

Can Reasoning Path still be Effective as Input? Bridging Post-Reasoning to Chain-of-Thought Compression

Chengzhengxu Li¹, Xiaoming Liu^{1,*}, Zhaohan Zhang², Shengchao Liu¹,
Guoxin Ma¹, Yu Lan¹, Cong Wang³, Chao Shen¹

¹Faculty of Electronic and Information Engineering, Xi'an Jiaotong University

²Queen Mary University of London, ³City University of Hong Kong

* Corresponding author

{czx.li, liusc, guoxin.ma}@stu.xjtu.edu.cn, zhaohan.zhang@qmul.ac.uk

{xm.liu, chaoshen, ylan2020}@xjtu.edu.cn, congwang@cityu.edu.hk

Abstract

Recent developments have enabled advanced reasoning in Large Language Models (LLMs) via long Chain-of-Thought (CoT), trading efficiency during inference for performance. Existing works focus on compressing generated CoT in reasoning, which impairs the necessary information for deriving the correct answer. In this work, we propose post-reasoning, a reasoning paradigm that takes CoT as a part of context to simplify the reasoning task for LLMs. We find that post-reasoning significantly reduces the generation length of LLMs, but its effectiveness hinges on the efficiency and the reliability of the contextual CoT generation. Therefore, we propose Upfront CoT (UCoT), an efficient post-reasoning framework for CoT compression. UCoT trains a lightweight model (compressor) to provide contextual CoT in form of soft tokens and trains the LLM (executor) to leverage this contextual CoT for producing the final answer. Extensive experiments show that UCoT maintains the powerful reasoning ability of executor while significantly reducing the length of CoT. It is worth mentioning that when applying UCoT to the Qwen2.5-7B-Instruct model, the usage of tokens on GSM8K dataset is reduced by 50%, while the performance is 3.08% higher than that of the state-of-the-art (SOTA) method. The code is available at: <https://github.com/czx-li/UCoT>.

1 Introduction

Chain-of-Thought (CoT) reasoning has emerged as a pivotal paradigm for overcoming the bottleneck of complex reasoning capabilities in large language models (LLMs) (Wei et al., 2022; Sprague et al., 2025). Recent advancements exemplified by models such as DeepSeek-R1 (Guo et al., 2025) and OpenAI’s o3 (El-Kishky et al., 2025), demonstrate that scaling test-time computation on reasoning is a key path to advanced intelligence (Snell et al., 2024). This technique strengthens the model’s abil-

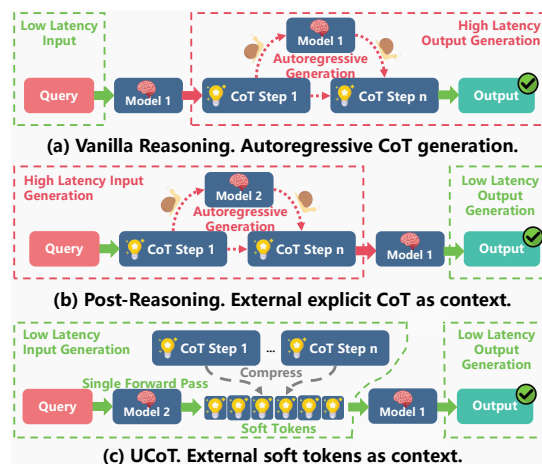


Figure 1: Comparison of reasoning paradigms. (a) Vanilla Reasoning generates CoT autoregressively, causing high output latency. (b) Post-Reasoning uses explicit CoT as input, trading contextual CoT generation for short output. (c) UCoT generates soft tokens via a single forward pass, achieving low latency and high performance.

ity to reason from scratch by decomposing complex problems into sequential steps, but it brings substantial computational overhead due to the autoregressive nature of current LLMs (Su et al., 2025; Arora and Zanette, 2025). Existing works about compressing the CoTs focus on simplifying model output through prompting strategies (Xu et al., 2025a; Aytes et al., 2025) or post-training (Kang et al., 2025; Xia et al., 2025) without providing any complementary contextual information, which leads to efficiency gains at the cost of significant performance degradation (Zhang et al., 2025).

In this work, we revisit the role of CoT as the premise for deriving the final answer. As illustrated in Figure 1 (a), we point that CoT is not merely a sequence of outputs, but effectively a self-generated input context that the LLM produces to bridge the logical gap between the input query and the final answer (Zhang et al., 2022; Kojima et al., 2022; Wu

et al.). Motivated by this observation, we first investigate a critical research question to CoT reasoning: “Can providing reasoning information as input reduce CoT generation in LLMs?” Toward this end, we introduce the paradigm of **post-reasoning** as illustrated in Figure 1 (b), where an LLM continues its reasoning based on contextual CoT (i.e., CoT generated by a lightweight external model and provided as part of the LLM input). We further validate the feasibility of this paradigm through a pilot study (Section 3) and pose two critical factors influencing the performance of post-reasoning:

- \mathcal{F}_1 : Post-reasoning trades contextual CoT generation for LLM shortened output.
- \mathcal{F}_2 : Post-reasoning performance hinges on the quality of contextual CoTs.

The dependency of post-reasoning on efficiently generating reliable contextual CoT motivates us to propose **Upfront Chain-of-Thought (UCoT)** as shown in Figure 1 (c), a post-reasoning framework designed to achieve an optimal trade-off between efficiency and effectiveness in reasoning. UCoT follows the post-reasoning paradigm, alleviating the need to generate lengthy reasoning paths by supplying compressed contextual CoT. Specifically, UCoT **constructs contextual CoT as a sequence of soft tokens** produced by a lightweight model (compressor), and performs post-reasoning using a large model (executor). We first distill the reasoning capability of the large model into the compressor, and train the compressor to encode reasoning processes into soft tokens. The executor is then trained to leverage this compressed contextual CoT to generate concise yet accurate responses. Experimental results demonstrate that UCoT compresses CoT outputs while preserving LLM reasoning ability, and outperforms baselines and SOTA on multiple tasks. For instance, UCoT helps Qwen2.5-7B-Instruct cut 50% of required tokens with only a 5.62% performance drop on GSM8K, obtaining 3.08% better than SOTA method Tokenskip (Xia et al., 2025). Our contributions are as follows:

- **Post-Reasoning.** We propose post-reasoning as a novel reasoning paradigm where the model continues reasoning conditioned to contextual CoT. It reveals the feasibility of breaking the latency bottleneck in autoregressive models by contextualizing the reasoning path.
- **Efficient Reasoning Framework.** We intro-

duce UCoT, a post-reasoning framework that takes a sequence of soft tokens as contextual CoT and reducing the generation length of LLMs. Through two-stage training, UCoT enables a lightweight compressor to inject high-quality contextual CoT that guides efficient and accurate reasoning.

- **Outstanding Performance.** Our method significantly outperforms baseline and SOTA methods on two open-source models and two public datasets. Furthermore, we also demonstrate its excellent generalization ability to currently popular large reasoning models and more challenging datasets.

2 Related Works

CoT Reasoning in LLMs. With the broader application of LLMs in logistically-complex domain such as mathematical study (Setlur et al., 2024), code generation (Fakhoury et al., 2024), and scientific question answering (Zhuang et al., 2023), it becomes essential to enhance the reasoning ability of LLMs (Liu et al., 2024; Ramprasad et al., 2024). Researchers have proposed the Chain-of-Thought (CoT) reasoning method (Wei et al., 2022; Wang et al., 2022), which enhances model performance by explicitly generating intermediate reasoning steps through LLMs, to unlock the potential of LLMs. Subsequent studies built upon CoT by employing multi-step sampling (Yu et al., 2023; Yao et al., 2023) or multi-path summarization (Chen et al., 2024; Sun et al., 2024) to aggregate information from multiple reasoning trajectories, thereby further enhancing LLMs’ reasoning capabilities in complex problem-solving. However, since transformer-based LLMs generate text autoregressively, their processing time for complex problems increases linearly with the length of the CoT (Nayab et al., 2024). Therefore, our work focus on reducing the length of CoT generated by LLMs while maintaining their reasoning performance.

CoT Compression. CoT compression aims to reduce the output length of the modern reasoning models while preserving reasoning accuracy. Numerous studies employ carefully designed discrete prompts to guide LLMs toward generating more concise reasoning paths at the output stage (Xu et al., 2025a; Han et al., 2024). However, due to the discrete nature of text, the discrete prompts cannot be directly optimized with gradient descent, necessitating extensive manual design (Aytes et al., 2025)

or complex gradient-free optimization methods (Li et al., 2025). Another line of research focuses on fine-tuning LLMs on datasets with short but high-quality CoT reasoning paths (Cui et al., 2025; Xia et al., 2025). However, the cost-effective construction of high-quality concise CoT datasets remains an outstanding challenge. In addition, Cheng and Van Durme (2024); Saunshi et al. (2025); He et al. (2025); Zhang et al. (2025) compress each CoT step into continuous latent representations, thereby limiting explicit generation by LLMs at the output stage while incurring performance degradation. In contrast, our work shifts the focus to the LLM input, using contextual CoT to help LLMs shorten reasoning before generation. This enables the spontaneous production of concise CoTs, avoiding information loss from external dataset construction or limiting explicit generation. In Appendix B, we also provide the formal definitions of Chain-of-Thought Compression.

3 Preliminary

In this section, we formalize the post-reasoning paradigm and conduct a pilot study to analyze the impact of post-reasoning on the output length and performance of LLM.

3.1 Paradigm Formulation

Vanilla Reasoning. Given an input query $Q = \{q_1, q_2, \dots, q_{|Q|}\}$ consisting of $|Q|$ tokens, vanilla reasoning process of the LLM is formulated as:

$$\{C, A\} = \text{LLM}(Q), \quad (1)$$

where $C = \{c_1, c_2, \dots, c_{|C|}\}$ is the CoT with length $|C|$ and $A = \{a_1, a_2, \dots, a_{|A|}\}$ is the final answer with length $|A|$. The CoT C is generated first, followed by the final answer A . Since $|C| \gg |A|$ in most cases, inference latency is primarily dominated by the generation of the CoT rather than the final answer (Pope et al., 2023).

Post-Reasoning. In post-reasoning paradigm, the contextual CoT C' generated by an external LLM is treated as a part of the input, which is formulated as:

$$\{\hat{C}, \hat{A}\} = \text{LLM}(Q \oplus C'), \quad (2)$$

where \hat{C} and \hat{A} represents the model’s output CoT and answer conditioned on the contextual CoT C' , respectively. The \oplus denote the concatenation of query Q and contextual CoT C' . We compare the length of $|\hat{C}| + |\hat{A}|$ with $|C| + |A|$ generated by

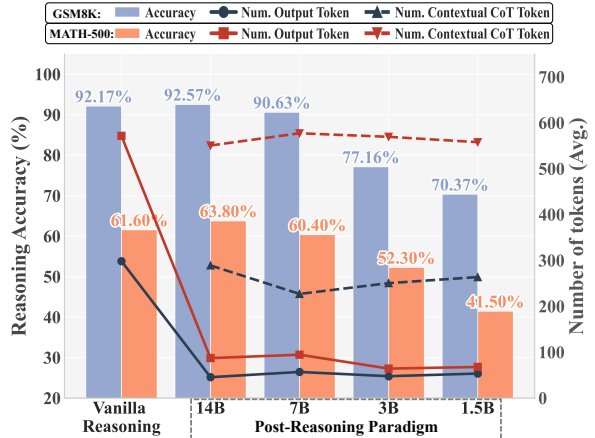


Figure 2: The impact Post-reasoning on the output length and performance. Post-reasoning facilitates reasoning tasks by trading upfront contextual CoT generation for a significant reduction in output tokens.¹

Vanilla Reasoning to measure if the model’s output can be reduced when given the contextual CoT C' . Based on this setup, we subsequently explore how post-reasoning affects both output lengths and final answer accuracy in the pilot study.

