2024b; Yi et al., 2024).

## 3 Thought Manipulation

In this section, we investigate when an LRM enters and exits its "thinking" state. A key observation is that all thoughts generated by LRMs are consistently framed within designated thinking tokens, specifically `<think>` and `</think>`. This consistent pattern intuitively suggests that inserting a pre-generated CoT between explicit thinking tokens may allow the model to effectively utilize this external reasoning, thereby reducing the necessity to internally generate intermediate reasoning steps. To validate this hypothesis, we first randomly collect 100 samples each from the GSM-8k and MATH-500 datasets, which are widely used datasets for benchmarking the model's reasoning ability. We then select several Qwen series models (Yang et al., 2024), including Qwen-Max, Qwen-Plus, Qwen-2.5-7B-Instruct, and Qwen-2.5-3B-Instruct, to generate high-level ideas for solving the problems. The generation process is guided by the provided prompt in Appendix A, which focuses solely on high-level reasoning steps without delving into detailed calculations or producing final answers. By employing CoTs across models of different scales, we obtain CoTs with different levels of quality.

Next, we insert generated thoughts, which are enclosed within `<think>` and `</think>` tokens, at the end of a standardized inference template, together with the user's prompts. The code example is shown in Appendix A. The template is then used to invoke LRMs such as QwQ and Deepseek-Distillation-Qwen-2.5-32b(14b)-instruct. This placement allows us to observe how effectively the LRMs leverage the provided CoTs to streamline their reasoning process and reduce unnecessary intermediate steps.

Table 1: Number of occurrences of `</think>` in the response using different models to generate CoT, which indicates the frequency of rethinking. The dataset is sampled from the original one.

| **Dataset** | **Model** | **CoT Template – standard** | | | |
|---|---|---|---|---|---|
| | | 3b | 7b | plus | max |
| GSM-8k | Distill-qwen-14b | 0 | 0 | 0 | 0 |
| | Distill-qwen-32b | 0 | 0 | 0 | 0 |
| | QwQ-32B | 89 | 59 | 38 | 44 |
| MATH-500 | Distill-qwen-14b | 3 | 3 | 0 | 0 |
| | Distill-qwen-32b | 0 | 0 | 0 | 0 |
| | QwQ-32B | 171 | 112 | 94 | 89 |

By analyzing the outputs of these models, we try to figure out whether the externally provided CoTs in the prompt can help LRMs reduce redundant reasoning. Specifically, we count the number of occurrences of the `</think>` in the generated response to provide insights into how often the model starts to rethink. An example of rethinking is provided in Appendix A.

**Inference Template**

`<|im_start|>` **User:** **[Question]** `<|im_end|>`
`<|im_start|>` **Assistant:** `<|im_end|>`
`<think>` **[Generated Thought]** `</think>`

The results are shown in Table 1, and we conduct additional experiments in Appendix A to show that an insert like that is a more optimal approach to manipulate the thought than other templates. For RL-based LRMs, even when a CoT is provided between the `<think>` and `</think>` tokens, the model still generates its own thoughts in many cases. We observe that providing higher-quality (e.g., Qwen-max) CoTs can effectively reduce the presence of rethinking.

Additionally, given CoT generated by Qwen-2.5-7b as an example, on the Math-500 dataset, the average difficulty level (identified by the 'level' data of the dataset) of the problem with/without rethinking is 3.58/2.96 out of 5. This suggests that while RL-based LRMs rely on their internal judgment of sufficient reasoning, they can be influenced by the quality of external thoughts.

In contrast, distillation-based LRMs show a different behavior. These models hardly generate additional thoughts beyond the provided CoT and start the final response when encountering the `</think>` token. This indicates that distillation-based LRMs may not truly “understand” the concept of reasoning or thinking. Instead, their behavior is primarily driven by pattern-following skills learned during supervised fine-tuning. Based on these observations, the findings can be summarized as follows:

- For RL-based LRMs, these models continue generating thoughts until they internally “perceive” that sufficient reasoning has been conducted, regardless of whether the closing `</think>` token is encountered. The quality of the provided CoT can influence the extent of additional reasoning. The LRMs rethink the harder problems more frequently.

- For distillation-based LRMs, these models terminate their reasoning process immediately upon encountering the `</think>` token, irrespective of the quantity or quality of thoughts provided. This behavior reflects a reliance on pattern-matching rather than an understanding of the reasoning process.

Based on these observations, we further propose our method, ThoughtMani, to leverage this behaviors and improve reasoning efficiency in LRMs.

