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 on various tasks. However, LRMs often suffer from an "overthinking" problem, where the model generates excessively redundant reasoning steps with limited performance gains. In this work, we empirically 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 this finding, we propose a simple yet efficient pipeline, ThoughtMani, to enable LRMs to bypass unnecessary intermediate steps, thereby significantly reducing computational costs. We conduct extensive experiments to evaluate the utility and efficiency of ThoughtMani. For instance, when applied to QwQ-32B on the LiveBench/Code dataset, ThoughtMani keeps the original performance while reducing output token counts by approximately 30%, with minimal overhead introduced by the CoT generator. Furthermore, we identify two suboptimal modes, blindly following flawed external thoughts and unnecessary rethinking, and show that simple mitigations, such as difficulty-aware fallbacks, can further improve performance. Overall, ThoughtMani offers a practical, general, and efficient way to optimize LRM inference, making powerful reasoning models more accessible and scalable 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 accuracy.

Despite the improved performance, scaling up CoT often requires exponentially larger computational resources (Sui et al., 2025). Models like QwQ consume 5 to 10 times more tokens to reach conclusions compared to standard approaches (Chen et al., 2024). 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.

In this work, we first investigate a natural yet underexplored question arising from the inefficiencies of the inherent thinking process in LRMs: **Can externally provided thoughts substitute and bypass the model's internal reasoning?** To answer this, we design a thought-replacing strategy that simulates the standard inference process, but with the internal CoT replaced by an external one generated by a smaller model. This external CoT is inserted between the <think> and </think> tokens in the input prompt, allowing us to observe whether the LRM accepts the provided reasoning and skips redundant thinking.

Our analysis shows that LRMs can indeed skip their internal reasoning process and proceed directly to answer generation when presented with external thoughts. Notably, RL-based LRMs exhibit a nuanced behavior: while they often accept the external CoT, they sometimes resume thinking, indicated by the reappearance of the </think> token

in their output—a behavior we term "rethinking." In contrast, distillation-based LRMs almost always terminate reasoning upon encountering `</think>`, leading to near-complete skipping. Further hidden-state-level analysis, based on the logit dynamics of the `</think>` token, confirms that this skipping behavior is not merely a superficial pattern match but reflects deeper internal state transitions. The rank of `</think>` in the output distribution drops sharply when the model decides to exit reasoning, suggesting a genuine shift in the model's internal decision-making process.

Crucially, we find that skipping internal reasoning leads to efficiency gains, often reducing output token counts by over 50%, while incurring only moderate performance drops (under 10%). Building on these insights, we propose ThoughtMani, a training-free inference pipeline that leverages external CoTs to reduce computation.

We comprehensively evaluate ten different compression techniques across three LRMs on four diverse datasets. For example, when using Qwen-2.5-7B-Instruct as the CoT generator for QwQ-32B on the GSM-8k dataset, ThoughtMani reduces output token count by approximately 40% (from 1,791 to 1,075 tokens), with an average overhead of only 52 additional tokens from the external thoughts. We also identify and quantify the frequency of two common suboptimal behaviors that affect efficiency. By applying a simple difficulty-aware mitigation, excluding external thoughts for the most challenging problems, we improve accuracy by up to 8% while still achieving substantial token savings compared to the full reasoning baseline. Further ablation studies confirm the robustness and practical effectiveness of ThoughtMani across different settings and model types.

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 introduce ThoughtMani, a training-free inference pipeline that reduces redundant reasoning by injecting thoughts generated from smaller models, achieving substantial computational savings without compromising performance or needing additional training.

Table 1: **Statistics of `</think>` Token Rank**: Mean position (± std) of the `</think>` token in different response stages. Only samples exhibiting rethinking (i.e., where `</think>` reappears) are included. Columns correspond to Question, External Chain-of-Thought (CoT), Rethinking (Re.), and Final Answer stages.

Ex.		Skip?	Num.	Question	CoT	Re.	Answer
QwQ	3B	Yes	48	$3278.0_{19250.0}$	$132.1_{168.9}$	-	$12.7_{23.6}$
		No	38	$2715.0_{17640.0}$	$123.7_{243.5}$	$11.9_{78.8}$	$4.7_{30.3}$
	7B	Yes	93	$2950.0_{18402.0}$	$66.8_{142.4}$	-	$15.3_{30.4}$
		No	130	$2690.0_{17554.0}$	$76.5_{175.0}$	$12.1_{28.1}$	$3.6_{22.1}$
	Plus	Yes	281	$2968.3_{18456.8}$	$69.6_{125.2}$	-	$18.5_{35.0}$
		No	219	$2592.7_{17240.0}$	$69.6_{120.5}$	$13.4_{57.6}$	$3.1_{5.5}$
	Max	Yes	219	$2899.5_{18218.0}$	$46.0_{84.1}$	-	$18.4_{69.9}$
		No	281	$2686.0_{17561.0}$	$47.6_{78.1}$	$13.0_{36.0}$	$6.1_{78.9}$
Deepseek	3B	Yes	86	$5259.0_{20324.0}$	$97.8_{162.1}$	-	$15.6_{29.0}$
	7B	Yes	223	$5261.3_{20330.9}$	$58.8_{133.4}$	-	$13.2_{25.7}$
	Plus	Yes	500	$5263.5_{20339.5}$	$47.3_{94.6}$	-	$12.4_{23.7}$
	Max	Yes	500	$5262.0_{20329.0}$	$32.2_{54.4}$	-	$11.3_{21.2}$

- Extensive experiments demonstrate the superiority of ThoughtMani in both utility and efficiency, with consistent improvements across diverse datasets and tasks, underscoring its practical value in real-world scenarios.