3.2 How Post-Reasoning Impacts Reasoning?

We conduct experiments with Qwen2.5-7B-Instruct (Hui et al., 2024). The model takes the query and CoT from external models of different scales as context to derive the output. We measure the average output length and calculate the answer accuracy for evaluating the paradigm performance.

As shown in Figure 2, our pilot study unveils following insights: *i) The post-reasoning paradigm significantly reduces the LLM’s output length but allocates more computation to contextual CoT generation.* Across both datasets, output tokens are reduced by over 80.00% relative to vanilla reasoning. However, this gain relies on generating contextual CoT of comparable scale. *ii) Post-Reasoning performance varies with the quality of contextual CoT.* When CoTs generated by smaller models (e.g., Qwen2.5-1.5B) are used as context, model accuracy drops sharply by approximately 20.00%, highlighting the necessity of reliable and informative contextual CoT. These findings validate the feasibility of post-reasoning as an efficient reasoning paradigm, and further reveal that the length and quality of the contextual CoT are two critical factors governing its efficiency and performance.

¹We employ Qwen2.5-1.5B, 3B, 7B, 14B-Instruct for constructing contextual CoT and test on the GSM8K (Cobbe et al., 2021) and Math-500 (Hendrycks et al.) datasets.

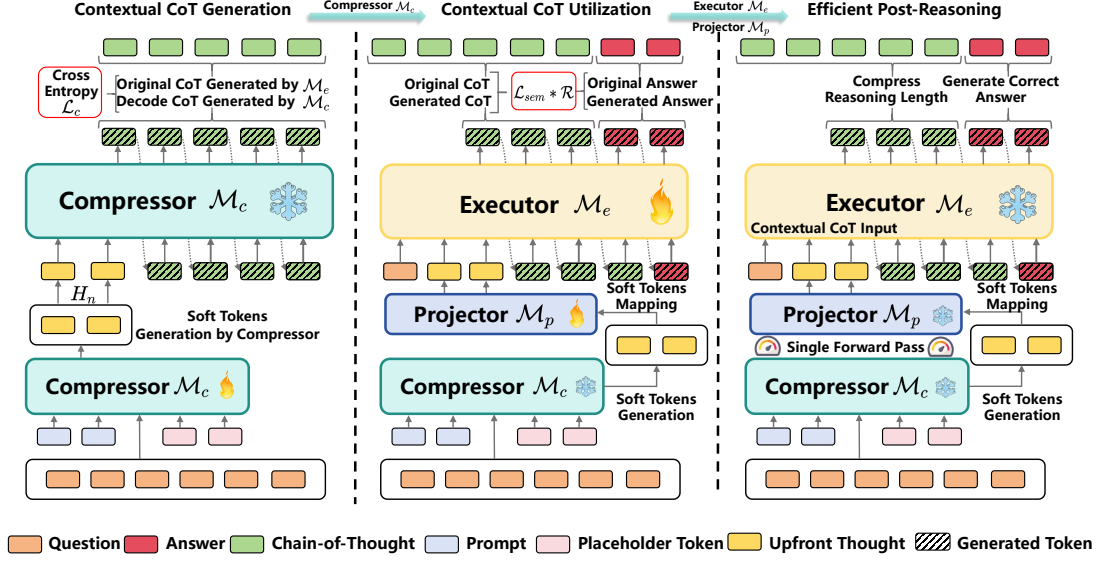


Figure 3: Overview of Upfront Chain-of-Thought. In contextual CoT generation stage (§4.1) compressor is trained \mathcal{M}_c to generate soft tokens. In contextual CoT utilization stage (§4.2), the executor \mathcal{M}_e learns to give correct responses with less CoT length based on the contextual soft tokens. In Efficient Post-Reasoning stage (§4.3), the compressor \mathcal{M}_c and the executor \mathcal{M}_e cooperate under the post-reasoning paradigm to achieve efficient reasoning.

4 Efficient Reasoning with Upfront Chain-of-Thought

Based on both insights, we propose the Upfront Chain-of-Thought (UCoT) framework to eliminate the autoregressive bottleneck of post-reasoning for contextual CoT. By transforming explicit reasoning paths into soft tokens via a single forward pass, UCoT achieves significant acceleration while maintaining high-fidelity logic. As illustrated in Figure 3, our pipeline facilitates a cooperative workflow between a lightweight **compressor** (for generating soft tokens) and a powerful **executor** (for deriving the final answer).

4.1 Contextual CoT Generation

The goal of this stage is to train a lightweight compressor \mathcal{M}_c to efficiently **condense complex contextual CoTs into a compact sequence of soft tokens**. Since lightweight models naturally lag behind larger models in reasoning capabilities (Wu et al., 2024), we distill the reasoning ability of larger model into the compressor.

Data Construction. We derive high-quality training data from the executor itself to ensure the reasoning style aligns with the downstream inference model. Given a dataset of query-answer pairs $\{(Q_n, A_n)\}_{n=0}^{N-1}$, we employ the executor \mathcal{M}_e to generate the ground-truth CoT C_n for each question Q_n :

$$C_n = \mathcal{M}_e(Z_e \oplus Q_n), \quad (3)$$

where Z_e is the system prompt for the executor, and \oplus denotes concatenation. This yields a training corpus $\mathcal{D} = \{(Q_n, C_n, A_n)\}_{n=0}^{N-1}$.

Contextual CoT Compression. To avoid the latency of autoregressive decoding, we train the compressor to encode the reasoning path into compact soft tokens. We append a sequence of placeholder tokens $S_{spec} = \{[\text{ucot}]_m\}_{m=0}^{M-1}$ of length M to the input. Instead of generating text, the compressor \mathcal{M}_c processes the input in a single forward pass, and we extract the last hidden states corresponding to these placeholders as the soft tokens:

$$X_c^n = (Z_c \oplus Q_n \oplus S_{spec}), \quad (4)$$

$$H_n = \mathcal{M}_c(X_c^n; \theta_c) = \{h_m\}_{m=0}^{M-1}, \quad (5)$$

where θ_c represents the learnable adaptor parameters (e.g., LoRA (Hu et al., 2022)) injected into the compressor, and H_n is the resulting soft tokens. Formally, H_n functions as the latent equivalent of C' in Eq. 2, allowing the subsequent executor to ingest the reasoning context in a compressed format.

Compressor Optimization. To ensure H_n captures the semantic essence of the reasoning path, we train the adaptor θ_c to reconstruct the original high-quality CoT C_n from these embeddings. Following the prompt tuning paradigm (Lester et al., 2021; Kuratov et al., 2025), we maximize the likelihood of the target CoT conditioned on the soft tokens:

$$\mathcal{L}_c = \mathbb{E}_{\mathcal{D}}(-\log P_{\mathcal{M}_c}(C_n | H_n)). \quad (6)$$

By minimizing \mathcal{L}_c , we force the compressor to squeeze the extensive reasoning information of C_n into the dense representation H_n . We present the pseudocode for training the compressor \mathcal{M}_c in Appendix A.6.

4.2 Contextual CoT Utilization

The soft tokens H_n need to be aligned with the embedding space of the executor to enable effective utilization by \mathcal{M}_e . Therefore, we use a projector \mathcal{M}_p , a two-layer MLP that maps H_n into \mathcal{M}_e 's embedding space. However, simple alignment is insufficient for effective compression. As noted in prior work (Xu et al., 2025b), LLMs before training often fail to directly leverage such compressed priors to actually reduce generation length or guide accurate reasoning. To address this, we introduce a reward-guided utilization mechanism that compels the executor to substitute explicit tokens with the provided soft token priors. Specifically, we constrain the reasoning process by imposing a limited decoding budget on the executor's output. This constraint yields a truncated residual CoT, denoted as \bar{C}_n , and is formulated as:

$$\bar{C}_n = \text{Cutoff}(\mathcal{M}_e(\mathcal{M}_p(H_n) \oplus Q_n; \theta_e))_\alpha, \quad (7)$$

where $\text{Cutoff}(\cdot)_\alpha$ truncates the explicit CoT to a compression ratio α , compelling the model to rely on H_n for the missing logic.

Executor Optimization. The executor is optimized towards ensuring both decoding complete semantic information from the contextual CoT and correctly answering the question. First, we design the semantic loss \mathcal{L}_{sem} to align the representation of the whole reasoning path, i.e., soft tokens and the executor's generation \bar{C}_n before answer with the original long CoT C_n in the executor's final hidden state:

$$\begin{aligned} \mathcal{L}_{\text{sem}} &= \mathbb{E}_D(\text{Dist}(H_{\text{UCoT}}, H_{\text{CoT}})), \\ H_{\text{UCoT}} &= f_{\mathcal{M}_e}(\mathcal{M}_p(H_n) \oplus \bar{C}_n), \\ H_{\text{CoT}} &= f_{\mathcal{M}_e}(C_n), \end{aligned} \quad (8)$$

where $f_{\mathcal{M}_e}(\cdot)$ extracts the last hidden state and $\text{Dist}(\cdot)$ is the standard Mean Absolute Error function. Second, to ensure the soft tokens guide the model toward the correct answer, we introduce a reward factor \mathcal{R} . This factor penalizes discrepancies in answer confidence between the compressed and original inference:

$$\begin{aligned} \mathcal{R} &= \mathbb{E}_D((r_{\text{UCoT}} - r_{\text{CoT}})^2), \\ r_{\text{UCoT}} &= -\log P_{\mathcal{M}_e}(A_n | \mathcal{M}_p(H_n); \bar{C}_n; \theta_e), \\ r_{\text{CoT}} &= -\log P_{\mathcal{M}_e}(A_n | C_n), \end{aligned} \quad (9)$$

The final objective $\mathcal{L}_e = \mathcal{L}_{\text{sem}} \cdot \mathcal{R}$ ensures that the executor effectively decodes the soft tokens to recover the full reasoning semantics while maintaining high confidence in the correct answer. The training pseudocode is provided in Appendix A.6.

4.3 Efficient Post-Reasoning

During inference, UCoT operates under the post-reasoning paradigm via a two-stage collaborative pipeline. First, the compressor \mathcal{M}_c distills the reasoning logic into a sequence of soft tokens. Next, the executor \mathcal{M}_e ingests these tokens as contextual CoT C' in Section 3.1 combined with the query to efficiently derive the answer. Formally, given a query Q_n , the inference process is formulated as:

$$\begin{aligned} C'_n &= \mathcal{M}_p(H_n) = \mathcal{M}_p(\mathcal{M}_c(Z_c \oplus Q_n \oplus S_{\text{spec}})), \\ \{\hat{C}_n, \hat{A}_n\} &= \mathcal{M}_e(Q_n \oplus C'_n), \end{aligned} \quad (10)$$

where the executor's output consists of a concise reasoning path \hat{C}_n and the final answer \hat{A}_n .

5 Experiments

To verify UCoT's effectiveness, we conduct extensive experiments using Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-7B-Instruct (Hui et al., 2024) as executors, and Qwen2.5-1.5B-Instruct (Hui et al., 2024) as the compressor with two mathematical reasoning benchmark datasets: GSM8K (Cobbe et al., 2021) and Math-500 (Hendrycks et al., 2021). For comprehensive details on the datasets, the implementation of our method, and the training and evaluation protocol, please refer to Appendix A.1, A.4, and A.5.

We compare our method UCoT to four baselines: Prompt (Xia et al., 2025), Truncation (Xia et al., 2025), CoD (Xu et al., 2025a), and Tokenskip (Xia et al., 2025) under three different compression ratios (0.9, 0.7, and 0.5) to verify the effectiveness of our method. The specific introductions and implementation processes of the comparison methods are in Appendices A.2 and A.3. At the same time, we conduct a systematic evaluation for the performance of each method in the field of CoT compression from four key metrics: accuracy, the number of compressed CoT tokens, end-to-end latency, and actual compression ratio², which have also been proposed and utilized in prior work (Xia et al., 2025). Among them, accuracy is used to measure the ability of the method to maintain the

²For simplicity, these key metrics are denoted as Acc, Tokens, Latency, and ActRatio in the results tables, respectively.

