

BudgetThinker: Empowering Budget-aware LLM Reasoning with Control Tokens

Hao Wen*, Xinrui Wu*,[‡] Yi Sun, Feifei Zhang, Liye Chen[‡], Jie Wang[‡], Yifan Su, Yunxin Liu, Ya-Qin Zhang, Yuanchun Li[†]

Institute for AI Industry Research (AIR), Tsinghua University
Beijing 100084, China

wen-h22@mails.tsinghua.edu.cn; liyuanchun@air.tsinghua.edu.cn

* Equal contribution. [†] Corresponding author. [‡] Work done during an internship at AIR.

Abstract

Scaling test-time computation improves the reasoning ability of large language models (LLMs), but the resulting latency and cost limit its use in latency- and cost-sensitive scenarios. We present BudgetThinker, a framework for reasoning-budget adherence that provides online control of chain-of-thought (CoT) length during decoding. Rather than relying on a one-shot prompt instruction, BudgetThinker periodically inserts specialized tokens that communicate the remaining reasoning budget. A two-stage training pipeline, consisting of supervised fine-tuning and curriculum-based reinforcement learning, teaches the model to interpret these signals and adapt its reasoning pace. Across the evaluated 1.5B and 7B text backbones, a 7B multimodal backbone, and a 1.5B code-reasoning model evaluated on HumanEval, BudgetThinker improves budget adherence and accuracy–efficiency trade-offs over competitive baselines. The controlled comparisons isolate the contribution of online ratio signaling from the shared training pipeline. Code is available at <https://github.com/MobileLLM/BudgetThinker>.

1 Introduction

A recent breakthrough in Large Language Models (LLMs) is the paradigm of test-time scaling, where models are prompted to “think longer” by generating extended Chain-of-Thought (CoT) reasoning before producing a final answer (DeepSeek-AI, 2025; OpenAI, 2024). This approach has achieved state-of-the-art performance on complex reasoning tasks in domains such as advanced mathematics (DeepSeek-AI, 2025; Google, 2025), code generation (Hui et al., 2024; Yang et al., 2025a), and multimodal reasoning (Team et al., 2025a; Huang et al., 2025; Shen et al., 2025). However, these performance gains come at the cost of increased latency and computational overhead, making these reasoning models impractical for real-

world systems with strict performance constraints, such as latency-critical agents and robotics (Sui et al., 2025).

This fundamental trade-off between reasoning quality and computational efficiency highlights a critical need for *controllable CoT reasoning*—the ability for a reasoning LLM to conclude its reasoning process within a specified budget. Such a budget may be a hard latency limit imposed by an application, such as the sub-100 ms requirement for autonomous driving (Lin et al., 2018; Chen et al., 2025a), or a flexible constraint based on user tolerance for cost and delay in agentic systems. Because available resources vary across hardware and workloads, a model should complete reasoning within the requested limit while allocating more computation when the problem requires it (Cai et al., 2020; Yu et al., 2019). We term this instruction-following capability **budget adherence**.

Throughout this paper, B denotes the maximum number of model-generated reasoning tokens. Externally inserted control tokens and the fixed final-answer suffix of at most 50 tokens are not included in B . Thus, BudgetThinker targets adherence to a reasoning-token budget rather than strict end-to-end token control or full budget exhaustion.

However, existing approaches to shorten the reasoning length fall short of providing a per-step, fine-grained budget signal that LLMs have been explicitly trained to act on. Natural-language budget hints (whether placed in the prompt (Sun et al., 2025; Xu et al., 2025b) or inserted into the decoding stream (Pu et al., 2025)) rely on the base model’s zero-shot ability to understand numeric budget, without any mechanism to train the model to act on them. Reasoning-effort post-training (Yang et al., 2025a; OpenAI, 2025b,a; Team et al., 2025d; Yang et al., 2025b; Team et al., 2025a) teaches the model to allocate reasoning effort along a small set of discrete modes (e.g., “think-

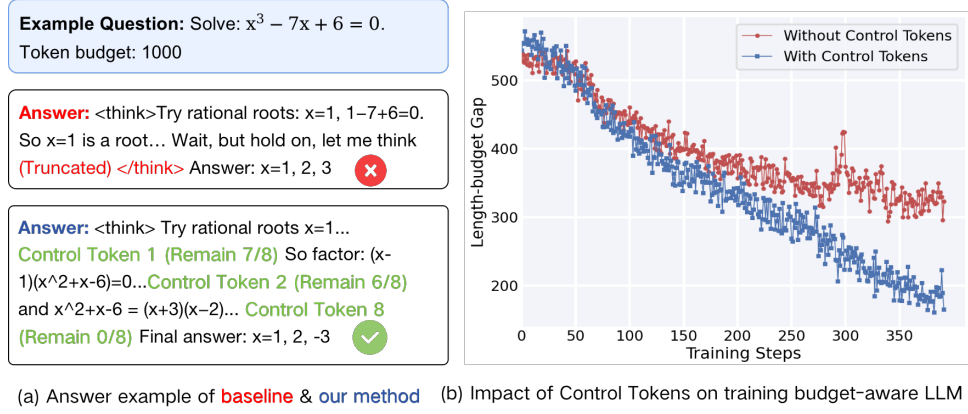


Figure 1: Overview of the BudgetThinker framework. (a) Budget-aware generation with Control Tokens: BudgetThinker periodically inserts special tokens into the reasoning process to continuously inform the model of the remaining token budget. (b) Effect of Control Tokens on training. We compare GRPO training of Qwen-2.5-1.5B with and without Control Tokens on the NuminaMath dataset (Project-Numina, 2024), measuring the gap between the generated length and the target budget. BudgetThinker reduces this length–budget gap more quickly and stably, indicating better budget adherence.

ing” vs. “non-thinking”), which cannot express the continuous, system-imposed token budgets we target. SFT- and RL-based pipelines (Hou et al., 2025; Aggarwal and Welleck, 2025; Wu et al., 2025d; Liu et al., 2025a; Xia et al., 2025; Qu et al., 2025; Yu et al., 2025c; Zhang et al., 2025a) train LLMs to internalize length preferences through reward shaping, but at inference time the model still receives no per-step indication of how much budget remains; consequently the gap between generated length and target budget plateaus around 300 tokens even after extensive training (Figure 1b).

In this paper, we try to address the challenge of fine-grained, dynamic control over the length of a model’s reasoning trace. We argue that merely stating a budget constraint in the initial prompt is insufficient, and the model needs to be continuously reminded of its remaining token budget as it generates reasoning tokens. To achieve this, we develop a training and inference framework that conditions the model on *online* budget signals. Specifically, we extend the LLM’s vocabulary with a small, fixed set of *learnable* special *Control Tokens* and periodically insert them into the decoding stream at budget milestones (e.g. every $1/K$ of reasoning budget B), as illustrated in Figure 1a. As shown in Figure 1b, training with these control tokens leads to a faster and more stable reduction in the gap between the generated length and the target budget, demonstrating their effectiveness in stimulating budget adherence.

We evaluate our method on DeepSeek-R1-distilled Qwen-2.5 1.5B and 7B models

(DeepSeek-AI, 2025), RoboBrain-7B-2.0 (Team et al., 2025a), and a 1.5B code-reasoning model on HumanEval (Chen et al., 2021). The evaluation covers MATH-500 (Hendrycks et al., 2021), AMC 2023, Embodied Reasoning QA Evaluation Dataset (ERQA) (Team et al., 2025b), and Spatial Aptitude Training Benchmark (SAT) (Ray et al., 2025). Across these evaluated scales and modalities, BudgetThinker improves the accuracy–budget trade-off and budget-following reliability over the original models and efficient-reasoning baselines. For the code experiment, we train both BudgetThinker and its no-control-token counterpart using the same two-stage SFT–RL pipeline, providing a controlled cross-domain test of the signaling mechanism.