2 Effectiveness of External Thoughts

In this section, we try to answer the question: **Can externally provided thoughts substitute and bypass the generation of the original ones?** First, we investigate the effect of placing external thoughts in positions typically reserved for model-generated thoughts, revealing distinct behaviors between RL-based and distillation-based LRMs in terms of skipping the original reasoning process. Second, we perform a fine-grained analysis of the remaining tokens to understand how external thoughts influence the reasoning process and token allocation. Finally, based on these insights, we introduce ThoughtMani, a training-free framework designed to enable efficient inference by leveraging external thoughts.

2.1 Replacing the Original Thought

We simulate the thinking process by placing external thoughts in positions typically reserved for model-generated thoughts. To generate the external thoughts, we select 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. This selection enables a fair comparison of the generated thoughts, where the larger model typ-

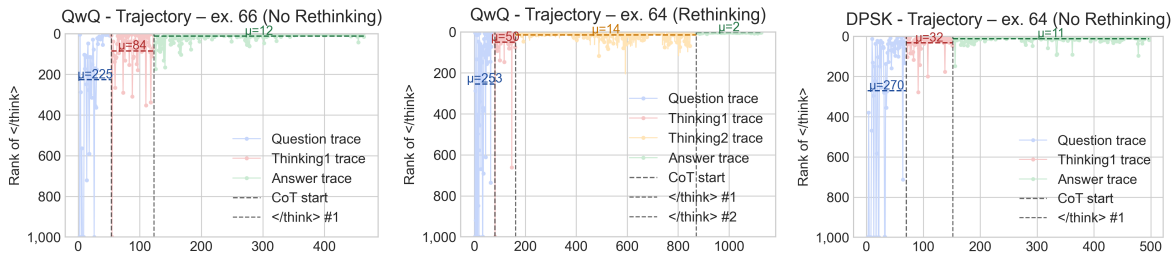

Figure 1: **Rank Trajectory of `</think>`**: For a prompt and answer pair, we identify the question, thinking, and answer traces by corresponding special tokens, i.e., `<|im_end|>` and `</think>`. The CoT generator is Qwen-Plus, and the inference models are QwQ and Deepseek-Distillation-Qwen-2.5-32B-Instruct (DPSK), respectively. Specifically, each trace is assigned to a different color and marked with the mean rank value μ and the rank is cut off by 1000 for clear demonstration. Sample content of these trajectories is shown in Appendix A.1.

ically exhibits stronger capability. Specifically, the generation process is prompted to focus solely on high-level reasoning steps without delving into detailed calculations or producing final answers. The prompt is shown as follows:

Thought Generation

"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."

The generated thoughts are then enclosed within `<think>` and `</think>` tokens and inserted into a standardized chat template, as shown below:

Inference Template

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

This placement simulates the normal inference process, except that the thought is now provided by an external source rather than being internally generated by the LRM. The template is then used to invoke LRMs such as QwQ and Deepseek-Distillation-Qwen-2.5-32b-instruct. By analyzing the model's response, we evaluate whether the inherent thinking process is bypassed and how effectively the external thoughts are utilized.

2.2 Skipped or Not

In this part, we first empirically examine the skip of inherent thinking by the token-level feature in the prompts, and reveal distinct behaviors between RL-based and distillation-based LRMs. Second, we perform a hidden-state-level analysis to uncover additional distinguishing features that further support the identification of thought skipping. The sample outputs are provided in Appendix A.1.

Token-Level Analysis. Since the external thoughts are included in the input prompt, we determine that the model skips its internal reasoning if the token `</think>` does not appear in the generated response. If `</think>` does appear, we consider the thinking process not skipped—meaning the LRM generates additional internal reasoning despite the provided external thought, which we refer to as "rethinking"[1]. This empirical analysis is conducted on the MATH-500 dataset, a widely used benchmark for evaluating reasoning performance.

The statistics of occurrence are shown in the **Number** column of Table 1. For RL-based LRM (QwQ), the model can not skip its thinking process in many cases. We observe that better external thoughts (generated by a larger model such as Qwen-max) can effectively improve the chances of skipping thoughts. Additionally, harder problems are more likely to induce rethinking. Given CoT generated by Qwen-2.5-7b as an example, our experiments show that 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.

In contrast, distillation-based LRMs show a different behavior. These models skip the thinking

[1]We use "not skipped" and "rethinking" interchangeably to describe this behavior.

process in all cases 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 (Wulfmeier et al., 2024; Swamy et al., 2025). Regarding the position where we place external thoughts, we try different approaches,e.g., inserting directly and within the chat template in Appendix A.2.

Hidden-State-Level Analysis. To reinforce that the findings are linked to the model's inner perception rather than being a mere formatting issue, we analyze the generation dynamics of the </think> token from the logit perspective. Specifically, we compute the rank of the probability assigned to the </think> token at each decoding position. Let V denote the vocabulary with size $|V|$, and let $\mathbf{z}_t \in \mathbb{R}^{|V|}$ be the logit vector at time step t. Let $\text{idx}_{\texttt{</think>}}$ be the index of the </think> token in the vocabulary. The rank of the </think> token at position t is computed as:

$$\text{Rank}_t = \left| \left\{ w' \in V \;\middle|\; \exp(z_t^{(w')}) > \exp\left(z_t^{(\text{idx}_{\texttt{</think>}})}\right) \right\} \right| + 1 \tag{1}$$

This rank reflects how strongly the model prioritizes emitting </think> at position t, with lower ranks indicating higher relative probability. We apply this computation across the full generation sequence and segment the input-output trace using special tokens: <|im_end|> marks the end of the user input, and </think> marks the end of a reasoning segment. This allows us to identify and analyze three distinct phases: (1) the question phase, (2) the thinking phase (including external CoT and any rethinking), and (3) the final answer generation phase.