Methods	Ratio	GSM8K				MATH-500			
		Acc. (%) \uparrow	Tokens \downarrow	Latency (s) \downarrow	ActRatio \downarrow	Acc. (%) \uparrow	Tokens \downarrow	Latency (s) \downarrow	ActRatio \downarrow
<i>Qwen2.5-7B Series</i>									
Original	-	92.17 _{0.00} \downarrow	298.63	3.83 _{1.00} \times	-	61.60 _{0.00} \downarrow	571.64	6.35 _{1.00} \times	-
Prompt (Xia et al., 2025)	0.9	91.47 _{0.70} \downarrow	281.65	3.74 _{1.02} \times	0.97	60.40 _{1.22} \downarrow	553.27	6.27 _{1.01} \times	0.97
	0.7	90.44 _{1.73} \downarrow	287.32	3.77 _{1.02} \times	0.99	59.90 _{1.72} \downarrow	567.49	6.34 _{1.00} \times	0.99
	0.5	89.69 _{2.48} \downarrow	283.20	3.69 _{1.04} \times	0.98	60.30 _{1.32} \downarrow	577.07	6.37 _{1.00} \times	1.01
Truncation (Xia et al., 2025)	0.9	79.68 _{12.49} \downarrow	272.32	3.47 _{1.10} \times	0.91	42.70 _{18.92} \downarrow	514.48	5.72 _{1.11} \times	0.90
	0.7	52.42 _{39.75} \downarrow	210.55	2.68 _{1.43} \times	0.70	35.30 _{26.32} \downarrow	405.86	4.50 _{1.41} \times	0.71
	0.5	12.30 _{79.87} \downarrow	154.68	2.01 _{1.91} \times	0.52	8.50 _{53.12} \downarrow	285.82	3.71 _{1.71} \times	0.50
CoD (Xu et al., 2025a)	0.9	90.98 _{1.19} \downarrow	257.77	3.43 _{1.12} \times	0.89	59.10 _{2.52} \downarrow	508.76	5.66 _{1.12} \times	0.89
	0.7	87.54 _{4.63} \downarrow	223.02	2.97 _{1.29} \times	0.77	56.60 _{5.02} \downarrow	394.43	4.53 _{1.40} \times	0.69
	0.5	84.96 _{7.21} \downarrow	150.61	1.99 _{1.92} \times	0.52	51.30 _{10.32} \downarrow	314.40	3.61 _{1.76} \times	0.55
Tokenskip (Xia et al., 2025)	0.9	91.13 _{1.04} \downarrow	277.21	3.62 _{1.06} \times	0.93	58.40 _{3.22} \downarrow	525.91	5.85 _{1.09} \times	0.92
	0.7	85.69 _{6.48} \downarrow	223.83	2.87 _{1.33} \times	0.75	52.00 _{9.62} \downarrow	417.30	4.62 _{1.37} \times	0.73
	0.5	83.47 _{8.70} \downarrow	157.58	2.07 _{1.85} \times	0.53	47.10 _{14.52} \downarrow	297.25	3.30 _{1.92} \times	0.52
UCoT	0.9	91.72 _{0.45} \downarrow	244.88	3.32 _{1.15} \times	0.85	60.10 _{1.52} \downarrow	491.61	5.62 _{1.13} \times	0.86
	0.7	87.98 _{4.19} \downarrow	194.63	2.57 _{1.49} \times	0.65	58.80 _{2.82} \downarrow	388.72	4.37 _{1.45} \times	0.68
	0.5	86.55 _{5.62} \downarrow	140.36	1.86 _{2.06} \times	0.48	53.70 _{7.92} \downarrow	280.10	3.17 _{2.00} \times	0.49
<i>Llama3.1-8B Series</i>									
Original	-	87.26 _{0.00} \downarrow	212.13	2.44 _{1.00} \times	-	42.70 _{0.00} \downarrow	574.28	6.88 _{1.00} \times	-
Prompt (Xia et al., 2025)	0.9	87.32 _{0.06} \uparrow	207.89	2.40 _{1.02} \times	0.98	42.60 _{0.10} \downarrow	528.34	6.75 _{1.02} \times	0.92
	0.7	86.79 _{0.47} \downarrow	205.88	2.33 _{1.05} \times	0.97	42.30 _{0.40} \downarrow	534.08	6.92 _{0.99} \times	0.93
	0.5	87.16 _{0.10} \downarrow	189.32	2.15 _{1.13} \times	0.89	42.50 _{0.20} \downarrow	533.79	6.89 _{1.00} \times	0.93
Truncation (Xia et al., 2025)	0.9	62.55 _{24.71} \downarrow	193.91	2.22 _{1.10} \times	0.91	37.30 _{5.40} \downarrow	511.10	6.72 _{1.02} \times	0.89
	0.7	27.48 _{59.78} \downarrow	148.49	1.72 _{1.42} \times	0.70	35.00 _{7.70} \downarrow	413.48	4.52 _{1.52} \times	0.72
	0.5	7.32 _{79.94} \downarrow	108.19	1.25 _{1.95} \times	0.51	12.40 _{30.3} \downarrow	304.37	3.89 _{1.77} \times	0.53
CoD (Xu et al., 2025a)	0.9	85.82 _{1.44} \downarrow	197.28	2.28 _{1.07} \times	0.93	39.60 _{3.10} \downarrow	545.57	6.57 _{1.05} \times	0.95
	0.7	83.90 _{3.36} \downarrow	154.85	1.84 _{1.33} \times	0.73	36.70 _{6.00} \downarrow	413.48	4.96 _{1.39} \times	0.72
	0.5	73.46 _{13.8} \downarrow	110.31	1.27 _{1.92} \times	0.52	29.80 _{12.9} \downarrow	321.60	3.88 _{1.77} \times	0.56
Tokenskip (Xia et al., 2025)	0.9	85.79 _{1.47} \downarrow	197.16	2.37 _{1.03} \times	0.94	40.10 _{2.60} \downarrow	511.12	6.15 _{1.12} \times	0.89
	0.7	82.57 _{4.69} \downarrow	161.21	1.86 _{1.31} \times	0.76	37.90 _{4.80} \downarrow	413.48	4.95 _{1.39} \times	0.72
	0.5	81.80 _{3.46} \downarrow	135.76	1.56 _{1.56} \times	0.64	36.80 _{5.90} \downarrow	327.34	4.01 _{1.72} \times	0.57
UCoT	0.9	86.32 _{0.94} \downarrow	180.31	2.19 _{1.11} \times	0.85	40.50 _{2.20} \downarrow	499.62	6.07 _{1.13} \times	0.87
	0.7	84.17 _{3.09} \downarrow	140.01	1.73 _{1.41} \times	0.66	39.50 _{3.20} \downarrow	396.25	4.80 _{1.43} \times	0.69
	0.5	83.62 _{3.64} \downarrow	101.82	1.26 _{1.94} \times	0.48	38.00 _{4.70} \downarrow	298.63	3.79 _{1.82} \times	0.52

Table 1: Performance comparison of the CoT compression task under four compression ratio settings. For the accuracy and end-to-end latency, we use color \uparrow and color \downarrow respectively to mark the performance changes after compression compared with those of the original model. In addition, we mark the optimal performance indicators under each compression ratio setting with color \times .

reasoning performance of reasoning models; end-to-end latency reflects the reasoning efficiency of reasoning frameworks; the number of compressed CoT tokens and the actual compression ratio reflect the execution effect of the method under different compression ratio settings.

5.1 Performance Comparison

We report the experiment results in Table 1. We find that although current methods struggle to maintain performance with shorter reasoning paths, UCoT finds a balance between effectiveness and efficiency. Specifically, the Prompt method fails to faithfully compress the CoT to the specified compression ratio, leading to good performance on accuracy using a similar length with the original CoT. The Truncation method exhibits the opposite characteristics. Although it achieves the given compression rate, the substantial information loss during direct truncation causes a significant drop

in model performance. CoD and Tokenskip are stronger competitors, however, they are still inferior in terms of performance and efficiency to our method. For example, when the compression ratio is set to 0.5, UCoT reduces the average CoT length of the model by 50.75%, while the average model accuracy is improved by 5.63% and 3.23% compared with CoD and Tokenskip, respectively. This is because CoD and Tokenskip impair the model’s reasoning by imposing rigid output constraints or fine-tuning on external short CoT datasets without information enrichment. In contrast, UCoT provides high-quality contextual soft tokens at the input stage, naturally guiding executor to reduce generation by leveraging the reasoning priors. This demonstrates the superiority of our compressor-executor framework: the system achieves optimal efficiency not by limiting the model, but by post-reasoning paradigm where the compressor provides the contextual CoT and the executor focuses on the

Methods	Types	GSM8K				MATH-500			
		Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓	Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓
Qwen2.5-7B Series									
Original	-	92.17 _{0.00↓}	298.63	3.83 _{1.00×}	-	61.60 _{0.00↓}	571.64	6.35 _{1.00×}	-
UCoT	with all	87.98 _{4.19↓}	194.63	2.57 _{1.49×}	0.65	58.80 _{2.82↓}	388.72	4.37 _{1.45×}	0.68
	w/o ST	73.23 _{18.94↓}	220.99	2.97 _{1.29×}	0.74	47.50 _{14.12↓}	390.84	4.40 _{1.44×}	0.68
	w/o \mathcal{L}_{sem}	87.32 _{4.85↓}	274.74	3.74 _{1.02×}	0.92	59.70 _{1.92↓}	554.49	6.32 _{1.00×}	0.97
	w/o \mathcal{R}	71.53 _{15.73↓}	206.05	2.86 _{1.34×}	0.69	44.90 _{16.72↓}	417.30	4.73 _{1.34×}	0.73
Llama3.1-8B Series									
Original	-	87.26 _{0.00↓}	212.13	2.44 _{1.00×}	-	42.70 _{0.00↓}	574.28	6.88 _{1.00×}	-
UCoT	with all	84.17 _{3.09↓}	140.01	1.73 _{1.42×}	0.66	39.50 _{3.20↓}	396.25	4.80 _{1.43×}	0.69
	w/o ST	70.20 _{17.06↓}	150.61	1.89 _{1.29×}	0.71	25.70 _{17.00↓}	407.74	4.87 _{1.41×}	0.71
	w/o \mathcal{L}_{sem}	86.49 _{0.77↓}	199.40	2.37 _{1.03×}	0.94	40.30 _{2.40↓}	522.59	6.42 _{1.07×}	0.91
	w/o \mathcal{R}	67.58 _{19.68↓}	144.25	1.80 _{1.41×}	0.68	22.80 _{19.90↓}	389.49	4.82 _{1.43×}	0.69

Table 2: Performance comparison of ablation study.