Our main contributions are:

- We introduce a periodic, ratio-based budget-signaling mechanism that conveys the *remaining* budget to a reasoning LLM during decoding. The signal is carried by a compact set of K *learnable* control tokens, whose size is independent of the budget magnitude.
- We design a two-stage training pipeline—SFT followed by curriculum-based RL with a length-aware reward—that teaches the model to interpret these tokens as online budget signals rather than verbal hints. The pipeline is plug-and-play and complements existing test-time scaling procedures.
- Across the evaluated 1.5B and 7B text models, the 7B multimodal model, and the HumanEval code experiment, we show improved accuracy–

efficiency trade-offs and budget adherence under varied reasoning budgets.

2 Special-Token Conditioned Budget-Aware Reasoning

We present BudgetThinker, a budget-aware generation framework that guides Large Language Models (LLMs) to adhere to a predefined reasoning-token budget. Our method conditions the model on explicit control signals throughout generation. We enhance budget adherence by (i) introducing a dynamic control-token insertion strategy that continuously informs the model of the remaining budget and (ii) developing a two-stage training pipeline that combines Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL) to teach the model to interpret and respond to these signals.

2.1 Special-Token Budget Signaling

Our method augments standard autoregressive generation with a small, fixed set of special control tokens inserted at positions corresponding to **fractions of the reasoning budget**. These tokens signal how much of the requested reasoning budget has been consumed. Let B be the target budget and K be a fixed hyperparameter for the number of control signals. We introduce a set of K special control tokens, $\mathcal{C} = \{c_1, c_2, \dots, c_K\}$, where c_k signals that the generation has progressed through the k -th fraction of the total budget (e.g., c_1 signifies 10% completion if $K=10$). Here and throughout the paper, B is the maximum number of model-generated reasoning tokens. The K externally inserted control tokens and the fixed final-answer suffix of at most 50 tokens are excluded from B . Consequently, the total emitted sequence can exceed B by this fixed overhead; our objective is reasoning-budget adherence, not strict end-to-end token control or full budget exhaustion. During generation, the token y_t at timestep t is determined by the following piecewise function:

$$y_t = \begin{cases} c_k, & \text{if } t = k \cdot \lfloor B/K \rfloor, \\ & \text{for an integer } k \in \{1, \dots, K\} \\ y'_t, & \text{otherwise, where } y'_t \sim \pi_\theta(\cdot \mid x, y_{<t}) \end{cases}$$

At each timestep t that corresponds to the end of a budget fraction $k \cdot \lfloor B/K \rfloor$, we deterministically insert the corresponding control token c_k . For all other timesteps, the next token y'_t is sampled from the LLM’s distribution π_θ , conditioned on the full

history of both model-generated tokens and our inserted control tokens.

This **ratio-based signaling** is highly scalable. In contrast, a naive baseline like Fixed-Interval Signaling, which inserts a token every I steps, would require a potentially unbounded number of unique control tokens as the budget B increases. Our approach always uses the same fixed set of K tokens, making it robust and efficient across widely varying budgets.

2.2 SFT Data Curation

To teach the LLM the semantics of our control tokens, we construct a specialized dataset for Supervised Fine-Tuning (SFT). The curation process involves two key steps: budget assignment and dataset balancing.

Budget Assignment. First, for each original data sample (x, y) , we assign a budget B by rounding the answer’s length, $|y|$, up to the nearest multiple of a granularity parameter T :

$$B = T \cdot \left\lceil \frac{|y|}{T} \right\rceil,$$

This calculation method is intentional. By design, the true answer length $|y|$ is almost always less than the allocated budget $|B|$. This gap teaches the model to robustly terminate its reasoning and stop generating tokens once the answer is complete, rather than artificially padding the output to fill the entire budget. Next, we transform each sample in the dataset. The input prompt is modified to include the budget constraint, and the target output is reconstructed to include the control tokens:

$$\hat{x} = \text{Concat}(x, \text{Please answer within } B_i \text{ tokens})$$

$$\hat{y} = \text{Insert}(y, \{c_j\}_{j=1}^K)$$

Here, *Insert* adds the control token c_{k+1} at each ratio milestone $t = k \lfloor B/K \rfloor$. Then, we get a reconstructed dataset $\{(\hat{x}, \hat{y})\}$, which we use to fine-tune LLM.

2.3 Dataset Construction and Balancing

To ensure the model generalizes across a wide range of output lengths, our SFT dataset is a balanced mixture of three distinct data types:

- (1) **Long Reasoning Data:** Full-length CoT reasoning sequences from powerful teacher models, such as DeepSeek-R1 (DeepSeek-AI, 2025).
- (2) **Iteratively Compressed Reasoning Data:** To

populate intermediate lengths, we use an LLM-based compressor to iteratively shorten the long CoT sequences (Kang et al., 2025). Critically, this compression is applied iteratively: the result of one compression step becomes the input for the next. This process continues until a comprehensive spectrum of shortened reasoning chains, derived from an original long CoT, is produced. This approach ensures our model learns to adhere to budgets requiring diverse levels of reasoning abbreviation. **(3) Non-thinking Data:** Direct, concise answers to handle scenarios with very tight budget constraints where elaborate reasoning is not feasible. This balanced composition is pivotal for developing a versatile and robustly budget-adherent model.

2.4 Reinforcement Learning

Following SFT, we employ Group Relative Policy Optimization (GRPO) algorithm (Shao et al., 2024) to train the model to generate responses that adhere to a specific token budget, B . Figure 2 illustrates the overall training pipeline: the SFT stage (top-left) grounds control-token semantics, and the RL stage (top-right) adapts reasoning across budgets via a curriculum followed by mixed-budget sampling. During each RL rollout (bottom), the reasoning trace is truncated at reasoning budget B to extract the answer for accuracy and format rewards, while the full rollout continues to length T for length-reward computation.

Reward Design. We design a composite reward function that balances factual accuracy with a penalty for deviating from the target length. For a generated response y , with ground truth y_{gold} and budget B , the reward R is defined as:

$$R(y, y_{\text{gold}}, B) = k_1 \mathbf{1}\{y = y_{\text{gold}}\} + k_2 \mathbf{1}\{\text{format}(y)\} + k_3 \max(1 - \gamma \delta^2, 0) \quad (1)$$

Indicator functions $\mathbf{1}\{y = y_{\text{gold}}\}$ and $\mathbf{1}\{\text{format}(y)\}$ are the original correctness and format check of GRPO, and k_1, k_2, k_3 are coefficients that control the relative importance of each part. The *Length Reward* incentivizes the model to use the budget efficiently. It is based on *normalized length deviation*, $\delta = \frac{||y| - B|}{B}$, which measures the fractional difference between the generated length $|y|$ and the target budget B . This deviation is modulated by an *asymmetric penalty coefficient*, γ :

$$\gamma = \begin{cases} 1 & \text{if } |y| \leq B \\ r & \text{if } |y| > B \end{cases}$$

Here, $r > 1$ is a hyperparameter that imposes a larger penalty for exceeding the budget than for falling short of it. This design encourages the model to generate responses that are correct and aligned with the requested length constraint.

Special Token Conditioned Sampling. Our framework’s special-token insertion is integrated directly into the GRPO sampling loop. For each training question, BudgetThinker injects control tokens at the appropriate budget fractions during generation. To provide a clear learning signal, the model completes its full reasoning rollout to calculate the length-based reward accurately. For answer extraction, however, the reasoning trace is truncated at B and followed by a final-answer suffix of at most 50 tokens. This protocol exposes the model to the consequences of its full rollout while evaluating adherence to the reasoning budget rather than imposing a strict end-to-end cap.