The sample-level results are shown in Figure 1, and the coarse-level results are shown in Table 1. We observe that during the thinking phase, the rank of </think> remains relatively high (i.e., low priority), indicating the model does not intend to terminate reasoning. As the model transitions toward the answer phase, the rank of </think> drops significantly, indicating a sharp increase in the likelihood of ending the thought process. These findings suggest the skip of thought is rooted in the model's learned internal dynamics, not merely surface-level pattern matching.

Table 2: **Efficacy Analysis**: We report average reasoning tokens (#Samples), steps (#Steps), and double-checks (#Checks) on MATH-500. Only cases with non-empty thoughts from the CoT generator are included.

	Ex.	Skip?	#Samples	Acc(%)	#Token	#Steps	#Check
QwQ	Baseline	-	500	88.47	4537	7.51	9.08
	3B	Yes	38	64.79	1775	4.84	-
		No	48	89.51	3541	7.46	9.90
	7B	Yes	130	77.71	663	6.52	-
		No	93	89.50	3269	7.32	8.31
	Plus	Yes	219	84.47	1212	6.17	-
		No	281	89.68	4003	7.53	6.94
	Max	Yes	281	82.92	1041	6.39	-
		No	219	87.67	4074	7.86	8.36
DPSK-14B	Baseline	-	500	65.80	2458	5.45	4.46
	3B	Yes	86	63.33	1052	6.19	-
	7B	Yes	223	64.08	715	7.44	-
	Plus	Yes	500	67.20	595	7.39	-
	Max	Yes	500	66.06	635	7.35	-

2.3 Efficacy of External Thoughts

Given the feasibility of skipping thinking, we conduct a fine-grained analysis to show whether LRMs can benefit from skipping the thoughts, especially on the efficiency aspect.

Efficacy Metrics. Based on the previous findings, we first categorize the questions based on whether the thinking process is skipped. We evaluate accuracy and token consumption to assess overall performance. Additionally, we count the number of reasoning steps and identify double-check conditions (determined by special tokens like "Hmm", "Wait", and "Alternatively"). Detailed calculations of these metrics are shown in Appendix A.3.

Efficacy Analysis. The results for QwQ and Deepseek-Distillation-Qwen-2.5-14B-Instruct on MATH-500 are presented in Table 2. We focus only on cases where the thought generator provides non-empty reasoning—i.e., it does not output <STOP>—which explains why the number of samples for smaller generators (e.g., 3B/7B) is not 500.

We observe significant token reduction when internal thinking is skipped. In these cases, both RL-based and distillation-based LRMs proceed directly to answer generation, eliminating two major sources of redundancy: (1) intermediate reasoning steps that would otherwise be generated internally, and (2) double-checking behaviors such as self-corrections or reconsiderations (e.g., "Hmm", "Wait", "Alternatively"). This is clearly reflected in the sharp declines in the # Steps and # Check

columns of Table 2. When the model does not skip thinking, some token savings still occur, but they are substantially smaller due to the generation of additional internal reasoning.

On average, accuracy in skipped cases is slightly lower than in non-skipped cases, suggesting that some external CoTs may be unreliable or misaligned with the LRM's reasoning style. Nevertheless, this degradation is generally acceptable: for Plus and Max generators, accuracy drops by only around 5%, while token usage is cut by more than half. The main exception involves the 3B and 7B generators on the RL-based model, where performance degradation is more pronounced. However, these cases are relatively rare (38 and 130 out of 500), so their overall impact remains limited. Further error analysis is provided in Section 3.3.

2.4 Pipeline of ThoughtMani

Based on these observations, we further propose our method, ThoughtMani, to leverage these behaviors and improve reasoning efficiency in LRMs. Specifically, when the CoT generator outputs only the placeholder <STOP>, indicating that it cannot provide useful reasoning, we discard the empty CoT and fall back to the original inference template. This ensures that the LRM retains full control over the reasoning process for challenging problems, avoiding potential degradation from incorrect or misleading external thoughts. The detailed pipeline is shown in Algorithm 1.

3 Experiment

3.1 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 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).

Metrics. We quantify the performance from three perspectives, i.e., utility, efficiency. 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 that of the larger reasoning model.

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). More settings are shown in Appendix B.1.

Baselines. We take the following methods as our baselines, and more details are in Appendix B.1:

- *Nothink* (Ma et al., 2025a) places "I have finished the thoughts" within the thinking tokens, as ThoughtMani does.
- *Truncation* directly terminates the thinking process when a predefined thinking budget is met and inserts a </think> token to output the answer. Specifically, we first generate the full thinking and cut 50% of the length.
- *Prompt Reduction* (Ding et al., 2024) provides 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.
- *CoT-Valve* (Ma et al., 2025b) 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.

3.2 Overall Efficiency and Utility

The main results of our experiments are shown in Table 3. *Nothink* can effectively reduce the tokens on the distillation-based models at the cost

Table 3: **Overall Efficiency and Utility**: Accuracy (Acc) and token counts (Tokens) are reported. For ThoughtMani, the number in parentheses after Tokens indicates additional CoT tokens.