Methods	Types	GPQA				HumanEval			
		Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓	Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓
Qwen3-8B									
Original	-	60.32 _{0.00↓}	8252.18	542.70 _{1.00×}	-	51.83 _{0.00↓}	1967.30	142.67 _{1.00×}	-
Tokenskip (Xia et al., 2025)	0.9	58.72 _{1.60↓}	7683.74	504.61 _{1.08×}	0.93	48.75 _{3.08↓}	1845.66	144.83 _{0.99×}	0.93
	0.7	57.79 _{2.53↓}	6213.35	408.25 _{1.33×}	0.75	46.13 _{5.70↓}	1462.13	108.91 _{1.31×}	0.73
	0.5	54.23 _{6.09↓}	4388.54	294.63 _{1.84×}	0.53	42.52 _{9.31↓}	1025.00	78.18 _{1.82×}	0.52
UCoT	0.9	59.30 _{1.02↓}	7097.48	469.22 _{1.16×}	0.86	48.33 _{3.50↓}	1761.89	126.63 _{1.13×}	0.89
	0.7	58.27 _{2.05↓}	5647.24	374.03 _{1.45×}	0.68	48.12 _{3.71↓}	1523.82	112.86 _{1.26×}	0.77
	0.5	56.86 _{3.46↓}	4065.58	277.23 _{1.96×}	0.49	46.55 _{5.28↓}	1021.93	78.84 _{1.81×}	0.52
Deepseek-R1-Distill-Qwen-7B									
Original	-	55.53 _{0.00↓}	9076.19	323.63 _{1.00×}	-	49.16 _{0.00↓}	1765.04	46.83 _{1.00×}	-
Tokenskip (Xia et al., 2025)	0.9	53.79 _{1.74↓}	8347.48	300.96 _{1.08×}	0.92	47.32 _{1.84↓}	1589.36	45.92 _{1.02×}	0.90
	0.7	52.88 _{2.65↓}	6626.87	242.99 _{1.33×}	0.73	44.03 _{5.13↓}	1275.88	39.36 _{1.19×}	0.72
	0.5	51.50 _{4.03↓}	5247.19	190.54 _{1.70×}	0.58	43.79 _{5.37↓}	900.040	27.88 _{1.68×}	0.51
UCoT	0.9	54.62 _{0.91↓}	7624.00	275.92 _{1.17×}	0.84	48.94 _{0.22↓}	1542.48	43.56 _{1.08×}	0.87
	0.7	52.97 _{2.56↓}	6361.32	232.54 _{1.39×}	0.70	45.76 _{3.40↓}	1227.76	37.72 _{1.24×}	0.69
	0.5	52.10 _{3.37↓}	4635.69	172.13 _{1.88×}	0.51	43.96 _{5.20↓}	870.68	25.12 _{1.86×}	0.49

Table 3: Performance comparison of extra long CoT.

result. In Appendices C.1–C.3 and C.5–C.6, we further analyze the impacts of compressor type, executor scale and prompt robustness, alongside investigations into UCoT case studies.

5.2 Ablation Study

To comprehensively investigate the influence of individual components within UCoT on its final performance, we conduct ablation experiments for soft tokens, semantic loss \mathcal{L}_{sem} , and reward factor \mathcal{R} under the compression ratio of 0.7. We remove each part separately during the executor training, specifically: (1) **w/o. ST** removes soft tokens from the input to the executor. (2) **w/o. \mathcal{L}_{sem}** trains the executor without semantic loss. (3) **w/o. \mathcal{R}** removes the reward factor \mathcal{R} from \mathcal{L}_e .

As shown in Table 2, incorporating soft tokens as part of the input significantly improved the reasoning performance of executor during training. This indicates that the contextual CoT output by the compressor can provide effective reasoning guid-

ance for executor. Moreover, compared to the complete version, **w/o. \mathcal{L}_{sem}** achieve higher reasoning performance for executor in some settings, but it cannot effectively complete the compression task for CoT, which is similar to the research results of (Xu et al., 2025b). This reveals our speculation in Section 4.2 about the importance of guiding large language models to fully utilize soft tokens. Furthermore, the reward factor \mathcal{R} further enhances the accuracy of the executor, which indicates that it effectively promotes the retention of the original executor’s reasoning ability.

5.3 Applicability to Extra Long CoT

Current large reasoning models are capable of generating reasoning paths spanning thousands of tokens to tackle complex tasks, which poses challenges to the effectiveness of CoT compression methods. To evaluate whether UCoT still performs well in compressing extra-long CoT, we test it on Qwen3-8B (Yang et al., 2025) and Deepseek-R1-

Distill-Qwen-7B (Guo et al., 2025) on two representative datasets for cover complex commonsense reasoning and code generation: GPQA (Rein et al., 2024) and HumanEval (Chen et al., 2021).

Experimental results in Table 3 show that even when dealing with complex data and in cases where R1 models adopt ultra-long CoTs, UCoT can still significantly improve their inference efficiency. For instance, when UCoT is applied to the Deepseek-R1-Distill-Qwen-7B model, the usage of inference tokens on the HumanEval dataset is reduced by 50.67%, while the performance is improved by 0.17% compared with Tokenskip, the current SOTA method. Similar improvements are observed in the Qwen3-8B model: UCoT reduces the model’s inference token usage on the dataset by 48.05% and improves its performance by 4.03% compared with SOTA method Tokenskip. This highlights UCoT’s ability to balance efficiency and performance. Additional evaluations on AIME2024 (Finkelstein et al., 2024) and ASDiv (Miao et al., 2020), covering mathematical reasoning tasks, are presented in Appendix C.4.

5.4 Effectiveness of Contextual CoT

We look into the reasons for the effectiveness of soft tokens from the perspective of information volume with two metrics proposed in Kuratov et al. (2025), namely **Token Gain** and **Information Gain**. Specifically, Token Gain describes the difference in the number of tokens that can be correctly decoded with and without the presence of continuous embeddings. In UCoT, this can be described as:

$$\mathcal{C}_{tokens} = \mathcal{C}_{tokens}^{\mathcal{M}_c+H_n} - \mathcal{C}_{tokens}^{\mathcal{M}_c}, \quad (11)$$

where $\mathcal{C}_{tokens}^{\mathcal{M}_c}$ and $\mathcal{C}_{tokens}^{\mathcal{M}_c+H_n}$ represent the number of tokens that can be correctly decoded by \mathcal{M}_c before and after adding soft tokens H_n . Information Gain measures the change in Cross-Entropy of the target text by calculating the output uncertainty eliminated by the language model before and after the addition of soft tokens H_n . In our work, Information Gain can be defined as:

$$\mathcal{H}_{ce} = \mathcal{H}_{ce}^{\mathcal{M}_c} - \mathcal{H}_{ce}^{\mathcal{M}_c+H_n}, \quad (12)$$

where $\mathcal{H}_{ce}^{\mathcal{M}_c+H_n}$ and $\mathcal{H}_{ce}^{\mathcal{M}_c}$ represent the Cross-Entropy value before and after adding H_n .

We control the information in soft tokens by early stopping compressor training at different step, then measure both the information volume

Metrics	Values	Acc	Tokens	Latency	Latency
Original	-	92.17 _{0.00↓}	298.63	3.83 _{1.00×}	-
Token	72.35	70.68 _{21.49↓}	218.00	2.83 _{1.35×}	0.73
Gain↑	123.62	81.45 _{10.72↓}	200.08	2.62 _{1.46×}	0.67
	182.45	86.55 _{5.62↓}	200.39	2.61 _{1.47×}	0.67
Information	841.07	67.38 _{24.79↓}	212.13	2.83 _{1.35×}	0.71
Gain↑	1077.82	75.95 _{16.22↓}	206.05	2.72 _{1.41×}	0.69
	1863.45	85.55 _{6.62↓}	197.10	2.65 _{1.45×}	0.66

Table 4: Token gain and information gain of UCoT.

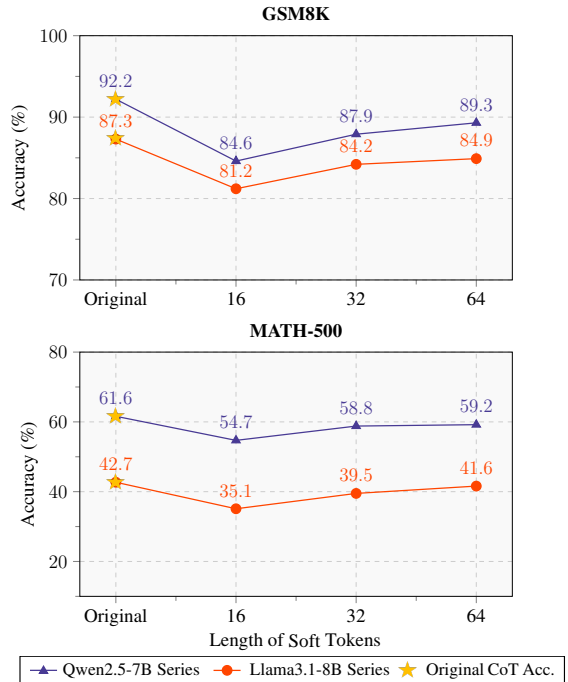


Figure 4: Performance comparison between original CoT and soft tokens of different lengths.

of contextual CoT and UCoT performance. The results with the Qwen2.5-7b-Instruct model on the GSM8K dataset are reported in Table 4. The positive correlation between the information volume in soft tokens and the performance of UCoT suggests that our method compresses rich reasoning information into the embedding space of LLMs, allowing the model to maintain performance with a shorter reasoning path.

5.5 Analysis of Length of Contextual CoT

Recent literature on prompt compression suggests that the information density of compressed soft tokens is highly sensitive to their allocated length (Mu et al., 2023). We explore the association between the length of soft tokens and the model performance in this section. As shown in Figure 4 the results indicate that as the length of the soft tokens gradually increases, the accuracy of the executor on various test datasets presents a significant upward

trend. Using Qwen2.5-7B-Instruct as the executor and inferencing the GSM8K dataset, the model’s accuracy improved by 4.75% when the soft tokens length increased from 16 to 64, gradually approaching that of the original CoT reasoning. This result fully demonstrates that as the length of the soft tokens increases, the compressor can provide more complete and accurate reasoning information for the executor.

5.6 Comparison with Other Latent Reasoning Methods

In this section, we present a comparative analysis of UCoT and existing latent reasoning methods. Specifically, latent reasoning methods for large models can be categorized into two types: one is the continuous token method, which converts CoT into continuous embedding vectors (Hao et al., 2024); the other is the loop layer method, which enables inputs to flow repeatedly through the same layer within the model (Geiping et al., 2025). We set three compression ratios (0.9, 0.7, and 0.5)

Methods	AIME 2024	GPQA	HumanEval	ASDiv
Original	66.67 _{0.00↓}	60.32 _{0.00↓}	51.83 _{0.00↓}	93.74 _{0.00↓}
Depth-Recurrent	50.00 _{16.7↓}	51.32 _{9.00↓}	34.36 _{17.5↓}	87.31 _{6.43↓}
Coconut	56.67 _{10.0↓}	55.42 _{4.90↓}	43.77 _{8.06↓}	88.26 _{5.48↓}
LightThinker	63.33 _{3.34↓}	57.92 _{2.40↓}	46.93 _{4.91↓}	93.42 _{0.32↓}
UCoT-0.9	66.67_{0.00↓}	59.30_{1.02↓}	48.33_{3.50↓}	93.53_{0.21↓}
UCoT-0.7	63.33 _{3.34↓}	58.27 _{2.05↓}	48.12 _{3.71↓}	93.32 _{0.42↓}
UCoT-0.5	56.67 _{10.0↓}	56.86 _{3.46↓}	46.55 _{5.28↓}	92.77 _{0.97↓}

Table 5: Comparison with other potential reasoning methods with Qwen3-8b backbone.

for UCoT, denoted as UCoT-0.9, UCoT-0.7, and UCoT-0.5 for brevity. We then compare UCoT with representative works of the three types of latent reasoning methods, namely Depth-Recurrent (Geiping et al., 2025), Coconut (Hao et al., 2024) and LightThinker (Zhang et al., 2025).

The results are presented in Table 5. Experimental findings indicate that under most settings, the performance of Depth-Recurrent and Coconut is at most comparable to that of UCoT-0.5. For instance, on the GPQA dataset, the accuracy rates of UCoT-0.5, Depth-Recurrent, and Coconut are 56.86%, 51.32%, and 55.42%, respectively. While LightThinker represents a formidable competitor, it nonetheless lags behind our method in maintaining performance stability. UCoT-0.9 achieves an average improvement of 1.56% over LightThinker across the four datasets.