Mixed Budget Reinforcement Learning. To build robustness across budgets, RL proceeds in two stages: (1) *Curriculum Stage*: Our LLM is trained iteratively on k different token budgets. The curriculum proceeds by decreasing the budget across stages, following the sequence $B_1 > B_2 > \dots > B_n$. (2) *Randomized Stage*: Each Training batch samples a budget B_k uniformly from $B_{j=1}^n$. This mixed strategy prevents catastrophic forgetting and maintains the model’s proficiency across all learned budget levels.

3 Experiments

In this section, we present a comprehensive empirical evaluation of BudgetThinker. We investigate accuracy and reasoning-budget adherence under varied budgets (Section 3.2), including a code-domain evaluation. We then analyze control-token insertion strategies (Section 3.3.1), iterative training (Section 3.3.2), and reinforcement learning (Section 3.3.3).

3.1 Experiment Setup

LLM Training Details. We use DeepSeek-R1 (DeepSeek-AI, 2025) distilled Qwen-2.5-1.5B and Qwen-2.5-7B (Hui et al., 2024) as backbone models. For SFT, we create a dataset of 41k instances for text-based LLMs. The details are listed in Appendix A.3. We train all SFT models for 6 epochs

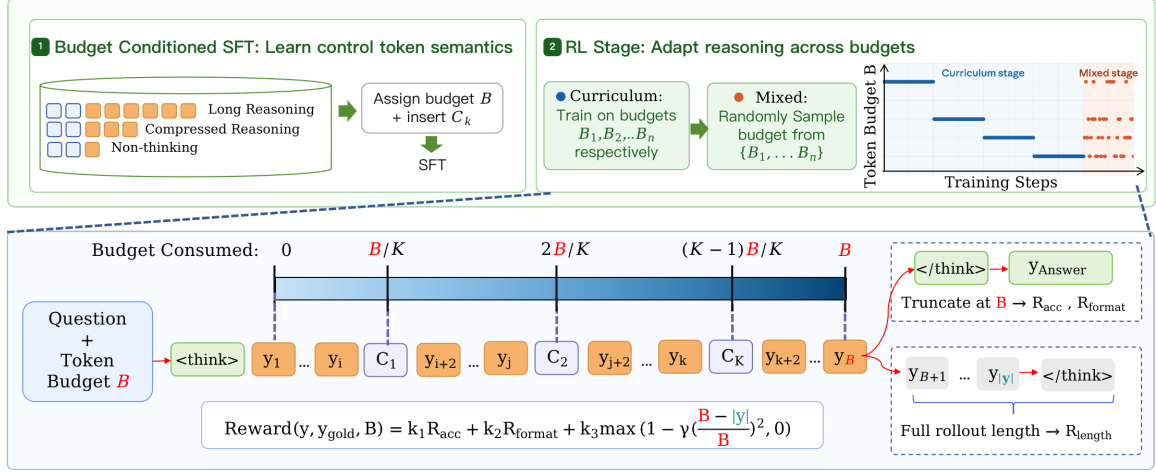


Figure 2: BudgetThinker training pipeline. Stage 1 (SFT) grounds control-token semantics using a mixture of long, compressed, and non-thinking reasoning data with assigned budgets. Stage 2 (RL) optimizes budget-conditioned reasoning via a curriculum of decreasing budgets ($B_1 > \dots > B_n$), followed by uniform mixed-budget sampling.

1.5B LLM											
Method	MATH-500						AMC				
	0.5K	1K	2K	4K	6K	10K	2K	4K	6K	10K	30K
ThinkPrune	44.0	69.4	79.4	82.8	83.3	83.6	65.5	70.1	71.9	72.0	73.6
Laser	48.4	70.2	80.2	82.6	83.8	83.2	59.0	67.8	73.4	75.9	75.9
AnytimeReasoner	51.0	66.8	76.0	83.4	84.8	87.4	57.2	68.8	72.8	74.0	75.0
Original Model	20.4	45.8	63.0	70.0	69.2	70.2	36.9	46.9	55.6	61.6	67.8
c2f	56.8	70.0	77.4	80.8	81.8	80.0	54.4	65.9	62.5	64.7	68.3
CoD	38.0	59.4	64.2	64.8	65.2	65.2	33.4	46.9	48.1	49.4	66.1
BudgetThinker	59.4	75.0	81.6	84.8	85.8	87.4	65.5	70.4	74.0	75.0	75.9
7B LLM											
Method	MATH-500						AMC				
	0.5K	1K	2K	4K	6K	10K	2K	4K	6K	10K	30K
ThinkPrune	52.8	75.4	85.6	90.0	90.6	91.6	65.6	82.5	83.4	87.5	90.0
Laser	53.8	80.0	88.8	90.6	92.0	92.2	67.8	83.8	86.3	87.5	88.1
Original Model	22.2	44.4	67.4	78.8	84.0	86.0	38.8	58.8	73.1	78.4	79.3
c2f	59.8	76.4	84.0	89.6	91.0	90.2	62.8	76.6	87.2	87.2	87.3
CoD	31.6	57.8	73.4	81.6	84.6	85.8	47.8	69.1	74.7	79.1	79.8
BudgetThinker	69.4	77.0	86.0	90.0	92.2	93.0	69.7	77.5	87.5	87.5	90.0
7B MLLM											
Method	ERQA					SAT					
	50	80	100	150	200	50	80	100	150	200	
c2f	4.5	6.5	10.0	24.6	39.6	23.0	55.0	66.0	69.0	74.6	
Laser	37.1	38.8	39.6	40.4	42.3	69.7	70.0	71.0	73.3	73.3	
BudgetThinker	41.1	41.9	43.6	43.4	43.6	70.7	72.0	72.7	74.0	76.7	

Table 1: Pass@1 accuracy of BudgetThinker vs. baselines across various generation budgets. The numeric headers (e.g., **0.5K**, ..., **30K**, **50**, ..., **80**) represent the Generation Budget.

with an initial learning rate of $2e-5$ and a maximum context length of 12000.

For Reinforcement Learning (RL), we employ the Group Relative Policy Optimization (GRPO) algorithm (Shao et al., 2024). Models are trained for 10 epochs with GRPO hyperparameters α and β set to 0.01 and 0.01, respectively. The maximum

context length during RL is 10000.

We configure BudgetThinker by setting the number of control intervals to $K = 8$. For the iterative training stage, we start from a budget of 6000, then reducing it to 4000, 3000, and finally 2000, with 1 epoch for each budget. In the randomized training stage, we randomly sample

$B \in \{6000, 4000, 3000, 2000\}$ for 1 epoch. Further training details are shown in Appendix A.7.

MLLM Training Details. We use RoboBrain-7B-2.0 (Team et al., 2025a) as the backbone model. For SFT, we use the visual question answering (VQA) data with CoT from RoboBrain-2.0, consisting of 37k question answer pairs. In RoboBrain-2.0, the CoT data is generated from GPT-4o (Hurst et al., 2024), and the length is concentrated in 200-300 tokens. Therefore, we use the GPT-4o-based compressor to compress the CoT length for 3 iterations, creating a dataset of 111k.

For RL, we randomly sample 2k instances from the VQA training dataset of RoboBrain-2.0 (Team et al., 2025a), train the MLLM for 6 epochs, with the initial learning rate of $1e-6$. Specifically, we set the number of control intervals to $K = 3$ because the shorter reasoning length on MLLMs, and the training budgets $B \in \{200, 150, 100, 80, 50\}$ for the iterative training phase, and train 1 epoch on each budget. Then, we train 1 epoch for mixed budget stage. We remain the other hyperparameters same with LLMs.

For fair comparison, we also train the original RoboBrain-7B-2.0 through the same procedure from SFT to RL using the same data and settings, with the difference that we don’t add budget constraints during training and set the maximum context length to 500, which is far beyond the largest budget 200 in our experiments.