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	-
Nothink	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	
Ours - 3b	70.0	14329	11	95.3	1725	7	86.1	4077	22	65.6	6842	2
Ours - 7b	70.0	13101	77	94.0	1075	52	86.0	3526	56	62.2	4409	120
Ours - Plus	75.6	11400	209	93.5	961	79	86.7	2792	141	64.1	4461	137
Ours - 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	-
Nothink	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	
Ours - 3b	62.2	10210	11	88.3	415	7	82.6	2526	22	59.1	6557	2
Ours - 7b	54.3	7985	77	86.8	292	52	79.4	2170	56	41.7	528	120
Ours - Plus	20.1	2076	209	87.5	263	79	68.3	554	141	45.8	528	137
Ours - 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	-
Nothink	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	
Ours - 3b	19.9	8649	11	86.4	691	7	65.2	2080	22	53.9	6670	2
Ours - 7b	24.4	7952	77	85.7	356	52	69.2	1742	56	38.5	588	120
Ours - Plus	16.6	2209	209	88.1	272	79	65.4	600	141	39.6	625	137
Ours - Max	18.8	1838	568	89.6	281	132	64.6	595	209	37.0	523	183

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 decreased.

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.

3.3 Suboptimal Mode Analysis

We analyze the outputs of ThoughtMani and identify two primary suboptimal modes that affect accuracy and efficiency. First, when the external thought contains flawed reasoning, such as an incorrect final inference despite mostly sound logic, the LRM may blindly follow this erroneous path, resulting in a wrong final answer (Figure 2, left). Second, the LRM may reject the external reasoning, generating new thoughts or performing redundant checks (e.g., “Hmm”, “Wait”), which increases token usage even when the final answer is correct (Figure 2, right). To systematically identify these error types, we prompt a strong LLM to judge

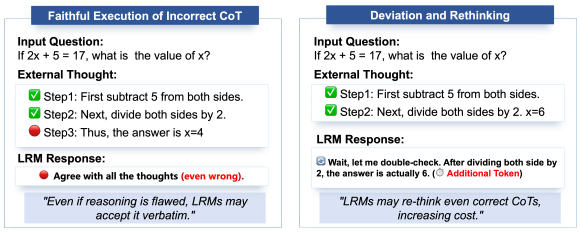

Figure 2: **Two Suboptimal Modes**: (Left) The LRM faithfully follows an incorrect external CoT, resulting in a wrong answer. (Right) The LRM rejects the external reasoning and generates additional thoughts, leading to high token consumption despite eventual correctness.

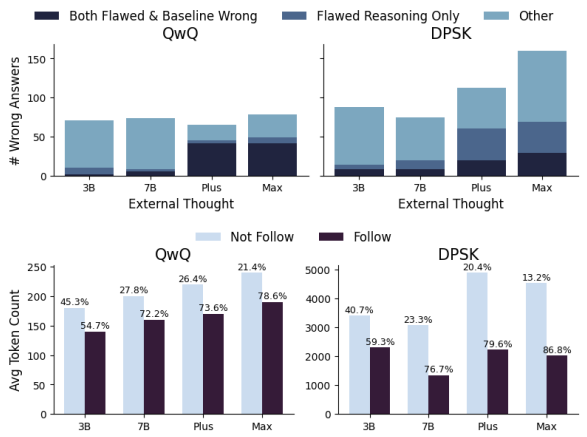

Figure 3: **Suboptimal Mode Analysis** (Upper) Visualization of how flawed reasoning impacts model performance, illustrating cases of degradation (baseline correct but misled by flawed thought), consistently wrong predictions (baseline incorrect despite flawed reasoning), and correct reasoning with incorrect final answers. (Lower) Bar plots depict average token counts for outputs that adhere to versus deviate from the provided reasoning, with numerical values indicating the percentage occurrence of each case.

whether the generated response contains reasoning flaws or diverges from the external CoT. The prompts and representative cases are included in Appendix B.2.

On the wrongly answered subset of MATH-500 (Figure 3, Upper), we further analyze the first suboptimal type by splitting it into two subcases: whether the problem can be solved correctly under vanilla inference (without external thoughts). When the original model also fails on a problem, the impact of flawed external reasoning is limited, since performance is already near the ceiling. The more harmful case occurs when a problem that could have been solved correctly under standard inference becomes erroneous due to misleading external thoughts. This degradation is particularly evident in distillation-based models, which lack the ability to re-think and are thus more vulnerable to faulty CoTs.

For the second suboptimal type (Figure 3, Lower), divergence from external reasoning, token consumption increases due to additional internal reasoning steps. While this reduces efficiency, the "unfollow" behavior often helps the model correct flawed or incomplete external thoughts, thereby improving robustness and accuracy. As such, this mode is less harmful overall and can even be beneficial in maintaining performance when external CoTs are unreliable.

Difficulty-Aware Mitigation. We focus on the harmful scenario in the first error mode—where flawed external reasoning leads the model to make mistakes on problems it could otherwise solve correctly. This issue is particularly pronounced on difficult queries, where larger CoT generators attempt to reason but produce incorrect or misleading steps. As a simple mitigation strategy, we exclude external thoughts for level-5 problems (the most challenging) in the MATH-500 dataset and allow the LRM to generate reasoning from scratch.

We apply this adjustment specifically to distillation-based models paired with larger CoT generators (e.g., Plus and Max), which are most vulnerable to this failure mode due to their inability to re-think. As shown in Table 4, this approach significantly improves accuracy and reduces the harmful cases, while still achieving substantial token savings compared to the baseline. The results suggest that a practical enhancement to ThoughtMani is to reserve internal reasoning for the hardest problems, effectively balancing efficiency and reliability. Although recent work has investigated difficulty-aware and adaptive reasoning mechanisms (Liu et al., 2025; Zhu and Li, 2025), integrating such methods is left for future work.