## 4 Pipeline of ThoughtMani

In this section, we design an inference pipeline that innovatively involves a small model to generate CoT and concatenates it at the end of the inference template. The following prompt guides the CoT generation.

**CoT Generation - ThoughtMani**

“If you are a teacher, you are listing the important key points for solving the problem and no calculation details should be included. You are not allowed to produce any final answer. Add `<STOP>` when the key points are finished. You may provide **only very high-level ideas** for solving the problem, no calculation details should be included. If you feel that you cannot solve it, output `<STOP>` and return.”

**Algorithm 1:** ThoughtMani Pipeline

```
Input: A dataset $D = \{q_1, q_2, \dots, q_n\}$
       containing problems, a CoT
       generator model $G$, a reasoning
       model $M$
Output: Final responses $\{r_1, r_2, \dots, r_n\}$
        for each problem in $D$.
$\mathcal{T}_{\text{Mani}} \leftarrow$
  <|im_start|>User : [Question] <|im_end|>
  <|im_start|>Assistant : <|im_end|>
  <think> [CoT] </think> ;
$\mathcal{T}_{\text{Ori}} \leftarrow$
  <|im_start|>User : [Question] <|im_end|>
  <|im_start|>Assistant : <|im_end|>
  <think> ;
for each problem $q_i \in D$ do
    $C_i \leftarrow G(q_i)$ // Generate CoTs
    if $C_i =$ <STOP> then
        $T_i \leftarrow \mathcal{T}_{\text{Ori}}[\text{Question} \leftarrow q_i]$
            // Format $\mathcal{T}_{\text{Ori}}$ with $q_i$
    else
        $T_i \leftarrow \mathcal{T}_{\text{Mani}}[\text{Question} \leftarrow q_i, \text{CoT} \leftarrow$
            $C_i]$ // Format $\mathcal{T}_{\text{Mani}}$
            with $q_i$ and $C_i$
    $r_i \leftarrow M(T_i)$ // Obtain final
        response from $M$
    Append $r_i$ to the output set
      $\{r_1, r_2, \dots, r_n\}$ ;
return $\{r_1, r_2, \dots, r_n\}$
```

Compared to the previous CoT generation approach, the key difference is that we prompt the model to produce a stopping identifier when encountering highly complex problems. This strategy aims to fully leverage the reasoning capabilities of LRMs for challenging scenarios while remaining efficient for simpler questions.

Specifically, if the generated CoT only contains “STOP”, we drop it and use the original inference template, which aims to prevent the incorrect CoTs from misleading the LRMs. The detailed pipeline is shown in Algorithm 1.

# 5 Experiment

## 5.1 Experimental Setup

**Datasets.** To evaluate the effectiveness of the proposed CoT-reduced reasoning process, we select four different datasets, covering reasoning ability in both math and coding. For reasoning, we select three widely used math datasets, including AIME-2024 (Maxwell-Jia, 2024), GSM-8k (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2023). For coding, we select the coding category from LiveBench (White et al., 2024). To evaluate the safety of the model response, we select the Wild-Jailbreak (Jiang et al., 2024) as the target dataset, which transforms harmful queries with randomly sampled in-the-wild jailbreak tactics including-GCG (Zou et al., 2023), AutoDAN (Liu et al., 2023), DeepInception (Li et al., 2023), etc. It contains over 2,000 adversarial jailbreak prompts.

**Metrics.** We quantify the performance from three perspectives, i.e., utility, efficiency, and safety. For utility, we extract answers via string matching for the AIME, GSM-8k, and MATH-500 datasets. Regarding the coding dataset, we follow the official guidance and report the pass@1 metric on private test cases. For efficiency, we compute the generated tokens from the reasoning model and the additional tokens produced by the CoT generators. The total cost of generation is evaluated as the sum of these two components. Since the CoTs are generated by smaller models, the cost of producing these additional tokens is significantly lower compared to the computational expense of the larger reasoning model. For safety, we utilize a widely used safety moderator, Llama-Guard-3-8B (Grattafiori et al., 2024), to evaluate the safety of model output. If the response is unsafe, the moderator will output “unsafe”, followed by the reason; otherwise, the moderator will output “safe.”