Evaluation Details. We evaluate our methods and baselines on 2 math benchmarks and 2 multi-modal reasoning benchmarks: MATH-500 (Hendrycks et al., 2021), AMC 2023, ERQA (Team et al., 2025b), and SAT (Ray et al., 2025). To assess budget control, we evaluate performance across a range of reasoning budgets, from $B = 50$ to $B = 10000$ tokens. For inference, we utilize a modified version of the vLLM engine (Kwon et al., 2023). Our modified engine enforces the budget by truncating the reasoning process once the length limit is reached and appending the “</think>**Final Answer**” tags to signal the model for a final answer. We then allocate an additional 50 tokens for final-answer generation before terminating the process. Thus, B is a reasoning-token ceiling, while the fixed suffix and inserted control tokens contribute a small, separately accounted overhead. We additionally measure wall-clock latency under continuous batching with this setup; latency is approximately linear in the generated-token count (~ 5.2 ms/token on

a single A800 GPU), supporting the use of token budget as a practical proxy for end-to-end latency. Full latency table is reported in Appendix A.9

Code-domain evaluation. To evaluate code-domain reasoning, we train a 1.5B model on 31,619 examples from the AIZU subset of OpenCodeReasoning (Ahmad et al., 2025), followed by GRPO on 427 MBPP problems (Austin et al., 2021). We then evaluate the trained model on HumanEval (Chen et al., 2021). BudgetThinker and the No Control Tokens variant use identical data, training settings, and evaluation protocols; the only changed variable is the ratio-based online signaling mechanism. Detailed settings and pass@1/pass@8 results are reported in Appendix A.6.

Baselines. We benchmark BudgetThinker against five baselines: (1) **ThinkPrune** (Hou et al., 2025): A typical efficient reasoning framework that also uses GRPO for budget control but does not incorporate explicit control tokens. We compare ThinkPrune only at 1.5B, as its official repository does not provide a 7B model. (2) **Laser** (Liu et al., 2025a): A state-of-the-art efficient reasoning framework enhances both fast and slow thinking capability. (3) **AnytimeReasoner** (Qi et al., 2025): An RL-based anytime reasoning framework that optimizes performance under varying token budgets via truncated reasoning and dense rewards. We compare AnytimeReasoner only at 1.5B, as its official release does not provide 7B training scripts or weights. (4) **CoD (Chain-of-Draft)** (Xu et al., 2025b): A prompt based method that facilitates fast reasoning by prompting LLMs to draft concise intermediate thought. (5) **c2f** (Sun et al., 2025): A prompt-only baseline using the coarse-to-fine (c2f) prompt style of Sun et al. (2025), which instructs the model to first produce a brief coarse analysis with an initial answer before expanding into fine-grained reasoning. (6) **Original Models:** The DeepSeek-R1 distilled Qwen-2.5-1.5B and 7B models that serve as the foundation for our fine-tuning.

3.2 Accuracy-Budget Tradeoff

We first evaluate the accuracy of BudgetThinker against the baselines across various token budgets, with results shown in Table 1. For results of MATH-500 and AMC 2023, BudgetThinker outperforms the original model and ThinkPrune across most budget allocations. Specifically, BudgetThinker improves accuracy by 2.1% over Laser,

3.2% over ThinkPrune and 22.4% over Original model on average.

On HumanEval, BudgetThinker improves pass@1 and pass@8 at five of six budgets. Averaged across all budgets, the gains over No Control Tokens are 2.61 and 0.71 percentage points, respectively, with the clearest pass@1 improvements between 1K and 4K tokens (Appendix A.6). This controlled cross-domain result provides evidence that online budget signaling contributes beyond the shared SFT and RL pipeline.

The accuracy gains of BudgetThinker coincide with more reliable use of the requested reasoning budget, as shown in Figure 3. We calculate the average relative length $|y|/B$ only for responses that terminate naturally within B ; this is a within-budget length ratio, not a measure of full budget exhaustion. The original model, lacking budget-specific training, often violates the requested limit. ThinkPrune (Hou et al., 2025) follows the limit more often but can terminate prematurely on complex problems. In contrast, BudgetThinker achieves higher natural-completion and budget-following rates, while using more of the available computation than the evaluated baselines at larger budgets. It does not force every response to consume the full budget: at $B = 10,000$, its average reasoning length is 3,691 tokens (36.9% of B). These results support improved reasoning-budget adherence and accuracy–efficiency trade-offs, rather than strict end-to-end control or full budget exhaustion. Additional metrics appear in Appendix A.10, and failure-mode analyses appear in Appendix A.11.

3.3 Ablations

To identify which components drive BudgetThinker’s performance, we conduct controlled ablations isolating three design choices: (i) the signaling mechanism (Section 3.3.1), where we fix the full training pipeline and vary only the token insertion strategy, further analyzing sensitivity to the number of control intervals K ; (ii) the curriculum training schedule (Section 3.3.2), evaluated via intermediate checkpoints; and (iii) the reinforcement learning stage (Section 3.3.3), comparing the full model against an SFT-only variant.

3.3.1 Ablation on Control Token Insertion Strategies

We analyze the performance of BudgetThinker under four distinct control token insertion strategies

to understand their impact:

- **No Control Tokens:** An ablation where no special tokens are inserted, relying solely on the RL objective for budget control.
- **Fixed Interval (50 tokens):** Control tokens are inserted at a fixed, frequent interval of every 50 tokens, with 200 special tokens added for a maximum budget of 10000 tokens.
- **Fixed Interval (250 tokens):** Control tokens are inserted at a larger, fixed interval of every 250 tokens, with 40 special tokens added for a maximum budget of 10000 tokens.
- **Budget Ratio (Default):** Our proposed method, where 8 tokens are inserted at intervals corresponding to each 1/8th of the total budget.

Note that while the “No Control Tokens” and “Budget Ratio” strategies are inherently scalable to any context length, fixed-interval strategies require adjustments to the number of special tokens for different budget sizes.

The results in Figure 4 indicate that the **Budget Ratio** strategy yields the best performance. All variants use identical compressed-CoT data, SFT and RL data, curriculum schedules, reward functions, training steps, and evaluation protocols; the signaling mechanism is the only changed variable. Relative to No Control Tokens, Budget Ratio improves accuracy by 3.2, 4.0, and 4.6 points at $B = 1K, 2K$, and $10K$, respectively. The fixed-interval variants remain weaker, showing that the gain is not explained merely by inserting additional tokens; communicating the remaining budget ratio is important. The 250-token interval outperforms the 50-token interval, consistent with a guidance–interference trade-off.

We further analyze the sensitivity to the number of control intervals K in Table 3. Since K controls the frequency of ratio-based budget signals, it directly affects the trade-off between guidance and interference. Overall, $K = 8$ achieves the best performance across most MATH-500 budgets. Smaller values such as $K = 2$ provide insufficient feedback for pacing, whereas larger values such as $K = 16$ may over-insert control tokens and disrupt reasoning. Therefore, we use $K = 8$ as the default setting for LLM experiments. This is an empirical operating point, not a theoretically optimal constant; semantic-boundary placement is left for future work.

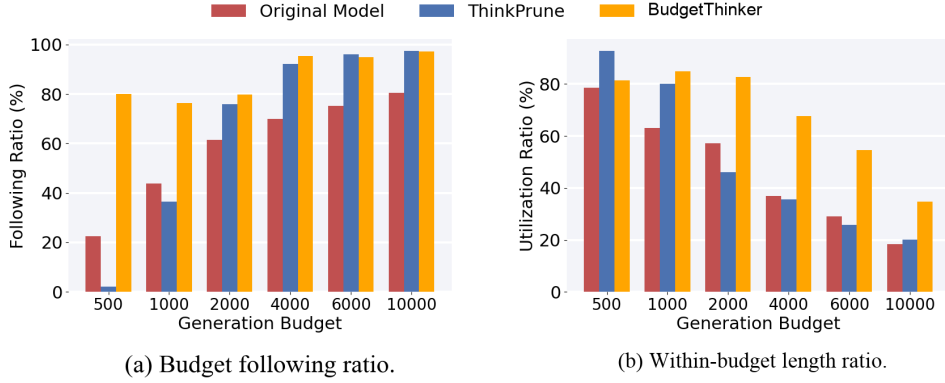


Figure 3: Budget following and within-budget length analysis of BudgetThinker-1.5B, ThinkPrune-1.5B, and the R1-1.5B on MATH-500. (a) Percentage of responses that terminate naturally within the reasoning budget B . (b) Average reasoning length as a percentage of B , computed only over responses that terminate naturally within B . The fixed final-answer suffix and externally inserted control tokens are excluded.