3.4 Execution Time

As shown in Table 5, we measure the execution time of ThoughtMani on the MATH-500 dataset to evaluate the real-world inference speedup. For distillation-based LRMs, the execution time is reduced by approximately 25%, while for RL-based models, the improvement reaches nearly 50%. The reduction in latency demonstrates that ThoughtMani not only saves tokens but also delivers acceleration in practice, highlighting its prac-

Table 4: **Error Mitigation**: We report the accuracy and token consumption using the mitigation. The number within parentheses indicates the average thought length. # degrades represents the number of degradation cases.

Ex.	Method	DPSK-14B			DPSK-32B		
		Acc	Token	# degrade	Acc	Token	# degrade
-	Baseline	65.3	2392	-	84	2973	-
Plus	Ours	65.4	600(141)	37	68.3	554(141)	43
	Enhanced	$66.4_{\uparrow 1.0}$	1381(96)	20	$75.0_{\uparrow 6.7}$	1752(96)	18
Max	Ours	64.6	595(209)	34	67.8	562(209)	40
	Enhanced	$69.4_{\uparrow 4.8}$	1398(143)	15	$75.8_{\uparrow 8.0}$	1684(143)	19

Table 5: **Execution Time**: we record the execution time of running ThoughtMani and baseline on the MATH-500 dataset. We report the average running time.

Model	Baseline	ThoughtMani			Improve
		Thoughts	Inference	Total	
QwQ	19.29	0.20	9.87	10.07	47.80%
DPSK-32B	10.11		7.35	7.55	25.32%
DPSK-14B	3.41		2.41	2.61	23.46%

tical value for real-world applications where efficiency and response speed are critical.

3.5 Other Evaluation

We further study the generalization ability of our approach across different tasks and the impact of token budgets on performance. Regarding the generalization ability, we utilize task-specific prompts to generate the external thoughts on the Code dataset, but observe no significant performance difference between the two settings. The results are shown in Appendix B.4 Regarding the token budget, we limit the token budget of generation and find that the token reduction persists at all budget levels. The results are shown in Appendix B.5

4 Related Work

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).

Chain-of-Thought Compression. Despite the improved capabilities, introducing intermediate CoTs brings additional token overhead. For traditional LLMs, lots of efforts have been put into reducing redundant steps (Ding et al., 2024; Liu et al., 2024; Ma et al., 2025b) 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 utilizes embeddings instead of tokens to serve as the CoT. With the development of LRMs, many works focus on solving the overthinking problem (Chen et al., 2024; Fan et al., 2025; Zhu and Li, 2025; Sui et al., 2025).

Our work is similar to speculative decoding (Leviathan et al., 2023), but differs in that we use a smaller model only to generate reasoning thoughts, enabling the main model to skip internal thinking and reduce token generation. Speculative decoding, in contrast, uses a small model to guess output tokens without shortening the reasoning process. Another similar method is NoThink (Ma et al., 2025a), which inserts a placeholder sentence to avoid thinking. While similar in spirit, our approach goes further by using meaningful external thoughts and revealing how RL-based and distillation-based models differ in their response to external reasoning—enabling more informed and effective reasoning skipping.

5 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, ThoughtMani makes LRMs more accessible for real-world, resource-constrained applications.

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*.

Chenrui Fan, Ming Li, Lichao Sun, and Tianyi Zhou. 2025. Missing premise exacerbates overthinking: Are reasoning models losing critical thinking skill? *arXiv preprint arXiv:2504.06514*.

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.

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*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

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. 2024. Can language models learn to skip steps? *arXiv preprint arXiv:2411.01855*.

Yongjiang Liu, Haoxi Li, Xiaosong Ma, Jie Zhang, and Song Guo. 2025. Think how to think: Mitigating overthinking with autonomous difficulty cognition in large reasoning models. *arXiv preprint arXiv:2507.02663*.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. 2025a. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025b. Cot-valve: 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*.

Gokul Swamy, Sanjiban Choudhury, Wen Sun, Zhiwei Steven Wu, and J Andrew Bagnell. 2025. All roads lead to likelihood: The value of reinforcement learning in fine-tuning. *arXiv preprint arXiv:2503.01067*.

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*.

Markus Wulfmeier, Michael Bloesch, Nino Vieillard, Arun Ahuja, Jorg Bornschein, Sandy Huang, Artem Sokolov, Matt Barnes, Guillaume Desjardins, Alex Bewley, and 1 others. 2024. Imitating language via scalable inverse reinforcement learning. *Advances in Neural Information Processing Systems*, 37:90714–90735.

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*.

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.

Jason Zhu and Hongyu Li. 2025. Towards concise and adaptive thinking in large reasoning models: A survey. *arXiv preprint arXiv:2507.09662*.