5.7 Case study for Contextual CoT

In this section, we explore what specific information the soft token contains from original CoT. We decode the contextual CoT in UCoT into text sequences with the compressor established in Section 4.1 to inspect the exact information it provides, and present examples in Figure 5 and Figure 6 in Appendix C.5. Such an analysis allows us to verify whether the compressor successfully distills essential reasoning logic into the compressed representations. We find the text decoded from the soft tokens exhibits reasonable logical structure that closely mirrors the original CoT.

For instance, in the case of Math-500, they both follow a similar reasoning pattern: *fractional quantity derivation* → *total amount aggregation* → *remainder calculation*, with only specific numbers and objects varied. These structural cues within the soft tokens act as a cognitive template, guiding the executor to maintain logical consistency. This observation suggests that compressed contextual CoT in UCoT provides demonstration for how to organize reasoning, echoing the principles of in-context learning and recent findings on the importance of reasoning structure (Li et al., 2025).

6 Conclusion

In this work, we propose post-reasoning, a paradigm which enables shorter LLM outputs by providing CoT as context rather than generation. We investigate the factors that affect the model behavior in post-reasoning, and find the latency in contextual CoT generation and the quality of contextual CoT play critical roles in the efficiency and performance of post-reasoning. To achieve optimal efficiency-performance trade-off, we further introduce UCoT inspired by the properties of post-reasoning to enable efficient and informative contextual CoT generation. This method trains a lightweight language model as the compressor to produce compact contextual CoT in the form of soft tokens, preserving essential reasoning structure within a limited length. The resulting contextual CoT is then utilized by a large executor model to generate accurate response with reduced output length. Experiments demonstrate that our method outperforms existing comparative CoT compression methods across a range of compression ratios, and further analysis of the contextual CoT sheds light on the underlying mechanism by which UCoT preserves reasoning capability.

Limitations

Despite the promising results achieved by UCoT in the context of CoT compression, there are three primary limitations to this study. **First**, our method has only been evaluated on a limited set of open-source inference models, and its applicability to closed-source models remains unexplored. **Second**, the method applies a uniform compression ratio across all inputs and lacks the capability to adaptively adjust compression strategies based on the complexity or difficulty of individual reasoning problems. **Third**, due to computational resource constraints, we have not evaluated UCoT on larger-scale LLMs (e.g., Qwen2.5-32B series and Qwen2.5-72B series). While our results are promising, evaluating UCoT on larger-scale models remains an important next step. This would offer a more comprehensive validation of our method’s scalability in real-world scenarios.

Ethics Statement

UCoT aims to help improve the reasoning efficiency of existing LLMs. Our goal is to develop a general reasoning method that can be applied to current mainstream LLMs and to inspire progress towards more efficient LLM reasoning frameworks. We explicitly do not encourage any malicious use of our work, especially attempts to circumvent or compromise LLM systems. The artifacts and datasets in our work are all under the restriction of the license and follow the intended use.

Acknowledgment

We thank all the reviewers and the area chair for their helpful feedback, which aided us in greatly improving the paper. This work is supported by National Key R&D Program (2023YFB3107400), National Natural Science Foundation of China (62272371, 62103323, (62521002, U24B20185, U2441240, 62132011), Initiative Postdocs Supporting Program (BX20190275, BX20200270), China Postdoctoral Science Foundation (2019M663723, 2021M692565), and Fundamental Research Funds for the Central Universities under grant (xzy012024144, xzy012025043). Cong Wang was supported in part by Hong Kong Research Grants Council (RGC) under Grants C6015-23G and CRS_HKUST601/24. Thanks to the New Cornerstone Science Foundation, the Xplorer Prize and K. C. Wong Education Foundation.

References

- Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*.
- Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#).
- Sijia Chen, Baochun Li, and Di Niu. 2024. Boosting of thoughts: Trial-and-error problem solving with large language models. *arXiv preprint arXiv:2402.11140*.
- 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 e-prints*, arXiv:2110.14168.
- Yingqian Cui, Pengfei He, Jingying Zeng, Hui Liu, Xianfeng Tang, Zhenwei Dai, Yan Han, Chen Luo, Jing Huang, Zhen Li, and 1 others. 2025. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. *arXiv preprint arXiv:2502.13260*.
- Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, and 1 others. 2025. Competitive programming with large reasoning models. *arXiv preprint arXiv:2502.06807*.
- Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, and Shuvendu K Lahiri. 2024. Llm-based test-driven interactive code generation: User study and empirical evaluation. *IEEE Transactions on Software Engineering*.
- Joseph Finkelstein, Robert Moskovitch, and Enea Parimbelli. 2024. *Artificial Intelligence in Medicine: 22nd International Conference, AIME 2024, Salt Lake City, UT, USA, July 9–12, 2024, Proceedings, Part I*, volume 14844. Springer Nature.
- Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein.

2025. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Letman, and 1 others. 2024. [The Llama 3 Herd of Models](#). *arXiv e-prints*, arXiv:2407.21783.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.
- Yinhan He, Wendy Zheng, Yaochen Zhu, Zaiyi Zheng, Lin Su, Sriram Vasudevan, Qi Guo, Liangjie Hong, and Jundong Li. 2025. Semcot: Accelerating chain-of-thought reasoning through semantically-aligned implicit tokens. *arXiv preprint arXiv:2510.24940*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. [Measuring Mathematical Problem Solving With the MATH Dataset](#). *arXiv e-prints*, arXiv:2103.03874.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2025. C3ot: Generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24312–24320.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.
- Yurii Kuratov, Mikhail Arkhipov, Aydar Bulatov, and Mikhail Burtsev. 2025. Cramming 1568 tokens into a single vector and back again: Exploring the limits of embedding space capacity. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19323–19339.
- Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.
- Chenghao Li, Chaoning Zhang, Yi Lu, Jiaquan Zhang, Qigan Sun, Xudong Wang, Jiwei Wei, Guoqing Wang, Yang Yang, and Heng Tao Shen. 2025. Syzygy of thoughts: Improving llm cot with the minimal free resolution. *arXiv preprint arXiv:2504.09566*.
- Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhmaneshi, Shishir G. Patil, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. [LLMs Can Easily Learn to Reason from Demonstrations Structure, not content, is what matters!](#) *arXiv e-prints*, arXiv:2502.07374.
- Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. 2024. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971*.
- Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.
- Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984.
- Jesse Mu, Xiang Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352.
- Sania Nayab, Giulio Rossolini, Marco Simoni, Andrea Saracino, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. 2024. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*.
- Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. LlmLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *ACL (Findings)*.
- Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624.

- Sanjana Ramprasad, Elisa Ferracane, and Zachary C Lipton. 2024. Analyzing llm behavior in dialogue summarization: Unveiling circumstantial hallucination trends. *arXiv preprint arXiv:2406.03487*.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. **GPQA: A graduate-level google-proof q&a benchmark**. In *First Conference on Language Modeling*.
- Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. 2025. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*.
- Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2024. RL on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031.
- 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*.
- Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. 2025. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. *Proceedings of the International Conference on Learning Representations*.
- Jinyan Su, Jennifer Healey, Preslav Nakov, and Claire Cardie. 2025. Between underthinking and overthinking: An empirical study of reasoning length and correctness in llms. *arXiv preprint arXiv:2505.00127*.
- Jiashuo Sun, Yi Luo, Yeyun Gong, Chen Lin, Yelong Shen, Jian Guo, and Nan Duan. 2024. Enhancing chain-of-thoughts prompting with iterative bootstrapping in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4074–4101.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Yichen Wang, Shangbin Feng, Abe Bohan Hou, Xiao Pu, Chao Shen, Xiaoming Liu, Yulia Tsvetkov, and Tianxing He. 2024. **Stumbling Blocks: Stress Testing the Robustness of Machine-Generated Text Detectors Under Attacks**. *arXiv e-prints*, arXiv:2402.11638.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.
- Zongqian Wu, Baoduo Xu, Ruochen Cui, Mengmeng Zhan, Xiaofeng Zhu, and Lei Feng. Rethinking chain-of-thought from the perspective of self-training. In *Forty-second International Conference on Machine Learning*.
- 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*.
- Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025a. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.
- Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025b. SoftCoT: Soft chain-of-thought for efficient reasoning with llms. In *Proceedings of ACL*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Junchi Yu, Ran He, and Rex Ying. 2023. Thought propagation: An analogical approach to complex reasoning with large language models. *arXiv preprint arXiv:2310.03965*.
- Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2019. **PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization**. *arXiv e-prints*, arXiv:1912.08777.
- Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. **LightThinker: Thinking step-by-step compression**. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13307–13328, Suzhou, China. Association for Computational Linguistics.
- Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. In *The eleventh international conference on learning representations*.
- Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143.

Appendix

This paper’s appendices have three parts: Appendix A (experimental details), Appendix B (CoT compression definitions), and Appendix C (performance expansion analysis).

A Experiment Setting Details

A.1 Datasets

Table 6 provides an overview of the datasets used in this work, including their domain categories and the sizes of their training and test splits. These datasets span three major task types: mathematical reasoning (GSM8K, MATH, AIME, ASDiv), scientific question answering (GPQA), and code generation (HumanEval). Following standard evaluation protocols from prior work (Xia et al., 2025), we adopt task-specific settings for each dataset. For the MATH dataset in particular, we report results on the MATH-500 subset to reduce computational overhead, as it has been shown to serve as a reliable proxy for the full test set (Xia et al., 2025).

Datasets	Train	Test	Domain
GSM8K	7473	1319	Math Reasoning
Math	7500	500	Math Reasoning
AIME 2024	-	30	Math Reasoning
ASDiv	-	2305	Math Reasoning
GPQA	-	448	Scientific QA
HumanEval	-	164	Code Generation

Table 6: Datasets in the main experiments.

A.2 Baselines

We conduct extensive experiments comparing four major competitors, including representative CoT compression methods based on discrete prompts and compressed datasets.

Prompt (Xia et al., 2025): In this method, we use prompts to guide the LLM to reduce a fixed percentage of the output token in the CoT process. For example, append a piece of text to the sample, "Please reduce your thought process by 50 percent." as input to the sample. Compared with CoT compression methods based on discrete prompts that require a large number of discrete prompts to be designed or use small-shot learning, this method only needs a single text instruction and does not need to rely on example demonstration.

Truncation (Xia et al., 2025): In this method, the CoT generated by the LLM is forcibly truncated according to the preset compression ratio to limit the generation delay of the LLM.

CoD (Xu et al., 2025a): This method drafts the concise intermediate reasoning of CoT through extensive manual design, capturing only the essential information, and guides the LLM to produce a succinct reasoning process through few-shot learning.

Tokenskip (Xia et al., 2025): In this method, the LLMingua-2 (Pan et al., 2024) method with variable compression ratio is used to construct a short CoT dataset and perform supervised fine-tuning of LLM to realize the compression of CoT by LLM with different amplitudes.

A.3 Implementation Details

We provide experimental details for all baseline methods in the main experiment here. For the Prompt (Xia et al., 2025) and CoD (Xu et al., 2025a) methods, we show detailed prompts of these methods at different compression ratios in Table 7. In addition, for the few-shot part of the CoD prompt, we used the original design and do not change it. For Truncation (Xia et al., 2025), we forcibly truncate the full CoT generated by the LLM to the corresponding compression ratio, and then tested the performance of the LLM. For Tokenskip (Xia et al., 2025), we employed its provided short CoT dataset to fine-tune the LLM using LoRA (Hu et al., 2022), an efficient and widely validated method for LLM adaptation. The LoRA configuration used a rank $r = 8$ and scaling parameter $\alpha = 16$. Training was conducted for 3 epochs with a minibatch size of 4, using the AdamW (Loshchilov and Hutter, 2017) optimizer and a learning rate of 5×10^{-5} . All experimental results are the average results of 5 different random seeds on a single NVIDIA A100 GPU.