**Models.** Regarding CoT generators, we consider Qwen-series (Yang et al., 2024), including Qwen-Max, Qwen-Plus, Qwen-2.5-7B-Instruct, and Qwen-2.5-3B-Instruct. Since we need to accurately manipulate the inference template in the stated approach, we only consider local open-source LRMs. Regarding RL-based LRMs, we select QwQ-32B (Team, 2025), which is derived by RL from Qwen-2.5-32B-Instruct. Regarding distillation-based LRMs, we select Deepseek-Distillation-Qwen-2.5-14B-Instruct and its 32B version, which distills the CoT generated from DeepSeek-R1 on Qwen series (DeepSeek-AI et al., 2025). Specifically, we use a 4-bit AWQ (Lin et al., 2024) quantized version of these models to save GPU memory and utilize the vLLM (Kwon et al., 2023) framework for efficient inference. Regarding the decoding algorithm, we follow their official guideline[1] and use greedy decoding to generate the outputs, where temperature is set to 0.7 and top-p is set to 0.95. In the efficiency and utility experiment, the max output token number of the AIME-2024 dataset is set to 30,000 due to the problem complexity, while we set the max output token number to 20,000 for the remaining datasets.

**Baselines.** We take the following methods as our baselines:

- *Empty Thought* works like ThoughtMani while placing empty CoT within the thinking tokens.
- *Truncation* directly terminates the thinking process by interrupting the generation when a predefined thinking budget is met and inserting a `</think>` token to output the answer. Specifically, we cut 50% length of the original thinking process.
- *Prompt Reduction* (Ding et al., 2024) provides specific instructions like “Let’s quickly conclude the answer without showing step-by-step reasoning.” to reduce the thinking process.
- *Tokenskip* (Xia et al., 2025) first constructs a dataset where less important tokens are pruned and fine-tunes the models on the compressed dataset to enable the model to selectively skip the redundant tokens in inference. For GSM-8k and MATH-500, we first fine-tune the model on the training set and evaluate the performance on the test set, where the training dataset has 7,453 and 7,500 problems, respectively. Since AIME-2024 and Code have no official training set, we transfer the tuned model from MATH, which includes more challenging problems.
- *CoT-Valve* (Ma et al., 2025) utilizes interpolation of the LLMs’ and LRMs’ parameters to collect CoTs of varying lengths, followed by progressively fine-tuning the LRMs to compress the length of CoT. We fine-tune the model on their officially provided dataset, i.e., MixChain-Z-GSM8K (6,863 samples), and select the best

[1] https://huggingface.co/Qwen/QwQ-32B

Table 2: **Efficiency and Utility Results**: Utility is reported by Accuracy and Pass@1 for different datasets (replicated three times each). Efficiency is reported by the number of generated tokens. Full and Empty represent inference with vanilla and *EmptyThought*, respectively. Prompt represents inference with *Prompt Reduction*. For ThoughtMani, we additionally reported the number of generated CoT tokens, which represents the additional cost.