A Supplement for Effectiveness of External Thoughts

A.1 Sample Output of Replacing Thoughts

Thinking is Skipped

<|im_start|> **User:** If $f(x) = \dfrac{2}{x+1}$, then what is the value of $f^{-1}\left(\frac{1}{5}\right)$? <|im_end|>
<|im_start|> **Assistant:** <|im_end|>
<think>
[External Thoughts]
Understand the inverse function: The inverse function $f^{-1}(y)$ is defined such that if $y = f(x)$, then $x = f^{-1}(y)$.
Set up the equation: Set $y = \frac{1}{5}$ and solve $\frac{1}{5} = \frac{2}{x+1}$.
Solve for x: Cross-multiplying gives $1 \cdot (x+1) = 5 \cdot 2$, so $x + 1 = 10$, hence $x = 9$. Therefore, $f^{-1}\left(\frac{1}{5}\right) = 9$. </think>

[Final Response] To find $f^{-1}\left(\frac{1}{5}\right)$ for the function $f(x) = \dfrac{2}{x+1}$:
1. **Set the function equal to $\frac{1}{5}$**:

$$\frac{2}{x+1} = \frac{1}{5}$$

2. **Solve for x**: - Cross-multiply to eliminate the denominators:

$$2 \cdot 5 = 1 \cdot (x+1)$$

- Simplify:

$$10 = x + 1$$

- Subtract 1 from both sides:

$$x = 10 - 1 = 9$$

Thus, the value of $f^{-1}\left(\frac{1}{5}\right)$ is $\boxed{9}$.

Thinking is Not Skipped (Rethink)

<|im_start|> **User:** A regular hexagon can be divided into six equilateral triangles. If the perimeter of one of the triangles is 21 inches, what is the perimeter, in inches, of the regular hexagon? <|im_end|>
<|im_start|> **Assistant:** <|im_end|>
<think>
[External Thoughts]
To solve this problem, consider the following key points:

- A regular hexagon can be divided into six equilateral triangles.
- The perimeter of one of these triangles is given as 21 inches.
- Since the triangle is equilateral, all three sides are equal. Thus, each side of the triangle is 7 inches (21 inches / 3).
- Each side of the equilateral triangle is also a side of the hexagon.
- A regular hexagon has six equal sides.
- Therefore, the perimeter of the hexagon is 6 times the length of one side of the triangle.

</think>

[Rethinking] Okay, let me try to figure this out. So, the problem is about a regular hexagon divided into six equilateral triangles. The perimeter of one of those triangles is 21 inches, and I need to find the perimeter of the hexagon. Hmm, let's start breaking it down step by step.
First, I know that a regular hexagon has all sides equal, and when it's divided into six equilateral triangles, those triangles must be using the sides of the hexagon as their sides. So, each triangle is equilateral, meaning all three sides are the same length.
The problem says the perimeter of one of these triangles is 21 inches. Since the triangle is equilateral, all three sides are equal. To find the length of one side, I can divide the perimeter by 3. Let me write that down:
Perimeter of triangle = 21 inches
Each side of the triangle = $21 \div 3 = 7$ inches
Okay, so each side of the triangle is 7 inches.
Now, since the hexagon is regular, all its sides are equal too. And since the triangles are formed by dividing the hexagon, each side of the triangle must correspond to a side of the hexagon. Wait, actually, let me make sure that's the case.
If you imagine a regular hexagon, and then draw lines from the center to each vertex, that divides it into six equilateral triangles. Each

of those triangles has two sides that are radii of the hexagon and one side that is a side of the hexagon. Wait a second, hold on. Maybe I need to clarify the structure here.
In a regular hexagon, when you connect the center to all six vertices, each of those six triangles is indeed equilateral. Each triangle has two sides that are equal to the radius of the circumscribed circle (the distance from the center to a vertex), and the third side is a side of the hexagon. But in an equilateral triangle, all sides must be equal. So, that would mean that the radius (the distance from the center to a vertex) is equal to the length of the side of the hexagon. Hmm, actually, that must be the case because if the triangle is equilateral, all three sides must be equal. Therefore, the two sides that are radii must be equal in length to the side of the hexagon. So, each of those triangles has sides equal to the side length of the hexagon. So, in that case, the perimeter of each triangle would be three times the side length of the hexagon.
Therefore, if the perimeter of the triangle is 21 inches, each side of the triangle (and hence each side of the hexagon) is $21 \div 3 = 7$ inches, as I calculated earlier.
Since the hexagon has six sides, each 7 inches long, the perimeter of the hexagon would be $6 \times 7 = 42$ inches.
Wait, that seems straightforward, but let me just verify once more to be sure.
Another way to think about this: The side of the equilateral triangle is the same as the side of the hexagon because the triangle is formed by connecting the center to two adjacent vertices. So the base of the triangle (the side that's part of the hexagon's perimeter) is equal in length to the other two sides (the radii). Therefore, each side of the triangle is equal to the side of the hexagon.
Given that, the perimeter of the triangle is $3s = 21$, so solving $3s = 21$ gives $s = 7$. Then the hexagon's perimeter is $6s = 6 \times 7 = 42$.
Hmm, seems consistent. I don't see any errors in that logic. Let me just visualize it again. Imagine a regular hexagon divided into six triangles with the center. Each triangle is equilateral, so all their sides are equal. Therefore, each triangle's sides must be equal to the edges of the hexagon. So, the perimeters relate as described.
Therefore, I think the correct answer is 42 inches.
Final Answer
The perimeter of the regular hexagon is $\boxed{42}$ inches. </think>
[Final Response] A regular hexagon can be divided into six equilateral triangles, each with a perimeter of 21 inches. To find the perimeter of the hexagon:
1. **Determine the side length of the equilateral triangle:** - The perimeter of one equilateral triangle is 21 inches. - Since all sides of an equilateral triangle are equal, each side length is $\frac{21}{3} = 7$ inches.
2. **Relate the side length of the triangle to the hexagon:** - Each side of the equilateral triangle corresponds to a side of the hexagon. - Therefore, each side of the hexagon is 7 inches.
3. **Calculate the perimeter of the hexagon:** - A regular hexagon has six equal sides. - The perimeter is $6 \times 7 = 42$ inches.
Thus, the perimeter of the regular hexagon is $\boxed{42}$ inches.