A.4 Training Details

In this subsection, we provide additional details for our method. To construct the training corpus for both the compressor and executor, we use Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct to directly generate the original CoTs by performing reasoning on the corresponding training datasets, without the need for an additional short CoT dataset. After preparing the training data, we train our compressor and executor modules in two separate

Prompt (Xia et al., 2025)	Ratio:0.9
system-prompt: You are a helpful assistant. Think step by step, reduce your thought process by 10 percent.	
Prompt (Xia et al., 2025)	Ratio:0.7
system-prompt: You are a helpful assistant. Think step by step, reduce your thought process by 30 percent.	
Prompt (Xia et al., 2025)	Ratio:0.5
system-prompt: You are a helpful assistant. Think step by step, reduce your thought process by 50 percent.	
CoD (Xu et al., 2025a)	Ratio:0.9
system-prompt: You are a helpful assistant. Think step by step, but only keep a minor draft for each thinking step, with 60 words at most. Return the answer at the end of the response after a separator. Put only the final answer inside <code>\boxed{...}</code> . format: Q:question A: answer few-shot: <demonstration>.	
CoD (Xu et al., 2025a)	Ratio:0.7
system-prompt: You are a helpful assistant. Think step by step, but only keep a minor draft for each thinking step, with 30 words at most. Return the answer at the end of the response after a separator. Put only the final answer inside <code>\boxed{...}</code> . format: Q:question A: answer few-shot: <demonstration>.	
CoD (Xu et al., 2025a)	Ratio:0.5
system-prompt: You are a helpful assistant. Think step by step, but only keep a minor draft for each thinking step, with 5 words at most. Return the answer at the end of the response after a separator. Put only the final answer inside <code>\boxed{...}</code> . format: Q:question A: answer few-shot: <demonstration>.	

Table 7: Prompt details of the main experiment in §5.1.

stages. In the Contextual CoT Generation stage, we use standard LoRA (Hu et al., 2022) to train a compressor with the Qwen2.5-1.5B as the backbone, setting the rank r to 8 and the scaling parameter α to 16. We adopt the AdamW to optimize both the compressor in 3 epochs of training, with a learning rate of 8×10^{-5} and a minibatch size of 16. In the Contextual CoT Utilization stage, the projector consists of two fully connected layers, $\omega_p^1 \in \mathbb{R}^{|H_c| \times |H_{middle}|}$ and $\omega_p^2 \in \mathbb{R}^{|H_{middle}| \times |H_e|}$, where $|H_c|$ and $|H_e|$ are the dimensions of the hidden layer of the compressor and executor respectively, and the size of the middle layer $|H_{middle}|$ is uniformly set to 2048 in our experiment. We also use standard LoRA (Hu et al., 2022) to add learnable parameters to the executor, with the rank r set to 16 and the scaling parameter α set to 32. We adopt AdamW to optimize both the executor and the projector in 3 epochs of training, with a learning rate of 3×10^{-5} and a minibatch size of 2. During the inference phase, the max length of tokens is set to 512 for GSM8K and 1024 for MATH, and to preserve the diversity of the executor output, we set the Temperature to 0.2 and Top - p to 0.9. Regarding the experimental setup, all experimental results are the average results of 5 different random seeds, and the experiments are conducted on a single NVIDIA A100 GPU. Moreover, in Table 8, we

show the prompts z_c and z_e used by the compressor and executor respectively.

Compressor System Prompt z_c

prompt: You are a helpful assistant. Analyze the question step-by-step, and and output the complete analysis process after the question.

Executor System Prompt z_e

prompt: You are a helpful assistant. Analyze the question step-by-step, and ensure your solution is highlighted within `\boxed{...}`.

Table 8: Prompt details of the UCoT.

A.5 Training and Evaluation Protocol

Since AIME, ASDiv, GPQA, and HumanEval do not have officially compiled standardized training datasets, their core function is positioned as model evaluation tools, we train the model solely on GSM8K and directly evaluate it on these four benchmarks without any domain-specific fine-tuning. This protocol naturally reflects a zero-shot generalization scenario and allows us to assess the method’s effectiveness across diverse tasks and domains. During inference, generation parameters are adjusted based on task complexity. We set the max-

imum output length to 512 for ASDiv and 20,480 for AIME, GPQA, and HumanEval. A temperature of 0.8 and top-p of 0.9 is used across all tasks, and all test results are based on pass@1, meaning that each sample is tested only once, and if the sample passes the test, the problem is considered solve.

A.6 Optimization Process

In the UCoT architecture, we meticulously design two crucial training stages: Contextual CoT Generation and Contextual CoT Utilization, as comprehensively illustrated in Algorithms 1 and 2, respectively. In addition, Algorithm 3 illustrates the reasoning process of UCoT.

Algorithm 1 Contextual CoT Generation

- 1: **Input:** compressor \mathcal{M}_c with learnable parameters θ_c , training dataset $\mathcal{D} = \{(Q_n, C_n, A_n)\}_{n=0}^{N-1}$, learning rate η , system prompt z_c , a sequence of placeholders s_{spec}
 - 2: Initialize learnable parameters θ_c with random values
 - 3: **while** not converged **do**
 - 4: **for** each sample (Q_n, C_n, A_n) in \mathcal{D} **do**
 - 5: Construct input sequence by Eq. (4)
 - 6: Get the soft tokens by Eq. (5)
 - 7: Compute loss \mathcal{L}_c by Eq. (6)
 - 8: Compute gradients $\nabla\theta_c = \frac{\partial\mathcal{L}_c}{\partial\theta_c}$
 - 9: Update parameters $\theta_c \leftarrow \theta_c - \eta\nabla\theta_c$
 - 10: **end for**
 - 11: **end while**
 - 12: **Output:** compressor \mathcal{M}_c with the trained parameters θ_c
-

Algorithm 3 Efficient Post-Reasoning

- 1: **Input:** trained projector \mathcal{M}_p , compressor \mathcal{M}_c with the trained parameters θ_c , executor \mathcal{M}_e with the trained parameters θ_e , testing dataset $\mathcal{D} = \{(Q_n)\}_{n=0}^{N-1}$, system prompt z_c and z_e , placeholders s_{spec}
 - 2: **for** each question q_n in \mathcal{D} **do**
 - 3: Construct input sequence $(z_c \oplus q_n \oplus s_{spec})$
 - 4: Get the soft tokens by Eq. (5)
 - 5: Get the output by Eq. (10)
 - 6: **end for**
 - 7: **Output:** the answer for every question
-

Algorithm 2 Contextual CoT Utilization

- 1: **Input:** projector \mathcal{M}_p , compressor \mathcal{M}_c with the trained parameters θ_c , executor \mathcal{M}_e with learnable parameters θ_e , training dataset $\mathcal{D} = \{(Q_n, C_n, A_n)\}_{n=0}^{N-1}$, learning rate η , system prompt z_c and z_e , a sequence of placeholders s_{spec}
 - 2: Initialize learnable parameters θ_e and \mathcal{M}_p with random values
 - 3: **while** not converged **do**
 - 4: **for** each sample (q_n, c_n, a_n) in \mathcal{D} **do**
 - 5: Construct input sequence by Eq. (4)
 - 6: Get the soft tokens by Eq. (5)
 - 7: Get the output \bar{C}_n by Eq. (7)
 - 8: Compute loss \mathcal{L}_{sem} by Eq. (8)
 - 9: Compute reward factor \mathcal{R} by Eq. (9)
 - 10: Compute loss $\mathcal{L}_e = \mathcal{L}_{sem} \cdot \mathcal{R}$.
 - 11: Compute gradients $\nabla\theta_e = \frac{\partial\mathcal{L}_e}{\partial\theta_e}$ and
 - 12: $\nabla\mathcal{M}_p = \frac{\partial\mathcal{L}_e}{\partial\mathcal{M}_p}$
 - 13: Update parameters $\theta_e \leftarrow \theta_e - \eta\nabla\theta_e$
 - 14: $\mathcal{M}_p \leftarrow \mathcal{M}_p - \eta\nabla\mathcal{M}_p$
 - 15: **end for**
 - 16: **end while**
 - 17: **Output:** executor \mathcal{M}_e with the trained parameters θ_e and \mathcal{M}_p
-

B Problem Formulation

This section briefly introduces definitions of Chain-of-Thought Compression.

Chain-of-Thought Compression. The chain-of-thought compression addresses the challenge posed by long reasoning paths in chain-of-thought reasoning by reducing the length of the reasoning path R while maintaining or even enhancing the reasoning performance. Specifically, given a question $Q = \{q_1, q_2, \dots, q_{|Q|}\}$ and compression ratio $\alpha \in (0, 1)$, chain-of-thought compression uses additional training parameters or prompt θ to guide model $\text{LLM}(\cdot)$ in generating a shorter reasoning path $\bar{C} = \{\bar{c}_1, \bar{c}_2, \dots, \bar{c}_{|\bar{C}|}\}$:

$$\bar{c}_{i+1} = \text{LLM}(\bar{C}_{\leq i}; Q; \theta), \quad (13)$$

while maintaining performance comparable to that of the original reasoning path C :

$$\begin{aligned} \text{LLM}(Q; \bar{C}; \theta) &\approx \text{LLM}(Q; C), \\ |\bar{C}|/|C| &= \alpha. \end{aligned} \quad (14)$$

Therefore, for a given dataset $\mathcal{D} = \{Q_n, A_n\}_{n=1}^N$, consisting of N question-answer pairs, the opti-

mization objective of CoT compression with compression ratio α can be expressed as:

$$\begin{aligned} \theta^* = \arg \min_{\theta} \mathbb{E}_{\mathcal{D}} [\mathbb{D}(\text{LLM}(Q; \bar{C}; \theta), \text{LLM}(Q; C))] \\ \text{s.t. } |\bar{C}|/|C| = \alpha, \end{aligned} \quad (15)$$

where $\mathbb{D}(\cdot)$ indicates the performance difference function of LLM before and after cot compression. In a nutshell, Eq.15 searches for parameters or prompt that minimize the difference in LLM inference performance at compression ratio α .

C Additional Evaluation and Discussion on UCoT Performance

C.1 Analysis of Compressor Type

We thoroughly explore the feasibility of applying different types of language models as compressors to the UCoT architecture. As shown in Table 9, we present the experimental results of using the Llama-3.2-1B-Instruct model as a compressor to perform the CoT compression task. The experimental result shows that under this setting, UCoT can still effectively reduce the length of the CoT output by the executor while maintaining the inference accuracy of the executor. When compared with the results of our main experiment (Table 1), UCoT still demonstrates significant competitive advantages over other comparative methods. The above results fully verify the adaptability of the compressor component in the UCoT architecture to different types of language models, indicating that this design has good generality and extensibility.

C.2 Analysis of Executor Scale

In the experiment of this section, we systematically explored the performance of the UCoT method when applied to executors of different scales. As shown in Table 10, we select Qwen2.5 models with four different parameter magnitudes of 1.5B, 3B, 7B, and 14B as executors and evaluated their performance on the GSM8K dataset. The experimental results indicate that the UCoT method can efficiently complete the compression of CoT for executors of different scales. Specifically, as the scale of the executor increases, the executor can better maintain its performance under the same compression ratio condition. Taking the Qwen2.5-14B model as an example, when the UCoT method reduces the token output by 48%, the accuracy of the model only decreases by 1.79%. This phenomenon fully demonstrates that a larger-scale Large Language

Model (LLM) can more effectively extract and utilize the key reasoning information in soft tokens, thus still maintaining a high performance level after compression.