| Method | AIME-2024 | | | GSM-8k | | | MATH-500 | | | Livebench/Coding | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | Tokens | CoT | Acc | Tokens | CoT | Acc | Tokens | CoT | Pass@k | Tokens | CoT |
| | QwQ-32B | | | | | | | | | | | |
| Full | 70.0 | 13661 | - | 95.3 | 1791 | - | 88.5 | 4537 | - | 66.7 | 6840 | - |
| Empty | 40.0 | 12085 | | 95.1 | 1552 | | 80.4 | 4321 | | 64.3 | 5865 | |
| Prompt | 43.3 | 10897 | | 93.1 | 665 | | 82.2 | 3190 | | 63.5 | 6518 | |
| Truncation | 36.7 | 12508 | | 95.7 | 1624 | | 81.0 | 4938 | | 57.8 | 4128 | |
| TokenSkip | 50.0 | 11172 | | 94.4 | 536 | | 86.8 | 3225 | | 65.9 | 4269 | |
| CoT-Valve | 74.4 | 14199 | | 95.5 | 1697 | | 89.2 | 4546 | | 74.6 | 6714 | |
| ThoughtMani- 3b | 70.0 | 14329 | 11 | 95.3 | 1725 | 7 | 86.1 | 4077 | 22 | 65.6 | 6842 | 2 |
| ThoughtMani- 7b | 70.0 | 13101 | 77 | 94.0 | 1075 | 52 | 86.0 | 3526 | 56 | 62.2 | 4409 | 120 |
| ThoughtMani- Plus | 75.6 | 11400 | 209 | 93.5 | 961 | 79 | 86.7 | 2792 | 141 | 64.1 | 4461 | 137 |
| ThoughtMani- Max | 60.0 | 9607 | 568 | 93.9 | 759 | 132 | 85.6 | 2335 | 209 | 60.9 | 4209 | 183 |
| | Deepseek-Distillation-Qwen-2.5-32B-Instruct | | | | | | | | | | | |
| Full | 68.9 | 9915 | - | 88.3 | 439 | - | 84.0 | 2973 | - | 60.2 | 6777 | - |
| Empty | 43.3 | 9032 | | 89.7 | 223 | | 69.4 | 609 | | 43.2 | 737 | |
| Prompt | 50.0 | 8808 | | 89.6 | 370 | | 78.2 | 2167 | | 57.3 | 5882 | |
| Truncation | 30.0 | 4638 | | 88.8 | 267 | | 75.8 | 1760 | | 54.7 | 10103 | |
| TokenSkip | 40.0 | 3455 | | 89.4 | 423 | | 76.6 | 1567 | | 49.5 | 6084 | |
| CoT-Valve | 63.3 | 10359 | | 88.8 | 478 | | 82.1 | 2856 | | 60.2 | 6012 | |
| ThoughtMani- 3b | 62.2 | 10210 | 11 | 88.3 | 415 | 7 | 82.6 | 2526 | 22 | 59.1 | 6557 | 2 |
| ThoughtMani- 7b | 54.3 | 7985 | 77 | 86.8 | 292 | 52 | 79.4 | 2170 | 56 | 41.7 | 528 | 120 |
| ThoughtMani- Plus | 20.1 | 2076 | 209 | 87.5 | 263 | 79 | 68.3 | 554 | 141 | 45.8 | 528 | 137 |
| ThoughtMani- Max | 21.1 | 1482 | 568 | 88.7 | 267 | 132 | 67.8 | 562 | 209 | 44.5 | 465 | 183 |
| | Deepseek-Distillation-Qwen-2.5-14B-Instruct | | | | | | | | | | | |
| Full | 31.1 | 8273 | - | 87.6 | 756 | - | 65.3 | 2392 | - | 54.7 | 6871 | - |
| Empty | 30.0 | 8215 | | 75.2 | 216 | | 63.8 | 796 | | 33.6 | 657 | |
| Prompt | 33.3 | 8803 | | 88.3 | 516 | | 65.2 | 1904 | | 54.9 | 6312 | |
| Truncation | 26.7 | 5204 | | 84.2 | 214 | | 62.6 | 1627 | | 46.9 | 9245 | |
| TokenSkip | 30.0 | 8503 | | 89.3 | 314 | | 73.2 | 1356 | | 0.0 | 10750 | |
| CoT-Valve | 15.0 | 10967 | | 86.7 | 681 | | 62.9 | 2190 | | 56.2 | 6042 | |
| ThoughtMani- 3b | 19.9 | 8649 | 11 | 86.4 | 691 | 7 | 65.2 | 2080 | 22 | 53.9 | 6670 | 2 |
| ThoughtMani- 7b | 24.4 | 7952 | 77 | 85.7 | 356 | 52 | 69.2 | 1742 | 56 | 38.5 | 588 | 120 |
| ThoughtMani- Plus | 16.6 | 2209 | 209 | 88.1 | 272 | 79 | 65.4 | 600 | 141 | 39.6 | 625 | 137 |
| ThoughtMani- Max | 18.8 | 1838 | 568 | 89.6 | 281 | 132 | 64.6 | 595 | 209 | 37.0 | 523 | 183 |

model for comparison. Specifically, we choose CoT-Valve+P as the fine-tuning pattern.

### 5.2 Efficiency and Utility Performance

The main results of our experiments are shown in Table 2. *EmptyThought* can effectively reduce the tokens on the distillation-based models at the cost of performance, while showing limited effects on RL-based models. *Prompt Reduction* and *Truncation* can decrease token counts to some extent, but the reduction varies unpredictably, and the associated performance drop can be substantial. For *Tokenskip*, the performance of in-domain cases, i.e., GSM-8k and MATH-500, is competitive in both utility and efficiency, while showing limited ability to transfer to other datasets. For *CoT-Vavle*, the reproduced performance shows increased utility while the compression ability is usually.

Generally, ThoughtMani shows competitive performance. For the RL-based model (QwQ), ThoughtMani with four different CoT generators reduces the response length by 1%, 18%, 26%, and 37% with 1.5%, 2.8%, 0.8%, and 7.2% performance drop for the average on four different datasets. For the distillation-based models, ThoughtMani with four different CoT generators reduces the response length by 2%, 45%, 82%, and 86% with a relatively higher 4.5%, 11.5%, 20.4%, and 18.2% performance drop for the average on four different datasets. Since smaller CoT generators may refuse to provide CoT in many hard cases and return empty thoughts, this makes the average CoT length relatively short.