Code Example for ThoughtMani **Template.**

```
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
```

A.2 Token-Level Analysis: Optimal Position

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 6.

Inference Template - Case 1 (within chat)

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

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 – 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 7.

Inference Template - Case 2 (no thinking token)

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

A.3 Metric of Efficacy

Double-check Condition. It measures how often the model shows signs of reconsidering or revising its reasoning before reaching a conclusion. This is detected by looking for specific words such as "Hmm", "Wait", and "Alternatively" in the part of the response that comes before the </think> tag. These words suggest the model is pausing, correcting itself, or exploring different ideas. The total count of such words is calculated using case-insensitive exact matching. If the </think> tag is missing, the count is zero. To avoid unrealistic values, any count of 30 or higher is treated as zero.

Table 7: 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

Reasoning Step. It reflects how the model breaks down its thinking into distinct parts after the </think> tag. The text is split into segments wherever a new step in reasoning begins, as identified by a regular expression that detects structural shifts such as line breaks or numbered points. The number of resulting segments is taken as the count of reasoning steps. If the text is empty or no boundaries are found, the entire content is treated as a single segment. As with double-checks, if the number of segments reaches 30 or more, it is set to zero to prevent distortion from unusually long or fragmented outputs.

Together, these metrics help assess the model's reasoning style—how carefully it checks its work and how clearly it structures its thoughts—offering insight beyond just the correctness of the final answer.

A.4 Pipeline

B Supplement for Experiment

B.1 Experimental Settings

Model Setting. 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[2] 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

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

Algorithm 1: ThoughtMani Pipeline

Input: A dataset $D = \{q_1, q_2, \ldots, q_n\}$ containing problems, a CoT generator model G, a reasoning model M

Output: Final responses $\{r_1, r_2, \ldots, r_n\}$ 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 $\{r_1, r_2, \ldots, r_n\}$;
return $\{r_1, r_2, \ldots, r_n\}$

output token number to 20,000 for the remaining datasets.

Baseline.

- *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., 2025b) 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 model for comparison. Specifically, we choose CoT-Valve+P as the fine-tuning pattern.

B.2 Suboptimal Mode

Flawed Thought Identification.

Flawed Thought

You are a highly skilled mathematical reasoning evaluator. Your task is to do the following:
1. **Analyze a provided solution or reasoning** for the same problem.
2. **Check whether the reasoning is correct**, identifying any logical flaws, computational errors, or misconceptions.
3. Specifically: - Verify the mathematical validity of each step.
- Determine if the final answer matches the correct one.
- Explain clearly whether the idea or solution is correct or not, and why.
Be thorough, precise, and educational in your response
Problem: problem Reasoning: reasoning
Your task: 1. Evaluate the reasoning and answer above. Is the logic correct? If not, where and why does it fail?
2. You don't need to provide a correct answer or provide a detailed analysis.
3. If any incorrect part exists, output True enclosed by boxed, otherwise False.

Sample Output of Flawed Thought

[Input Question] In triangle ABC, $AB = 17$, $AC = 8$, and $BC = 15$. Let D be the foot of the altitude from C to AB. Find the area of triangle ACD.
[Input Resoning] To find the area of triangle ACD, we can use the following key points:
1. Recognize that D is the foot of the altitude from C to AB, which means CD is perpendicular to AB.
2. Use Heron's formula or another method to calculate the area of the whole triangle ABC, as it will be useful for later steps.
3. The area of triangle ABC can also be expressed as $\frac{1}{2} \times$ base $\times$ height, where the base

is AB and the height is CD.
4. Since we have the length of AB and can find the area of triangle ABC, we can solve for the length of CD.
5. Knowing CD (the height) and AC (one of the bases), we can then find the area of triangle ACD using the formula $\frac{1}{2} \times$ base $\times$ height.
6. It's important to note that since D is on AB, $AD + DB = AB$, but we don't need this information directly to find the area of ACD.

[Error Analysis] We are given a problem involving triangle ABC with side lengths:
- $AB = 17$ - $AC = 8$ - $BC = 15$
Point D is the foot of the altitude from C to AB, and we are to **find the area of triangle ACD**.
—
Step 1: Analyze the provided reasoning
The reasoning outlines the following steps:
1. D is the foot of the altitude from C to AB, so $CD \perp AB$. (Correct)
2. Use Heron's formula or another method to compute the area of triangle ABC. (Valid idea)
3. The area of triangle ABC can also be written as $\frac{1}{2} \cdot AB \cdot CD$, since CD is the height from C to base AB. (Correct)
4. Since $AB = 17$ and area of ABC can be computed, solve for CD. (Valid)
5. Then, **use AC as a base and CD as the height** to compute the area of triangle ACD using $\frac{1}{2} \cdot AC \cdot CD$. (**This is incorrect**)
6. Notes that $AD + DB = AB$, but claims it's not needed. (Partially true, but missing insight)
—
Step 2: Identify the flaw
The **critical error** lies in **Step 5**:
> "Knowing CD (the height) and AC (one of the bases), we can then find the area of triangle ACD using the formula 1/2 * base * height."
This assumes that CD is the height **relative to base AC** in triangle ACD. But **this is not true**.
- In triangle ACD, if we take AC as the base, then the height must be the perpendicular distance from D to line AC.
- However, CD is perpendicular to AB, **not** to AC.
- Therefore, CD is **not** the height corresponding to base AC in triangle ACD.
Thus, computing $\frac{1}{2} \cdot AC \cdot CD$ does **not** give the area of triangle ACD. That formula would only be valid if CD were perpendicular to AC, which it is not.
Even though CD is an altitude to AB, it is **not** an altitude to AC, so using it as the height with base AC is geometrically invalid.
—
Step 3: Conclusion
The logic fails in Step 5 due to a **misapplication of the area formula**. The height CD cannot be used with base AC to compute the area of triangle ACD, because the height must be measured perpendicular to the chosen base.
While earlier steps (computing area of ABC, solving for CD) are valid and useful, the final step incorrectly assumes CD serves as a height relative to AC, which it does not.
To correctly find the area of triangle ACD, one would need to find either:
- The length AD, then use $\frac{1}{2} \cdot AD \cdot CD$ (since $CD \perp AB$, and AD lies on AB), or
- Use coordinates or trigonometry to find the appropriate height.
—
Final Answer:

$$\boxed{\text{True}}$$

The reasoning contains an error — specifically, the incorrect assumption that $\frac{1}{2} \cdot AC \cdot CD$ gives the area of triangle ACD. Therefore, the answer is **True**, indicating that the reasoning **is flawed**.

Thought Deviation.

Prompt For Identifying Thought Deviation

Here are the reasoning steps provided by the model:
segments
And here is the reference CoT:
reference_cot
Please analyze the following:
1. How many steps does the reference CoT contain?
2. How many steps of the reference CoT are followed by the model's response?
3. Does the model adopt any new approach or steps that were not mentioned in the reference

CoT?
Output:
- Number of reference CoT steps: [number of steps]
- Percentage of followed steps: [percentage %]
- Does the model adopt a new way to solve the problem: [Yes/No]

Sample Output of Deviation - 1

- Number of reference CoT steps: 6
- Percentage of followed steps: 100%
- Does the model adopt a new way to solve the problem: Yes
The model's response follows all the steps outlined in the reference CoT, but it also includes additional details and calculations, such as explicitly checking specific values of (k) and verifying the calculations for (m) and (n). This indicates that the model adopts a more detailed approach to ensure the solution is correct, which was not explicitly mentioned in the reference CoT."

Sample Output of Deviation - 2

- Number of reference CoT steps: 4
- Percentage of followed steps: 0%
- Does the model adopt a new way to solve the problem: Yes
The model's response takes a different approach by analyzing the parity of each intermediate function in the composition, rather than focusing on the sign changes with each application of f and g as in the reference CoT.

B.3 Identifying the Skip of Thought

Our initial experiments on MATH-500 show that a simple heuristic—checking for the absence of phrases like "Alright" or "I think"—can detect skipped reasoning with about 90% recall, meaning it correctly identifies most cases where the model skips internal thinking. However, the precision for detecting rethinking (i.e., when the model does not skip) is only around 40%, meaning many predicted rethinking cases are false positives.

This low precision is not a major concern because rethinking is a useful and often necessary behavior. When the external CoT is incomplete or wrong, the LRMs needs to generate its own reasoning to arrive at the correct answer. While this increases token usage, it ensures accuracy. Therefore, high recall in detecting skipping is more important—it lets us reliably identify when the model is saving computation, while occasional misclassification of rethinking does not harm overall performance or efficiency. We leave more accurate detection methods for future work.

B.4 Generalization Ability

To evaluate the generalization capability of our approach across different task domains, we conduct experiments on the Code dataset, which involves code generation tasks fundamentally distinct from mathematical reasoning. We investigate whether task-specific prompting is necessary for effective performance or if a general-purpose CoT generation framework can transfer seamlessly.

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

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>"

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>"

Table 8: **Generalization Ability**: The results are reported using Pass@1 and the number of generated tokens. Normal represents using original COTs, while Specific represents using the task-specific CoTs.

Ex.	QwQ-32B				DPSK-32B			
	Normal		Specific		Normal		Specific	
	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
3b	65.6	6842	64.1	7009	59.1	6557	55.7	6163
7b	62.3	4409	61.7	4485	41.7	528	42.4	627
Plus	64.1	4461	61.9	4408	45.8	528	45.3	539
Max	60.9	4209	62.2	4181	44.5	465	41.1	577

In the baseline setting, we employ the standard, domain-agnostic prompt described in previous sections. At the same time, we further test a specialized variant of this prompt—adapted to emphasize code-related reasoning (e.g., algorithmic structure and control flow), while still adhering to the principle of avoiding implementation-level details.

The results, shown in Table 8, 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.

B.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 4. 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.

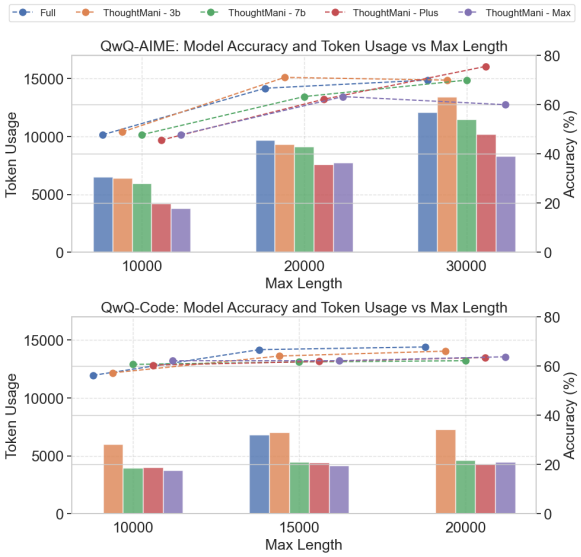

Figure 4: **Relation between Model Performance and Token Budgets:** Bar plot represents the token consumption and line plot represents the model utility (Accuracy or Pass@1)