C.3 Analysis of Prompt Robustness

Currently, the research on the robustness of the CoT compression method is still limited. In this part, we carry out two common perturbation attacks, Paragraphing (Wang et al., 2024) and Character-Substituted (Wang et al., 2024), on the input prompts of Tokenskip and UCoT, aiming to observe whether they can still effectively perform the CoT compression task³. As shown in Table 11 and Table 12, we conducted the experiment under the experimental setting with a compression ratio of 0.7. The experimental results indicate that when facing Paragraphing (Wang et al., 2024) and Character-Substituted (Wang et al., 2024) attacks, both UCoT and Tokenskip can complete the CoT compression task, but UCoT demonstrates better robustness. This indicates that UCoT method, leveraging continuous embeddings, can more effectively mitigate the impact of perturbation attacks, which enhances its viability for real-world deployment.

C.4 Performance comparison on AIME2024 and ASDiv Datasets

This section provides additional evaluations of two difficulty levels for mathematical reasoning tasks: AIME 2024 (Finkelstein et al., 2024) and ASDiv (Miao et al., 2020). Results in Table 13 show UCoT exhibits excellent CoT compression performance, whether on the complex AIME 2024 or simpler ASDiv. For example, when UCoT is applied to the deepseek-r1-distill-qwen-7b model, the usage of inference tokens on the ASDiv dataset is reduced by 45%, while the performance is improved by 2.25% over the SOTA method Tokenskip.

C.5 The Case Studies for Contextual CoT

We present a case study of soft tokens with Qwen2.5-7B-Instruct as executors on the GSM8K and Math in Figure 5 and Figure 6.

³Paragraphing method uses the Pegasus encoder (Zhang et al., 2019) to restructure input prompts. Character-Substituted method replaces part of the characters in the prompt according to predetermined rules. For simplicity, the initial settings of the two methods, as well as the perturbation attacks Paragraphing and Character-Substituted, are referred to and presented by I, P, and C in the relevant tables.

Methods	Ratio	GSM8K				MATH-500			
		Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓	Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓
Qwen2.5-7B Series									
Original	-	92.17 _{0.00↓}	298.63	3.83 _{1.00×}	-	61.60 _{0.00↓}	571.64	6.35 _{1.00×}	-
UCoT	0.9	90.58 _{1.59↓}	271.75	3.52 _{1.09×}	0.91	59.80 _{1.80↓}	485.91	5.53 _{1.15×}	0.85
	0.7	87.62 _{4.55↓}	206.05	2.71 _{1.41×}	0.69	58.40 _{3.20↓}	394.44	4.41 _{1.44×}	0.69
	0.5	86.55 _{5.62↓}	155.29	2.04 _{1.88×}	0.52	53.20 _{8.40↓}	285.82	3.18 _{2.00×}	0.50
Llama3.1-8B Series									
Original	-	87.26 _{0.00↓}	212.13	2.44 _{1.00×}	-	42.70 _{0.00↓}	574.28	6.88 _{1.00×}	-
UCoT	0.9	86.32 _{0.94↓}	186.65	2.21 _{1.10×}	0.88	40.30 _{2.40↓}	493.86	6.06 _{1.14×}	0.86
	0.7	83.97 _{3.29↓}	154.58	1.82 _{1.34×}	0.73	39.20 _{3.50↓}	384.77	4.73 _{1.45×}	0.67
	0.5	83.16 _{4.10↓}	114.52	1.42 _{1.72×}	0.54	37.30 _{5.40↓}	287.16	3.74 _{1.84×}	0.50

Table 9: Performance comparison of compressor type.

Methods	Ratio	GSM8K				Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓
		Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓				
Qwen2.5-1.5B Series									
Original	-	70.96 _{0.00↓}	317.14	3.67 _{1.00×}	-	83.62 _{0.00↓}	316.83	3.71 _{1.00×}	-
UCoT	0.9	68.77 _{2.19↓}	291.77	3.41 _{1.08×}	0.92	83.45 _{0.17↓}	275.64	3.40 _{1.09×}	0.87
	0.7	64.15 _{6.81↓}	228.34	2.75 _{1.33×}	0.72	81.52 _{2.10↓}	224.95	2.82 _{1.32×}	0.71
	0.5	61.58 _{9.38↓}	155.40	1.83 _{2.01×}	0.49	80.67 _{2.95↓}	164.75	2.07 _{1.79×}	0.52
Qwen2.5-7B Series									
Original	-	92.17 _{0.00↓}	298.63	4.28 _{1.00×}	-	93.12 _{0.00↓}	314.37	4.57 _{1.00×}	-
UCoT	0.9	91.72 _{0.45↓}	244.88	3.72 _{1.15×}	0.85	92.86 _{0.26↓}	279.79	3.63 _{1.26×}	0.89
	0.7	87.98 _{4.19↓}	194.63	2.83 _{1.51×}	0.65	91.72 _{1.40↓}	213.77	2.79 _{1.64×}	0.68
	0.5	87.55 _{4.62↓}	140.36	2.25 _{1.90×}	0.48	91.33 _{1.79↓}	169.76	2.36 _{1.94×}	0.54

Table 10: Performance comparison of executor size.

Methods	Types	GSM8K				MATH-500			
		Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓	Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓
Qwen2.5-7B Series									
Original	-	92.17 _{0.00↓}	298.63	3.83 _{1.00×}	-	61.60 _{0.00↓}	571.64	6.35 _{1.00×}	-
Tokenskip (Xia et al., 2025)	I	85.69 _{6.48↓}	223.97	2.89 _{1.33×}	0.75	56.60 _{5.00↓}	417.29	4.72 _{1.35×}	0.73
	P	78.93 _{13.24↓}	215.01	2.79 _{1.37×}	0.72	52.70 _{8.90↓}	402.76	4.53 _{1.40×}	0.70
	C	80.11 _{12.06↓}	218.00	2.82 _{1.36×}	0.73	51.30 _{10.30↓}	394.43	4.50 _{1.41×}	0.69
UCoT	I	87.98 _{4.19↓}	194.11	2.65 _{1.45×}	0.65	58.80 _{2.80↓}	388.71	4.46 _{1.42×}	0.68
	P	84.62 _{7.55↓}	200.08	2.69 _{1.42×}	0.67	56.30 _{5.30↓}	405.86	4.53 _{1.40×}	0.71
	C	85.39 _{6.78↓}	203.17	2.71 _{1.41×}	0.68	56.70 _{4.90↓}	402.88	4.55 _{1.40×}	0.70

Table 11: Performance comparison of prompt robustness in Qwen2.5-7B series.

Methods	Types	GSM8K				MATH-500			
		Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓	Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓
Llama3.1-8B Series									
Original	-	87.26 _{0.00↓}	212.13	2.44 _{1.00×}	-	42.70 _{0.00↓}	574.28	6.88 _{1.00×}	-
Tokenskip (Xia et al., 2025)	I	82.57 _{4.69↓}	161.21	1.86 _{1.31×}	0.76	37.90 _{4.80↓}	413.48	4.95 _{1.39×}	0.72
	P	78.32 _{8.94↓}	150.62	1.79 _{1.36×}	0.71	34.70 _{8.00↓}	431.02	5.23 _{1.32×}	0.75
	C	79.68 _{7.58↓}	157.11	1.84 _{1.33×}	0.74	36.70 _{6.00↓}	442.26	5.32 _{1.29×}	0.77
UCoT	I	84.17 _{3.09↓}	140.01	1.72 _{1.42×}	0.66	39.50 _{3.20↓}	396.25	4.80 _{1.43×}	0.69
	P	82.55 _{4.71↓}	144.25	1.74 _{1.40×}	0.68	37.80 _{4.90↓}	402.00	4.97 _{1.38×}	0.70
	C	81.97 _{5.29↓}	146.37	1.74 _{1.40×}	0.69	39.10 _{3.60↓}	413.49	5.19 _{1.33×}	0.72

Table 12: Performance comparison of prompt robustness in Llama3.1-8B series.

C.6 Case Studies for UCoT

In this section, we present a case study of UCoT with Qwen2.5-7B-Instruct and Llama-3.1-8B-Instruct as executors on the GSM8K and Math

dataset. The experimental results are shown in Figures 7 to 10 where UCoT preserves the logical semantics in the original CoT prompts while significantly reducing the output length of the executor.

Methods	Types	AIME 2024				ASDiv			
		Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓	Acc. (%) ↑	Tokens ↓	Latency (s) ↓	ActRatio ↓
Qwen3-8B									
Original	-	66.67 _{0.00↓}	6357.13	418.07 _{1.00×}	-	93.74 _{0.00↓}	475.49	4.72 _{1.00×}	-
Tokenskip (Xia et al., 2025)	0.9	66.67 _{0.00↓}	5912.13	393.51 _{1.06×}	0.93	92.55 _{1.19↓}	446.96	4.44 _{1.06×}	0.94
	0.7	60.00 _{6.67↓}	4765.88	320.55 _{1.30×}	0.75	93.48 _{0.26↓}	361.37	3.59 _{1.31×}	0.76
	0.5	56.67 _{10.0↓}	4499.99	307.49 _{1.36×}	0.70	91.57 _{2.17↓}	294.83	2.93 _{1.61×}	0.62
UCoT	0.9	66.67 _{0.00↓}	5662.74	382.23 _{1.09×}	0.89	93.53 _{0.21↓}	413.67	4.12 _{1.15×}	0.87
	0.7	63.33 _{3.34↓}	4513.79	298.97 _{1.40×}	0.71	93.32 _{0.42↓}	304.31	3.03 _{1.56×}	0.64
	0.5	56.67 _{10.0↓}	4132.18	281.55 _{1.48×}	0.65	92.77 _{0.97↓}	257.76	2.57 _{1.84×}	0.54
Deepseek-R1-Distill-Qwen-7B									
Original	-	53.33 _{0.00↓}	1286.16	417.46 _{1.00×}	-	95.47 _{0.00↓}	468.94	5.48 _{1.00×}	-
Tokenskip (Xia et al., 2025)	0.9	46.67 _{6.66↓}	1208.24	402.41 _{1.04×}	0.94	95.18 _{0.29↓}	431.55	5.07 _{1.08×}	0.92
	0.7	43.33 _{10.0↓}	9774.88	320.44 _{1.30×}	0.76	94.76 _{0.71↓}	337.63	3.95 _{1.39×}	0.72
	0.5	43.33 _{10.0↓}	9260.35	286.88 _{1.46×}	0.72	94.33 _{1.14↓}	301.32	3.52 _{1.56×}	0.64
UCoT	0.9	50.00 _{3.33↓}	1093.34	342.96 _{1.22×}	0.85	95.43 _{0.04↓}	384.61	4.52 _{1.21×}	0.82
	0.7	46.67 _{6.66↓}	8875.30	307.05 _{1.36×}	0.69	96.01 _{0.54↑}	300.12	3.52 _{1.56×}	0.64
	0.5	43.33 _{10.0↓}	8231.43	274.18 _{1.52×}	0.64	96.58 _{1.11↑}	257.93	3.01 _{1.82×}	0.55

Table 13: Performance comparison of Qwen3-8B model and Deepseek-R1-Distill-Qwen-7B model on both AIME 2024 and AsDiv tasks.

Original CoT:

To determine the total number of beetles consumed per day within this ecosystem's food chain, we can follow these systematic steps: Calculate the total number of snakes eaten by the jaguars: There are 6 jaguars, and each jaguar consumes 5 snakes per day. Therefore, the total number of snakes consumed is calculated by: $5 \times 6 = 30$ snakes. Determine the total number of birds consumed based on the snake count: The problem establishes a dietary ratio of 3 birds for every 1 snake consumed. Given the total of 30 snakes, the number of birds consumed is: $30 \times 3 = 90$ birds. Calculate the final total of beetles consumed per day: For each bird identified in the previous step, there are 12 beetles consumed. The final quantity is determined by multiplying the bird count by the beetle-per-unit rate: $90 \times 12 = 1080$ beetles. Thus, the total number of beetles eaten per day in the forest is $\boxed{1080}$.