**Larger CoT Generators Are Not Better.** Across various experiments, we observe that using stronger CoT generators, such as Qwen-Max, can negatively impact the performance of our ThoughtMani inference framework. Larger models generate more specific and detailed CoT processes. While these detailed thoughts may appear helpful, they often contain hallucinations or reasoning paths that are misaligned with the LRM's expectations, leading to suboptimal performance.

In contrast, as discussed in Section 4, we prompt the CoT generator to return an identifier (e.g., `<STOP>`) when the problem is too complex to provide meaningful insights. Due to their relatively limited reasoning capabilities, smaller models tend to reject generating detailed thoughts and leave the heavy burden to the LRM. Generally, we find Qwen-2.5-7b-Instruct is the optimal or suboptimal solution in nearly all scenarios.

**RL-based LRMs Benefits More.** Our findings indicate that RL-based LRMs benefit more significantly from external CoTs compared to distillation-based LRMs. This is because RL-based LRMs are trained to dynamically evaluate the sufficiency of their reasoning process with rewards. Given the observation in Section 3, the RL-based model has the capability to "rethink", thus dynamically deciding when to accept the external CoT and when to rethink to support and revise the provided insufficient information. As a result, RL-based LRMs are less sensitive to the quality of provided while preserving the accuracy and utility.

On the other hand, distillation-based LRMs, which rely more heavily on pattern-matching during training, tend to terminate their reasoning process immediately upon encountering the `</think>` token, regardless of the quality or completeness of the provided CoT. This rigidity limits their ability to fully utilize external thoughts, resulting in less pronounced gains compared to RL-based models.

### 5.3 Safety Performance

Despite the improved efficiency, it remains unclear whether the efficiency-oriented methods will influence the safety performance. To fill this gap, we evaluate the robustness of ThoughtMani, along with other baselines, against jailbreak prompts. We generate the CoT and response using the pipeline stated in Section 4 and set the model length to 10,000. The experiment compares the performance across different baselines and ThoughtMani with different CoT generators. The results are shown in Table 3. Benefit from the CoT generated by well-aligned non-reasoning models, we find that ThoughtMani improves the model safety by 10% on average, while the fine-tuning-based methods show an average 7% safety drop.

Table 3: **Safety Results:** Safety is reported based on the judgment from Llama-Guard-3-8B, which assigns scores within a range of 100. Higher values indicate safer responses.

| | QwQ | Qwen-32B-distill | Qwen-14B-distill |
|---|---|---|---|
| Full | 66.33 | 65.48 | 64.80 |
| Empty | 95.56 | 98.18 | 95.97 |
| Prompt | 79.35 | 66.86 | 72.70 |
| Truncation | 95.79 | 71.67 | 73.85 |
| TokenSkip | 73.30 | 60.07 | 61.40 |
| CoT-Valve | 64.40 | 34.67 | 72.26 |
| ThoughtMani-3b | 70.41 | 64.98 | 65.66 |
| ThoughtMani-7b | 66.47 | 60.68 | 65.11 |
| ThoughtMani-Plus | 77.92 | 74.34 | 75.07 |
| ThoughtMani-Max | 74.03 | 71.45 | 73.48 |

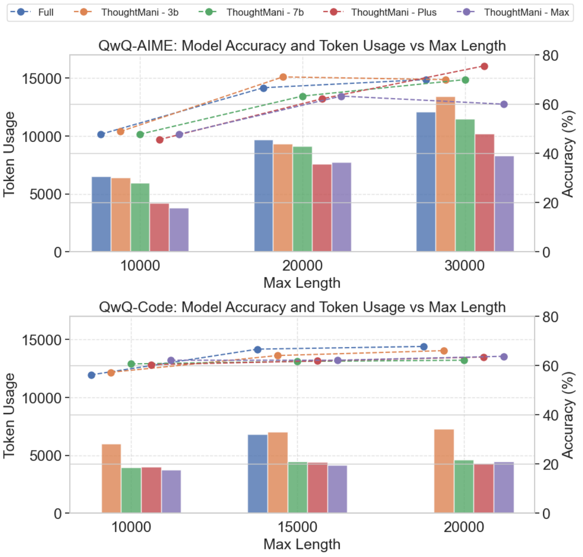

Figure 1: **Relation between Model Performance and Token Budgets:** Bar plot represents the token consumption and line plot represents the model utility (Accuracy or Pass@1)

Additionally, an interesting observation is that empty or truncated thoughts can effectively improve safety performance, while the mechanism remains to be further discussed. These findings uncover an important yet largely ignored aspect of the current study.