Model	Accuracy (%)				Budget Following Ratio (%)			
	2K	4K	6K	10K	2K	4K	6K	10K
BudgetThinker 6k	79.6	85.2	86.0	84.6	47.0	67.8	76.2	85.6
BudgetThinker 6k-4k	80.0	86.0	84.6	86.4	62.8	78.6	85.8	93.8
BudgetThinker 6k-4k-3k	80.0	84.2	85.0	87.2	66.4	84.6	89.8	95.4
BudgetThinker 6k-4k-3k-2k	83.6	84.6	84.6	85.2	86.8	95.4	97.2	98.6
BudgetThinker (Full Training)	81.6	84.8	85.8	87.4	79.8	95.4	95.0	97.2

Table 2: Accuracy (%) and Budget Following Ratio (%) of BudgetThinker-1.5B at different stages of the iterative training. Each row represents a checkpoint (*e.g.*, “6k-4k” denotes the model after training sequentially on 6k and 4k budgets). The “Full Training” model completes the 6k-to-2k curriculum plus a final mixed-budget training phase where budgets are randomly sampled. Darker cells indicate higher values within each column.

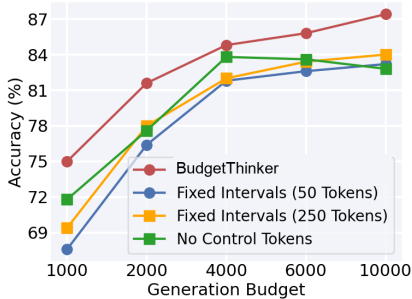


Figure 4: Ablation of control token insertion strategies.

3.3.2 Ablation on Iterative Training

To investigate the impact of our mixed budget reinforcement learning, we evaluate checkpoints of BudgetThinker-1.5B on MATH-500, as shown in Table 2. The results reveal that as the model is trained on smaller budgets, its reasoning capability on larger budgets initially decreases. However, after the final mixed-budget training stage, accuracy on larger budgets recovers. We also observed that training on smaller budgets improves the budget following ratio, encouraging the model to generate more concise answers. After the full curriculum,

Budget	$K = 2$	$K = 4$	$K = 8$	$K = 16$
500	55.8	60.6	59.4	60.0
1000	66.2	70.4	75.0	67.0
2000	77.4	77.2	81.6	75.6
4000	81.8	78.6	84.8	79.4
6000	81.2	83.0	85.8	77.6
10000	83.8	85.6	87.4	83.2

Table 3: Sensitivity analysis of the number of control intervals K on MATH-500.

the LLM achieves a balanced capability to handle all budgets, even if it is not individually optimal for every single budget constraint.

3.3.3 Ablation on Reinforcement Learning

To isolate the contribution of reinforcement learning, we compare the full BudgetThinker model with a version trained only with Supervised Fine-Tuning (SFT). The results in Figure 5 demonstrate that the full RL-tuned model consistently surpasses the SFT-only model across most tested budgets. This underscores the importance of the RL phase. While SFT teaches the model the format and semantic meaning of control tokens, RL encourages the model to actively explore and dis-

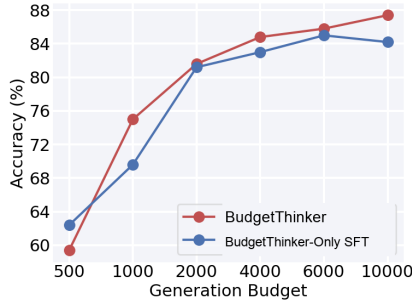


Figure 5: Ablation of reinforcement learning method.

cover more effective reasoning strategies that optimize the reward under specific budget constraints, ultimately leading to higher accuracy.

4 Related Work

Test-time scaling improves LLM reasoning through extended chain-of-thought generation (DeepSeek-AI, 2025; Google, 2025; Yang et al., 2025a; OpenAI, 2024), but the resulting latency motivates a growing line of works on efficient and controllable reasoning (Liu et al., 2025b; Wu et al., 2025a; Wang et al., 2024a). Prior approaches include prompt-level brevity instructions, adaptive switching between thinking modes, fine-tuning or RL objectives that penalize length. These methods either provide coarse control over average length, or rely on zero-shot instruction following that degrades over long decoding. In contrast, BudgetThinker trains LLMs to interpret a fixed, budget-agnostic set of ratio-based control tokens as online signals of remaining budget, enabling fine-grained adherence across widely varying budget levels. Detailed discussions in Appendix A.2.

5 Conclusion

We introduce BudgetThinker, a framework for reasoning-budget adherence based on periodic ratio-based control tokens and a two-stage SFT-RL training pipeline. Across the evaluated 1.5B and 7B text models, a 7B multimodal model, and a 1.5B code-reasoning model evaluated on HumanEval, BudgetThinker improves budget following and accuracy-efficiency trade-offs over the evaluated baselines. The controlled ablation shows that these gains persist when the data, curriculum, reward, and training compute are held fixed, isolating the contribution of online remaining-budget signaling. BudgetThinker improves completion within a requested reasoning

budget, but it does not require every response to exhaust that budget.

Limitations

Our evaluation covers 1.5B and 7B text models, one 7B multimodal model, and one 1.5B code-reasoning model. Scalability to substantially larger backbones, broader domains, and more multimodal tasks remains to be verified. The reasoning budget excludes the K inserted control tokens and the fixed final-answer suffix, so our protocol is not strict end-to-end token control and does not guarantee full budget exhaustion. The choices of K (8 for LLMs and 3 for MLLMs), $T = 50$, and the one-quarter-budget reward cutoff are practical empirical settings rather than theoretically optimal constants. We use ratio-based placement at token milestones and have not evaluated insertion at semantically meaningful reasoning boundaries. Finally, under very small budgets, BudgetThinker can still omit necessary constraints or commit too early to a compressed strategy, as shown by the failure-mode analysis in Appendix A.11.

References

- Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. *Preprint*, arXiv:2503.04697.
- Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. 2025. Opencodereasoning: Advancing data distillation for competitive coding. In *Second Conference on Language Modeling*.
- Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.
- Jiale Chen, Duc Van Le, Yuanchun Li, Yunxin Liu, and Rui Tan. 2025a. Timelynet: Adaptive neural architecture for autonomous driving with dynamic deadline. *ACM Trans. Embed. Comput. Syst.*, 24(5s).

- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025b. Do not think that much for $2+3=?$ on the overthinking of o1-like llms. *Preprint*, arXiv:2412.21187.
- Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, et al. 2026. Process reinforcement through implicit rewards. *Transactions on Machine Learning Research*.
- Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. 2024. Learning how hard to think: Input-adaptive allocation of lm computation. *Preprint*, arXiv:2410.04707.
- DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.
- Razvan-Gabriel Dumitru, Darius Peteleaza, Vikas Yadav, and Liangming Pan. 2025. Conciserl: Conciseness-guided reinforcement learning for efficient reasoning models. *Preprint*, arXiv:2505.17250.
- Yu Fei, Yasaman Razeghi, and Sameer Singh. 2025. Nudging: Inference-time alignment of LLMs via guided decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12702–12739, Vienna, Austria. Association for Computational Linguistics.
- Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. 2025a. Efficiently scaling llm reasoning with certindex. *Preprint*, arXiv:2412.20993.
- Yichao Fu, Xuwei Wang, Yuandong Tian, and Jiawei Zhao. 2025b. Deep think with confidence. *arXiv preprint arXiv:2508.15260*.
- 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. *CoRR*, abs/2502.05171.
- Google. 2025. Gemini 2.5: Our most intelligent ai model. <https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking>.
- Hasan Abed Al Kader Hammoud, Kumail Alhamoud, Abed Hammoud, Elie Bou-Zeid, Marzyeh Ghassemi, and Bernard Ghanem. 2025. Train long, think short: Curriculum learning for efficient reasoning. *Preprint*, arXiv:2508.08940.
- Jinyi Han, Ying Huang, Ying Liao, Zishang Jiang, Xikun Lu, Haiquan Zhao, Xinyi Wang, Guanghao Zhou, Sihang Jiang, Jiaqing Liang, Weikang Zhou, Zeye Sun, Fei Yu, and Yanghua Xiao. 2025. Your models have thought enough: Training large reasoning models to stop overthinking. *Preprint*, arXiv:2509.23392.
- Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.
- Qianyu He, Siyu Yuan, Xuefeng Li, Mingxuan Wang, and Jiangjie Chen. 2025. Thinkdial: An open recipe for controlling reasoning effort in large language models. *Preprint*, arXiv:2508.18773.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.
- Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. 2025. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning. *Preprint*, arXiv:2504.01296.
- Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. 2025. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.
- Neel Jain, Aditya Shrivastava, Chenyang Zhu, Daben Liu, Alfie Samuel, Ashwinee Panda, Anoop Kumar, Micah Goldblum, and Tom Goldstein. 2025. Refusal tokens: A simple way to calibrate refusals in large language models. In *Second Conference on Language Modeling*.
- Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2025. C3ot: generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in*

- Artificial Intelligence*, AAAI’25/IAAI’25/EAAI’25. AAAI Press.
- Eunki Kim, Sangryul Kim, and James Thorne. 2025. Learning to insert [PAUSE] tokens for better reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 23760–23777, Vienna, Austria. Association for Computational Linguistics.
- 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*.
- Bespoke Labs. 2025. Bespoke-stratos: The unreasonable effectiveness of reasoning distillation. <https://www.bespokelabs.ai/blog/bespoke-stratos-the-unreasonable-effectiveness-of-reasoning-distillation>. Accessed: 2025-01-22.
- Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, Kai Jia, and Zhifang Sui. 2026a. Selfbudgeter: Adaptive token allocation for efficient llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 21135–21156.
- Zhong-Zhi Li, Xiao Liang, Zihao Tang, Lei Ji, Peijie Wang, Haotian Xu, Haizhen Huang, Weiwei Deng, Yeyun Gong, Zhijiang Guo, et al. 2026b. Too long, do re-weighting for efficient llm reasoning compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 39943–39962.
- 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. *Preprint*, arXiv:2305.20050.
- Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the twenty-third international conference on architectural support for programming languages and operating systems*, pages 751–766.
- Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng Zhang. 2024. Can language models learn to skip steps? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng Liu, Yuntian Deng, Yizhe Zhang, and Junxian He. 2025a. Learn to reason efficiently with adaptive length-based reward shaping. *arXiv preprint arXiv:2505.15612*.
- Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025b. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.
- Yichuan Ma, Linyang Li, Yongkang Chen, Peiji Li, Xiaozhe Li, Qipeng Guo, Dahua Lin, and Kai Chen. 2026. Timely machine: Awareness of time makes test-time scaling agentic. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4619–4636.
- Minjia Mao, Bowen Yin, Yu Zhu, and Xiao Fang. 2025. Early stopping chain-of-thoughts in large language models. *Preprint*, arXiv:2509.14004.
- Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.
- Tergel Munkhbat, Namgyu Ho, Seohyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. 2025. Self-training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*.
- OpenAI. 2024. Openai o1 system card.
- OpenAI. 2025a. Gpt-5 system card. <https://cdn.openai.com/gpt-5-system-card.pdf>. Accessed: 2025-08-21.
- OpenAI. 2025b. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.
- Project-Numina. 2024. Numinamath. <https://huggingface.co/datasets/AI-MO/NuminaMath-CoT>.
- Xiao Pu, Michael Saxon, Wenye Hua, and William Yang Wang. 2025. Thoughtterminator: Benchmarking, calibrating, and mitigating overthinking in reasoning models. *arXiv preprint arXiv:2504.13367*.
- Penghui Qi, Zichen Liu, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Optimizing anytime reasoning via budget relative policy optimization. *arXiv preprint arXiv:2505.13438*.
- Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. 2025. Optimizing test-time compute via meta reinforcement finetuning. In *Workshop on Reasoning and Planning for Large Language Models*.
- Arijit Ray, Jiafei Duan, Ellis L Brown II, Reuben Tan, Dina Bashkurova, Rose Hendrix, Kiana Ehsani, Aniruddha Kembhavi, Bryan A. Plummer, Ranjay Krishna, Kuo-Hao Zeng, and Kate Saenko. 2025. SAT: Dynamic spatial aptitude training for multi-modal language models. In *Second Conference on Language Modeling*.

- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.
- Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, et al. 2025. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*.
- Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2026. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *Preprint*, arXiv:2503.04472.
- Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.
- Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*.
- Yi Sun, Han Wang, Jiaqiang Li, Jiacheng Liu, Xiangyu Li, Hao Wen, Yizhen Yuan, Huiwen Zheng, Yan Liang, Yuanchun Li, and Yunxin Liu. 2025. An empirical study of llm reasoning ability under strict output length constraint. *arXiv preprint arXiv:2504.14350*.
- Huajie Tan, Yuheng Ji, Xiaoshuai Hao, Xiansheng Chen, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. 2026. Reason-rft: Reinforcement finetuning for visual reasoning of vision language models. *Advances in neural information processing systems*, 38:5772–5822.
- Siao Tang, Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2025. Concisehint: Boosting efficient reasoning via continuous concise hints during generation. *arXiv preprint arXiv:2506.18810*.
- BAAI RoboBrain Team, Mingyu Cao, Huajie Tan, Yuheng Ji, Xiansheng Chen, Minglan Lin, Zhiyu Li, Zhou Cao, Pengwei Wang, Enshen Zhou, et al. 2025a. Robobrain 2.0 technical report. *arXiv preprint arXiv:2507.02029*.
- Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. 2025b. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*.
- Kimi Team. 2025a. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.
- Kimi Team et al. 2025c. Kimi k1.5: Scaling reinforcement learning with llms. *Preprint*, arXiv:2501.12599.
- Kwai Keye Team, Biao Yang, Bin Wen, Changyi Liu, Chenglong Chu, Chengru Song, Chongling Rao, Chuan Yi, Da Li, Dunju Zang, et al. 2025d. Kwai keye-vl technical report. *arXiv preprint arXiv:2507.01949*.
- NovaSky Team. 2025b. Sky-t1: Train your own o1 preview model within \$450. <https://novasky-ai.github.io/posts/sky-t1>. Accessed: 2025-01-09.
- Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. 2024a. Reasoning in token economies: Budget-aware evaluation of LLM reasoning strategies. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 19916–19939, Miami, Florida, USA. Association for Computational Linguistics.
- Siyuan Wang, Enda Zhao, Zhongyu Wei, and Xiang Ren. 2025a. Stepwise informativeness search for efficient and effective llm reasoning. *Preprint*, arXiv:2502.15335.
- Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025b. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. *Preprint*, arXiv:2408.13457.
- Xinyi Wang, Lucas Caccia, Oleksiy Ostapenko, Xingdi Yuan, William Yang Wang, and Alessandro Sordani. 2024b. Guiding language model reasoning with planning tokens. In *First Conference on Language Modeling*.
- Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. Parthinker: Native parallel thinking as a new paradigm to scale llm test-time compute. *arXiv preprint arXiv:2509.04475*.
- Sherman Wong, Zhenting Qi, Zhaodong Wang, Nathan Hu, Samuel Lin, Jun Ge, Erwin Gao, Wenlin Chen, Yilun Du, Minlan Yu, et al. 2025. Confucius code agent: Scalable agent scaffolding for real-world codebases. *arXiv preprint arXiv:2512.10398*.
- Chao Wu, Baoheng Li, Mingchen Gao, Yu Tian, and Zhenyi Wang. 2025a. From efficiency to adaptivity: A deeper look at adaptive reasoning in large language models. *arXiv preprint arXiv:2511.10788*.
- Menghua Wu, Cai Zhou, Stephen Bates, and Tommi Jaakkola. 2025b. Thought calibration: Efficient and confident test-time scaling. *Preprint*, arXiv:2505.18404.
- Tong Wu, Chong Xiang, Jiachen T Wang, G Edward Suh, and Prateek Mittal. 2025c. Effectively controlling reasoning models through thinking intervention. *arXiv preprint arXiv:2503.24370*.