Decoded Soft Tokens:

Start by figuring out how many books each shelf holds: $12 \text{ books/row} * 5 \text{ rows} = \llangle 12 * 5 = 60 \gg 60$ books per shelf. Next, calculate the total books in the section: $60 \text{ books/shelf} * 7 \text{ shelves} = \llangle 60 * 7 = 420 \gg 420$ books total. Finally, determine how many magazines there are, given they're a third of the book count: $420 \text{ books} / 3 = \llangle 420 / 3 = 140 \gg 140$ magazines.

Figure 5: A case study on GSM8K dataset. Output using Qwen2.5-7B-Instruct as the executor and Qwen2.5-1.5B as the compressor. The structural information about reasoning that appear simultaneously in both the original CoT and the decoded soft tokens are highlighted in red ■.

Original CoT:

To determine the final number of fairies remaining in the forest after the sequence of events described, we can follow these systematic steps: Calculate the number of fairies seen by Katelyn's friend: Katelyn initially saw 50 fairies. Her friend saw half as many fairies as Katelyn did. Therefore, the number of fairies seen by the friend is calculated by: $\$1/2 \times 50 = 25\$$ fairies. Determine the total number of fairies currently in the forest: The total population is the sum of the fairies Katelyn saw and those her friend spotted. The current total number of fairies flying above the forest is: $\$50 + 25 = 75\$$ fairies. Calculate the final count after some fairies flew away: The problem states that thirty fairies subsequently left the area. The number of remaining fairies is determined by subtracting those who left from the previous total: $\$75 - 30 = 45\$$ fairies. Thus, the total number of fairies that remained in the forest is $\boxed{45}$.

Decoded Soft Tokens:

Liam's brother found $1/100 = \ll 1/100 = 10 \gg 10$, a third as many fireflies as Liam first spotted \n The total number of fireflies glowing in the garden now is $30 + 10 = \ll 30 + 10 = 40 \gg 40$ \n If fifteen fireflies then dimmed their lights and left, there were $40 - 15 = \ll 40 - 15 = 25 \gg 25$ fireflies that stayed.

Figure 6: A case study on Math dataset. Output using Qwen2.5-7B-Instruct as the executor and Qwen2.5-1.5B as the compressor. The structural information about reasoning that appear simultaneously in both the original CoT and the decoded soft tokens are highlighted in red ■.

Question

Kalinda is working on a 360 piece puzzle with her mom. Kalinda can normally add 4 pieces per minute. Her mom can typically place half as many pieces per minute as Kalinda. How many hours will it take them to complete this puzzle?

CoT (336)

To determine how long it will take Kalinda and her mom to complete the 360-piece puzzle, we need to follow these steps: 1. **Calculate Kalinda's rate of placing pieces:** Kalinda can place 4 pieces per minute. 2. **Calculate her mom's rate of placing pieces:** Her mom can place half as many pieces per minute as Kalinda. Therefore, her mom's rate is: $\frac{4}{2} = 2$ pieces per minute. 3. **Determine their combined rate:** To find their combined rate, we add Kalinda's rate and her mom's rate: $4 + 2 = 6$ pieces per minute. 4. **Calculate the total time required to complete the puzzle:** The puzzle has 360 pieces. At a combined rate of 6 pieces per minute, the time (t) required to complete the puzzle is given by: $t = \frac{360 \text{ pieces}}{6 \text{ pieces per minute}} = 60 \text{ minutes}$. 5. **Convert the time from minutes to hours:** Since there are 60 minutes in an hour, the time in hours is: $\frac{60 \text{ minutes}}{60 \text{ minutes per hour}} = 1$ hour. Thus, it will take them $\boxed{1}$ hour to complete the puzzle.

UCoT (218)

determine hours for Kalinda and mom to complete 360-piece puzzle follow steps: 1. **Calculate rate Kalinda adds pieces** Kalinda adds 4 pieces per minute. 2. **Calculate rate mom adds pieces** mom places half as many pieces per minute as Kalinda. {Rate of mom} = $\frac{4}{2} = 2$ pieces per minute. 3. **Determine combined rate adding pieces** Add rates Kalinda and mom {Combined rate} = $4 + 2 = 6$ pieces per minute. 4. **Calculate total time required complete puzzle** find time divide total number pieces by combined rate. {Total time (in minutes)} = $\frac{360 \text{ pieces}}{6 \text{ pieces per minute}} = 60 \text{ minutes}$. 5. **Convert total time from minutes to hours** 60 minutes in hour. {Total time (in hours)} = $\frac{60 \text{ minutes}}{60 \text{ minutes per hour}} = 1 \text{ hour}$ time for Kalinda and mom to complete puzzle is $\boxed{1}$ hour. The final answer is: $\boxed{1}$

Figure 7: A case study on GSM8K dataset. Output using Qwen2.5-7B-Instruct as the executor with the CoT compression ratio of 0.7. We highlight the number of CoT tokens in red within parentheses.

Question

Elliott is trying to walk 10,000 steps a day. He finished half of his steps on his walks to and from school and did another 1,000 steps going for a short walk with his friend. He also went for a short jog around the block and realized that after he had finished his jog, he only had 2,000 steps left to take. How many steps did Elliott take during his jog?

CoT (322)

Let's break down the problem step by step. 1. **Total Steps Goal**: Elliott aims to walk 10,000 steps a day. 2. **Steps Completed Before Jogging**: - He completed half of his steps on his walks to and from school. - This means he walked $\frac{10,000}{2} = 5,000$ steps to and from school. - Additionally, he took another 1,000 steps going for a short walk with his friend. - Therefore, the total steps taken before jogging is $(5,000 + 1,000 = 6,000)$ steps. 3. **Steps Left After Jogging**: - After his jog, he had 2,000 steps left to reach his goal of 10,000 steps. - This means he had already taken $(10,000 - 2,000 = 8,000)$ steps in total. 4. **Steps Taken During Jogging**: - The number of steps taken during his jog can be found by subtracting the steps taken before jogging from the total steps taken. - Thus, the steps taken during the jog is $(8,000 - 6,000 = 2,000)$ steps. Therefore, the number of steps Elliott took during his jog is $\boxed{2000}$.

UCoT (201)

determine steps Elliott took during jog follow information step by step. 1. **Total Steps Goal** Elliott's goal 10,000 steps a day. 2. **Steps Completed Half Steps** walked to and from school $\frac{10,000}{2} = 5,000$ steps. Additional 1,000 steps short walk with friend. Total steps completed before jog $5,000 + 1,000 = 6,000$ steps 3. **Remaining Steps After Jog** After jog Elliott has 2,000 steps left to take. 4. **Steps Taken During Jog** Subtract remaining steps from total steps goal find steps taken during jog: $10,000 - 2,000 = 8,000$ steps number of steps Elliott took during jog is $\boxed{8000}$. The final answer is: $\boxed{8000}$ The final answer is: $\boxed{8000}$

Figure 8: A case study on GSM8K dataset. Output using Llama-3.1-8B-Instruct as the executor with the CoT compression ratio of 0.7. We highlight the number of CoT tokens in red within parentheses.

Question

I have a bag with yellow and blue marbles in it. At the moment, the ratio of blue marbles to yellow marbles is 4:3. If I add 5 blue marbles and remove 3 yellow marbles, the ratio will be 7:3. How many blue marbles were in the bag before I added more?

CoT (448)

Let's denote the number of blue marbles initially as (B) and the number of yellow marbles initially as (Y) . Given that the initial ratio of blue to yellow marbles is 4:3, we can write: $\frac{B}{Y} = \frac{4}{3}$ This implies: $3B = 4Y$ Next, we are told that if we add 5 blue marbles and remove 3 yellow marbles, the new ratio becomes 7:3. So, the new number of blue marbles is $(B + 5)$ and the new number of yellow marbles is $(Y - 3)$. The new ratio can be written as: $\frac{B + 5}{Y - 3} = \frac{7}{3}$ This implies: $3(B + 5) = 7(Y - 3)$ Expanding and simplifying this equation, we get: $3B + 15 = 7Y - 21$ $3B + 36 = 7Y$ Now we have two equations: 1. $(3B = 4Y)$ 2. $(3B + 36 = 7Y)$ We can substitute $(3B)$ from equation (1) into equation (2): $4Y + 36 = 7Y$ Subtracting $(4Y)$ from both sides, we get: $36 = 3Y$ Dividing both sides by 3, we find: $Y = 12$ Now, substituting $(Y = 12)$ back into equation (1): $3B = 4 \times 12$ $3B = 48$ Dividing both sides by 3, we find: $B = 16$ Therefore, the number of blue marbles initially in the bag is $\boxed{16}$.

UCoT (287)

solve problem set equations based on given information define (b) number blue marbles (y) number yellow marbles initial ratio blue to yellow marbles 4:3 write equation $\frac{b}{y} = \frac{4}{3}$ relationship expressed as $3b = 4y$ after adding 5 blue marbles removing 3 yellow marbles new number blue marbles $(b + 5)$ new number yellow marbles $(y - 3)$ new ratio blue to yellow marbles 7:3 write equation $\frac{b + 5}{y - 3} = \frac{7}{3}$ relationship expressed as $3(b + 5) = 7(y - 3)$ simplify equation $3b + 15 = 7y - 21$ $3b + 36 = 7y$ solve system equations $3b = 4y$ $3b + 36 = 7y$ substitute $(3b = 4y)$ into second equation $4y + 36 = 7y$ $36 = 3y$ $y = 12$ substitute $(y = 12)$ back into first equation $3b = 4(12)$ $3b = 48$ $b = 16$ number blue marbles bag before adding more $\boxed{16}$. The final answer is: $\boxed{16}$

Figure 9: A case study on Math dataset. Output using Qwen2.5-7B-Instruct as the executor with the CoT compression ratio of 0.7. We highlight the number of CoT tokens in red within parentheses.

Question

Yann and Camille go to a restaurant. If there are 10 items on the menu, and each orders one dish, how many different combinations of meals can Yann and Camille order if they refuse to order the same dish? (It does matter who orders what---Yann ordering chicken and Camille ordering fish is different from Yann ordering fish and Camille ordering chicken.)

CoT (222)

To determine the number of different combinations of meals that Yann and Camille can order, given that they do not order the same dish, we can follow these steps: 1. **Identify the total number of choices for Yann:** There are 10 items on the menu. Therefore, Yann has 10 possible choices. 2. **Determine the number of choices for Camille after Yann's choice:** Since Camille cannot order the same dish as Yann, Camille has 9 remaining choices. 3. **Calculate the total number of combinations:** For each of Yann's 10 choices, Camille has 9 possible choices. Therefore, the total number of combinations is calculated by multiplying the number of choices for Yann by the number of choices for Camille: $10 \times 9 = 90$. Thus, the total number of different combinations of meals that Yann and Camille can order is $\boxed{90}$.

UCoT (146)

determine number different combinations meals Yann Camille order follow steps: 1. **Identify choices for Yann** Yann has 10 options choose any dish from 10 items on menu. 2. **Determine choices for Camille after Yann chosen dish** Camille cannot order same dish as Yann left with remaining 9 dishes. 3. **Calculate total number combinations multiply number choices Yann by number choices for Camille:** $\text{Total combinations} = 10 \times 9$ multiplication gives: $10 \times 9 = 90$ total number different combinations meals Yann Camille order is $\boxed{90}$. The final answer is: $\boxed{90}$ The final answer is: $\boxed{90}$

Figure 10: A case study on Math dataset. Output using Llama-3.1-8B-Instruct as the executor with the CoT compression ratio of 0.7. We highlight the number of CoT tokens in red within parentheses.