### 5.4 Dataset-specific CoTs

For the Code dataset, the task involves coding rather than mathematical reasoning, and we evaluate the impact of using task-specific system

Table 4: **Performance of Utilizing CoTs generated from Different Prompts** The results are reported using Pass@1 and number of generated tokens. Normal represents using original COTs, while Specific represents using the task-specific CoTs.

| Method | QwQ-32B | | | | Deepseek-Distillation-Qwen-2.5-32B-Instruct | | | | Deepseek-Distillation-Qwen-2.5-14B-Instruct | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Normal | | Specific | | Normal | | Specific | | Normal | | Specific | |
| | Pass@1 | Tokens | Pass@1 | Tokens | Pass@1 | Tokens | Pass@1 | Tokens | Pass@1 | Tokens | Pass@1 | Tokens |
| ThoughtMani- 3b | 65.6 | 6842 | 64.1 | 7009 | 59.1 | 6557 | 55.7 | 6163 | 53.9 | 6670 | 54.4 | 6535 |
| ThoughtMani- 7b | 62.3 | 4409 | 61.7 | 4485 | 41.7 | 528 | 42.4 | 627 | 38.5 | 588 | 35.9 | 582 |
| ThoughtMani- Plus | 64.1 | 4461 | 61.9 | 4408 | 45.8 | 528 | 45.3 | 539 | 39.8 | 625 | 41.9 | 536 |
| ThoughtMani- Max | 60.9 | 4209 | 62.2 | 4181 | 44.5 | 465 | 41.1 | 577 | 36.9 | 523 | 38.8 | 572 |

prompts for CoT generators. In the common setting, we utilize the general prompt described in Section 4, which is designed to generate high-level reasoning steps applicable to a wide range of tasks. For the code generation task, we modify the prompt slightly to emphasize the coding scenario, while still adhering to the principle of providing only high-level ideas without delving into implementation details. Prompts and the comparison of the generated CoTs are shown in Appendix B.

The results, shown in Table 4, reveal no significant performance difference between the two settings. This finding highlights the one-for-all property of our approach: the general CoT generation framework is robust enough to handle diverse tasks without requiring task-specific adjustments. In other words, ThoughtMani demonstrates strong adaptability across domains, eliminating the need for choosing different CoT templates for different types of problems.

### 5.5 Performance under Different Token Budgets

For more challenging datasets, such as AIME and Code, the inference process of RL-based LRMs typically demands a significant number of tokens to achieve high-quality reasoning. To systematically study the impact of token budgets on performance, we adjust the max_model_length parameter during inference and evaluate the corresponding outcomes. Specifically, for the AIME dataset, we set the maximum token length to 10,000, 20,000, and 30,000, while for the Code dataset, we use 10,000, 15,000, and 20,000. The results are presented in Figure 1. Generally, as the max_model_length increases, the accuracy improves while maintaining inference efficiency.

An interesting observation is that the performance gap between the baseline and ThoughtMani is more pronounced when the token budget is limited. One possible explanation is that the externally generated CoTs, especially for very hard problems, may include hallucinations or insufficient information. In such cases, the LRM compensates by generating additional thoughts to supplement the provided CoTs, showcasing one limitation of ThoughtMani. In contrast, for simpler datasets like GSM-8k, where most questions can be resolved within fewer than 1,000 tokens, this performance gap does not exist. These findings underscore the importance of balancing token budgets with problem complexity. While ThoughtMani demonstrates benefits in reducing computational overhead, it shows effectiveness more evidently in scenarios where the token budget is sufficient.

## 6 Conclusion

In this paper, we propose ThoughtMani, a simple yet efficient inference pipeline, and reveal an important characteristic of LRM behaviors. Through extensive experiments, we demonstrate that ThoughtMani can significantly reduce computational costs while keeping the utility. By providing a practical solution to improve efficiency without compromising utility or safety, ThoughtMani makes LRMs more accessible for real-world, resource-constrained applications.

**Implications.** Our findings highlight significant implications for LRMs. By understanding the distinct behaviors of RL-trained and distillation-trained LRMs, we can make better-informed deployment decisions. Since model vendors typically serve models of different sizes simultaneously, ThoughtMani offers a practical solution to reduce computational costs while maintaining accuracy, making LRMs more efficient and accessible for real-world, resource-constrained scenarios.

## Limitations

**Dataset and Models.** Despite extensive experiments having been conducted, we mainly cover the performance of ThoughtMani on mathematical and code reasoning tasks, leaving other important aspects such as instruction-following and function-calling untested. Additionally, since we cannot manipulate the inference template of API calling, we only employ ThoughtMani for mainstream local reasoning models. Further studies on different tasks and APIs are encouraged.