- Xingyu Wu, Yuchen Yan, Shangke Lyu, Linjuan Wu, Yiwen Qiu, Yongliang Shen, Weiming Lu, Jian Shao, Jun Xiao, and Yueting Zhuang. 2025d. Lapo: Internalizing reasoning efficiency via length-adaptive policy optimization. *arXiv preprint arXiv:2507.15758*.
- Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.
- Haotian Xu, Xing Wu, Weinong Wang, Zhongzhi Li, Da Zheng, Boyuan Chen, Yi Hu, Shijia Kang, Jiaming Ji, Yingying Zhang, Zhijiang Guo, Yaodong Yang, Muhan Zhang, and Debing Zhang. 2025a. Redstar: Does scaling long-cot data unlock better slow-reasoning systems? *Preprint*, arXiv:2501.11284.
- Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025b. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.
- An Yang, Anfeng Li, Baosong Yang, et al. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.
- Qi Yang, Bolin Ni, Shiming Xiang, Han Hu, Houwen Peng, and Jie Jiang. 2025b. R-4b: Incentivizing general-purpose auto-thinking capability in mllms via bi-mode annealing and reinforce learning. *arXiv preprint arXiv:2508.21113*.
- Wang Yang, Debargha Ganguly, Xinpeng Li, Chaoda Song, Shouren Wang, Vikash Singh, Vipin Chaudhary, and Xiaotian Han. 2026. Mid-think: Training-free intermediate-budget reasoning via token-level triggers. *arXiv preprint arXiv:2601.07036*.
- Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.
- Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable neural networks. In *International Conference on Learning Representations*.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. 2025a. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL <https://arxiv.org/abs/2503.14476>.
- Zhaojian Yu, Yinghao Wu, Yilun Zhao, Arman Cohan, and Xiao-Ping Zhang. 2025b. Z1: Efficient test-time scaling with code. *arXiv preprint arXiv:2504.00810*.
- Zishun Yu, Tengyu Xu, Di Jin, Karthik Abinav Sankararaman, Yun He, Wenxuan Zhou, Zhouhao Zeng, Eryk Helenowski, Chen Zhu, Sinong Wang, Hao Ma, and Han Fang. 2025c. Think smarter not harder: Adaptive reasoning with inference aware optimization. *Preprint*, arXiv:2501.17974.
- Wenhao Zeng, Yaoning Wang, Chao Hu, Yuling Shi, Chengcheng Wan, Hongyu Zhang, and Xiaodong Gu. 2025. Pruning the unsurprising: Efficient llm reasoning via first-token surprisal. *arXiv preprint arXiv:2508.05988*.
- Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, et al. 2025a. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*.
- Xiaoyun Zhang, Jingqing Ruan, Xing Ma, Yawen Zhu, Haodong Zhao, Hao Li, Jiansong Chen, Ke Zeng, and Xunliang Cai. 2025b. When to continue thinking: Adaptive thinking mode switching for efficient reasoning. *Preprint*, arXiv:2505.15400.
- Xuechen Zhang, Zijian Huang, Chenshun Ni, Ziyang Xiong, Jiasi Chen, and Samet Oymak. 2025c. Making small language models efficient reasoners: Intervention, supervision, reinforcement. *arXiv preprint arXiv:2505.07961*.
- Xuechen Zhang, Zijian Huang, Ege Onur Taga, Carlee Joe-Wong, Samet Oymak, and Jiasi Chen. 2024. Efficient contextual llm cascades through budget-constrained policy learning. In *Proceedings of the 38th International Conference on Neural Information Processing Systems, NIPS '24*, Red Hook, NY, USA. Curran Associates Inc.
- Kai Zhao, Yanjun Zhao, Jiaming Song, Shien He, Lusheng Zhang, Qiang Zhang, and Tianjiao Li. 2025. Saber: Switchable and balanced training for efficient llm reasoning. *Preprint*, arXiv:2508.10026.
- Jason Zhu and Hongyu Li. 2025. Towards concise and adaptive thinking in large reasoning models: A survey. *Preprint*, arXiv:2507.09662.

A Appendix

A.1 LLM Usage Statement

Large language models (e.g., ChatGPT) were used only for minor language polishing and grammar correction. All ideas, experimental design, data analysis, and writing were conducted by the authors.

A.2 Related Works

A.2.1 Test-Time Scaling for Reasoning LLMs

Recent advances in test-time scaling improve LLM reasoning by increasing computational depth during decoding. Among these methods, reinforcement learning enables LLMs to explore diverse solution strategies and allocate more computation to reflection, substantially improving reasoning across domains such as math (DeepSeek-AI, 2025; Google, 2025), coding (Hui et al., 2024;

Yang et al., 2025a), agentic tasks (Team, 2025a), and multimodal reasoning (Tan et al., 2026; Shen et al., 2025; Team et al., 2025b,a). Other works distill reasoning ability from long CoTs generated by large models into smaller LLMs to encourage deeper thinking (Muennighoff et al., 2025; Labs, 2025; Team, 2025b; Ye et al., 2025; Xu et al., 2025a; Geiping et al., 2025). Despite their effectiveness, these approaches often incur substantial inference latency because of lengthy outputs (Sun et al., 2025; Zhu and Li, 2025; Qu et al., 2025), limiting their practicality in real-time systems. Moreover, excessively long outputs can cause overthinking (*e.g.*, meaningless repetition) and even introduce errors (Yu et al., 2025a; Chen et al., 2025b). To improve efficiency, recent studies explore controlling reasoning effort by adaptively switching thinking modes (He et al., 2025; Zhao et al., 2025; Hammoud et al., 2025; Wu et al., 2025b; Zhang et al., 2025b; Mao et al., 2025; Han et al., 2025; Shen et al., 2026). Our method instead teaches LLMs to follow a user-specified token budget during reasoning, enabling more controlled and efficient generation.

A.2.2 Efficient Chain of Thoughts

A growing body of work studies how to improve the token efficiency of LLM reasoning (Liu et al., 2025b; Wu et al., 2025a; Wang et al., 2024a). At the prompt level, prior work reduces reasoning length by explicitly specifying brevity or budget constraints in the input (Xu et al., 2025b; Han et al., 2024; Muennighoff et al., 2025; Wong et al., 2025). At inference time, more adaptive methods regulate computation during generation (He et al., 2025; Yang et al., 2026; Yu et al., 2025c; Wang et al., 2025a; Damani et al., 2024; Snell et al., 2025; Fu et al., 2025a; Wang et al., 2025b; Fu et al., 2025b; Li et al., 2026a; Yu et al., 2025b; Yang et al., 2026).