**Manipulation for Other Tasks.** In this paper, we propose ThoughtMani primarily for achieving efficient inference. However, the insights gained from our study open up several other meaningful research directions. One direct application is to insert malicious or misleading thoughts into the reasoning process to manipulate model behavior. Another promising line of research is to investigate the inherent mechanisms that govern when a model stops thinking or how to detect whether a model is actively engaged in reasoning. Understanding these dynamics could lead to more precise control over the reasoning process, enabling better alignment with desired outcomes and further optimizing computational efficiency.

## References

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2024. Do not think that much for 2+3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

Jeffrey Cheng and Benjamin Van Durme. 2024. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*.

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*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 81 others. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

Yuntian Deng, Yejin Choi, and Stuart Shieber. 2024. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*.

Mengru Ding, Hanmeng Liu, Zhizhang Fu, Jian Song, Wenbo Xie, and Yue Zhang. 2024. Break the chain: Large language models can be shortcut reasoners. *arXiv preprint arXiv:2406.06580*.

Yichen Gong, Delong Ran, Xinlei He, Tianshuo Cong, Anyu Wang, and Xiaoyun Wang. Safety misalignment against large language models.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, and 79 others. 2024. Gpt-4o system card. *CoRR*, abs/2410.21276.

Liwei Jiang, Kavel Rao, Seungju Han, Allyson Ettinger, Faeze Brahman, Sachin Kumar, Niloofar Mireshghallah, Ximing Lu, Maarten Sap, Yejin Choi, and Nouha Dziri. 2024. Wildteaming at scale: From in-the-wild jailbreaks to (adversarially) safer language models. *Preprint*, arXiv:2406.18510.

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*.

Xuan Li, Zhanke Zhou, Jianing Zhu, Jiangchao Yao, Tongliang Liu, and Bo Han. 2023. Deepinception: Hypnotize large language model to be jailbreaker. *arXiv preprint arXiv:2311.03191*.

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*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for llm compression and acceleration. *Preprint*, arXiv:2306.00978.

Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng Zhang. 2024a. Can language models learn to skip steps? *arXiv preprint arXiv:2411.01855*.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2023. Autodan: Generating stealthy jailbreak prompts on aligned large language models. *arXiv preprint arXiv:2310.04451*.

Yule Liu, Zhen Sun, Xinlei He, and Xinyi Huang. 2024b. Quantized delta weight is safety keeper. *arXiv preprint arXiv:2411.19530*.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. Cotvalve: Length-compressible chain-of-thought tuning. *arXiv preprint arXiv:2502.09601*.

Maxwell-Jia. 2024. AIME 2024 Dataset. `https://huggingface.co/datasets/Maxwell-Jia/AIME_2024`.

OpenAI. 2025a. Introducing openai o1. `https://openai.com/o1/`. Accessed: 01-April-2025.

OpenAI. 2025b. Openai o3-mini. `https://openai.com/index/openai-o3-mini/`. Accessed: 01-April-2025.

Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas Back. 2024. Reasoning with large language models, a survey. *arXiv preprint arXiv:2407.11511*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Hanjie Chen, Xia Hu, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

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*.

M.-A-P. Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Kang Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, Kaixin Deng, Shian Jia, Sichao Jiang, Yiyan Liao, Rui Li, Qinrui Li, Sirun Li, and 77 others. 2025b. Supergpqa: Scaling LLM evaluation across 285 graduate disciplines. *CoRR*, abs/2502.14739.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, and 1 others. 2024. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, and 1 others. 2025. Towards large reasoning models: A survey of reinforced reasoning with large language models. *arXiv preprint arXiv:2501.09686*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li. 2024. Jailbreak attacks and defenses against large language models: A survey. *arXiv preprint arXiv:2407.04295*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

## A Supplement for Thought Manipulation

**Prompt for CoT Generation.** The prompt for generating the CoT is shown as follows:

**CoT Generation - Standard**

"If you are a teacher, you are listing the important key points for solving the problem and no calculation details should be included. You are not allowed to produce any final answer. Add <STOP> when the key points are finished. You may provide **only very high-level ideas** for solving the problem, no calculation details should be included."

**Code Example for ThoughtMani Template.**

```python
def process_example(question, external_thought):
    messages = [
        {"role": "user", "content": f"Question: {question}"},
        {"role": "assistant", "content": ""}
    ]
    template = tokenizer.apply_chat_template(
        messages,
        add_generation_prompt=True,
        tokenize=False
    )
    if not template.endswith(('<think>', '<think>\n')):
        template += '</think>'
    template += f'{external_thought}\n\n</think>'
    return template
```

**Ablation Study for Optimal Manipulation.** Additionally, we conduct an ablation study to show the effectiveness of placing CoTs, enclosed by `<think>` and `</think>`, at the end of the chat template. Specifically, we design the following two cases and represent them using chat templates, where the dashed line represents the start position for model generation.

For the first case, we consider placing the CoTs enclosed by thinking tokens within the template, i.e., before `<|im_end|>`. The experiment results are shown in Table 5.

**Inference Template - Case 1 (within chat)**

`<|im_start|>` **User:** **[Question]**
`<|im_end|>`
`<|im_start|>` **Assistant:**
`<think>`
**[Generated Thought]**
`</think>`
`<|im_end|>`

Table 5: Number of occurrences of `</think>` in the response using different models to generate CoT, which indicates the frequency of rethinking. The dataset is sampled from the original one.

| Dataset | Model | CoT Template – within chat | | | |
|---|---|---|---|---|---|
| | | 3b | 7b | plus | max |
| GSM-8k | Distill-qwen-14b | 86 | 88 | 85 | 86 |
| | Distill-qwen-32b | 0 | 0 | 0 | 0 |
| | QwQ-32B | 98 | 97 | 86 | 89 |
| MATH-500 | Distill-qwen-14b | 164 | 160 | 172 | 155 |
| | Distill-qwen-32b | 9 | 1 | 1 | 0 |
| | QwQ-32B | 193 | 183 | 184 | 163 |

For the second case, we consider directly placing the CoTs enclosed after the template, i.e., after `<|im_end|>` but without thinking tokens. The experiment results are shown in Table 6.

**Inference Template - Case 2 (no thinking token)**

`<|im_start|>` **User:** **[Question]**
`<|im_end|>`
`<|im_start|>` **Assistant:** `<|im_end|>`
**[Generated Thought]**

**Rethinking Example.** We provide an example for

Table 6: Number of occurrences of `</think>` in the response using different models to generate CoT, which indicates the frequency of rethinking. The dataset is sampled from the original one.

| Dataset | Model | CoT Template – no thinking | | | |
|---|---|---|---|---|---|
| | | 3b | 7b | plus | max |
| GSM-8k | Distill-qwen-14b | 86 | 87 | 91 | 81 |
| | Distill-qwen-32b | 0 | 0 | 0 | 0 |
| | QwQ-32B | 97 | 81 | 83 | 55 |
| MATH-500 | Distill-qwen-14b | 165 | 161 | 172 | 163 |
| | Distill-qwen-32b | 12 | 2 | 0 | 0 |
| | QwQ-32B | 195 | 193 | 180 | 145 |

rethinking as follows, which is referred to as re-generating the thinking process, although external thoughts have been provided.

**A Rethinking Example**

<|im_start|> **User:** A tower is made out of 4 blue blocks, twice as many yellow blocks, and an unknown number of red blocks. If there are 32 blocks in the tower in total, how many red blocks are there? <|im_end|>
<|im_start|> **Assistant:** <|im_end|>
<think>
**[External CoT]**
- Identify the number of blue blocks, which is given as 4.
-Determine the number.
...
</think>

**[Inner Thought]**
Okay, let's see here. The problem is about ...
...
**Final Answer**The number of red blocks is boxed{20}.
**[Final Response]**
The tower is made up of 4 blue blocks, ...,Thus, the number of red blocks is ( boxed{20}).",

## B Supplement for Dataset-specific Prompt

**Prompt for Code Reasoning.** The prompt for generating code-specific CoT is shown as follows:

**CoT Generation**

"If you are a teacher, you are listing the important key points for writing code. You are not allowed to output any Final Code. Add <STOP> when the key points are finished. Note that you may provide **only very high-level ideas** for solving the problem and writing code. If you feel that you cannot solve it, output <STOP> and return. "

**Difference in Generated CoTs.** We provide two examples to demonstrate the difference between the CoTs generated by general and code-specific prompts. The code-specific prompts generally provide more details and reminders for the code-writing task.

**Example of General CoT**

"To solve this problem, follow these key points: newline 1. Understand that the goal is to remove elements smaller than 'k' until all remaining elements are greater than or equal to 'k'.
2. Count how many elements in the array are less than 'k'.
3. The count of such elements will be the minimum number of operations required.
<STOP>"

**Example of Code-specific CoT**

" 1. Initialize a counter to zero, which will keep track of the number of operations.
2. Iterate through the array 'nums' and count how many elements are less than 'k'.
3. Each element less than 'k' represents one operation needed to remove it.
4. Return the total count as the minimum number of operations required.
5. If all elements are already greater than or equal to 'k', return 0 operations.
<STOP>"