Other works improve efficiency through learning. Some methods construct targeted fine-tuning data for shorter or more economical reasoning (Chen et al., 2025b; Han et al., 2024; Zeng et al., 2025; Xia et al., 2025; Kang et al., 2025; Munkhbat et al., 2025; Liu et al., 2024), while others use RL to encourage concise yet accurate reasoning, often through budget-aware objectives, concise reasoning rewards, curriculum learning, adaptive computation, or early stopping (Zhang et al., 2024; Qi et al., 2025; Li et al., 2026b; Zhang et al., 2025c; Yang et al., 2025a;

Aggarwal and Welleck, 2025; Hou et al., 2025; Arora and Zanette, 2025; Ma et al., 2026; Dumitru et al., 2025; Hammoud et al., 2025; Zhao et al., 2025; Wu et al., 2025b; Zhang et al., 2025b; Mao et al., 2025; Han et al., 2025; Shen et al., 2026). More recently, hybrid reasoning models such as Qwen3 (Yang et al., 2025a), Kimi k1.5 (Team et al., 2025c), and GPT-5 (OpenAI, 2025a) combine long- and short-thinking modes to better balance reasoning quality and efficiency. Concurrently, ParaThinker (Wen et al., 2025) trains LLMs to generate and aggregate multiple reasoning paths in parallel, whereas BudgetThinker targets fine-grained adherence of a single reasoning trace to an explicit token budget.

Despite this progress, most existing approaches provide only coarse control over reasoning effort, such as choosing a thinking mode, adjusting compute adaptively, or encouraging shorter outputs on average. In contrast, our method targets fine-grained reasoning-budget adherence: LLMs are trained to understand explicit budget constraints and to adjust their reasoning process according to inserted special tokens. The budget applies to model-generated reasoning tokens; the control-token overhead and fixed final-answer suffix are excluded.

A.2.3 CoT Controlling By Token Insertion

Token insertion during generation has been explored to control LLM behavior. In safety alignment, inserted intervention sequences steer models toward safer outputs without retraining (Jain et al., 2025; Wu et al., 2025c; Fei et al., 2025). Other works use inserted tokens to influence reasoning itself, such as injecting tokens to end reasoning (Sun et al., 2025) or for structured thinking (Wang et al., 2024b), or allocating more computation at uncertain steps (Kim et al., 2025). These methods mainly target reasoning quality, while our focus is reasoning length control.

Other related works use in-generation hints to control length, including token-level triggers for intermediate-budget reasoning (Yang et al., 2026), general recipes for controllable reasoning effort (He et al., 2025), and periodic multi-token textual hints (Pu et al., 2025; Tang et al., 2025). In contrast, our approach uses *single special tokens* rather than textual hints, making it more token-efficient and practical for small budgets. Moreover, unlike hint-based methods that rely on zero-shot instruction following, BudgetThinker is a

trainable framework that learns to treat these tokens as online budget signals.

A.3 LLM Dataset Details

For SFT, we select all 1k reasoning samples from s1k (Muennighoff et al., 2025), 0.8k reasoning samples from LIMO (Ye et al., 2025), and 17k reasoning samples from Bespoke-Stratos-17k (Labs, 2025), and a subset from NuminaMath (Project-Numina, 2024), Math (Hendrycks et al., 2021). Due to computational resource constraints, we remove all question-answer pairs with Chain-of-Thought (CoT) lengths over 10,000 tokens.

For reinforcement learning, the training data is a set of 14k math problems, comprising two parts: one part from the *math* category of the prm800k dataset (Lightman et al., 2023), and the other part from the *numina* and *amc* labeled data in the PRIME-RL/Eurus-2-RL-Data (Cui et al., 2026) dataset.

A.4 Compressor Prompt

Compressor Prompt Example

You are an agent tasked with compressing a reasoning process into a shorter, clearer version for a robot. Given the task description, the reasoning process, and the final answer, your goal is to shorten the reasoning process while preserving as many important details and as much logical completeness as possible.

Task: `<image><image>` The images are frames from a video. The video shows a ... Carefully analyze the multiple-choice question above and reason through it step by step.

Reasoning process: In the initial frame, the red fire box is centered, ... Therefore, the camera was moving left during the video. `<answer>A</answer>`

Note that:

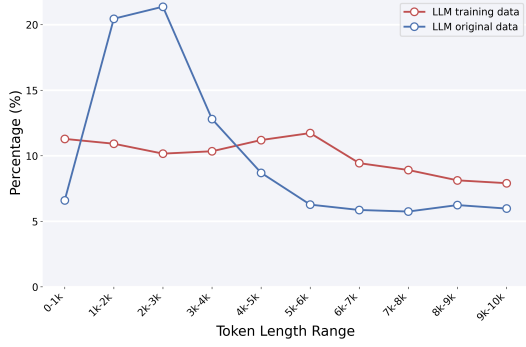
- You must keep the original tags `<think>`, `</think>`, `<answer>`, `</answer>` exactly as they are. Do not shorten, rename, or remove them.
- You cannot change the final answer. Only compress the reasoning process.
- The reasoning process should be as concise as possible while preserving logical flow and essential details.
- Return only the compressed reasoning process. Do not include any extra comments, explanations, or text.

Your answer:`<think>`

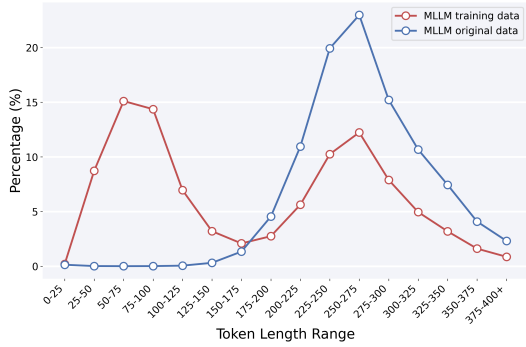
A.5 SFT Dataset Distribution

To ensure the model generalizes across various reasoning lengths, we implemented a data augmentation strategy using GPT-4o-mini to generate compressed chain-of-thought variations. For the MLLM dataset (originally 37k), we produced two compressed versions (a first-round and a second-round compression), expanding the dataset to 111k. For the LLM dataset (originally 21k), we generated one compressed version, bringing the total to 41k.

Figure 6b compares the length distributions of the original data against the augmented training



(a) Distribution between training and original data for LLM.



(b) Distribution between training and original data for MLLM.

Figure 6: Data distribution between the LLM and MLLM SFT datasets before and after augmentation.

set. As illustrated, the original data was heavily concentrated in specific length ranges. The inclusion of compressed data successfully expands the distribution to cover a much broader spectrum of answer lengths.

A.6 Code-domain Evaluation on HumanEval

We train a 1.5B DeepSeek-R1-distilled Qwen model for code-domain reasoning. SFT uses 31,619 examples from the AIZU subset of OpenCodeReasoning (Ahmad et al., 2025). We perform full fine-tuning for two epochs using AdamW, a learning rate of 2×10^{-5} , and a cosine scheduler. GRPO uses 427 MBPP training problems (Austin et al., 2021) and AdamW with a learning rate of 1×10^{-6} . We set the maximum model length to 12,288 tokens. For HumanEval (Chen et al., 2021), we sample eight solutions per problem with temperature 0.6 and top- p 0.95, apply a 3.0-second execution timeout, and report pass@1 and pass@8. The No Control Tokens and BudgetThinker variants use identical training and evaluation settings; the only changed variable is the ratio-based online signaling mechanism.

BudgetThinker improves both pass@1 and

pass@8 at five of six budgets, with average gains of 2.61 and 0.71 percentage points, respectively. The gains are especially clear for pass@1 at 1K–4K budgets. The small regression in pass@8 at 500 tokens illustrates that budget signaling improves pacing on average but cannot guarantee success when the requested reasoning budget is intrinsically insufficient.

A.7 Training and Evaluation Details

The granularity parameter for generating training budgets is set to $T = 50$. The importance hyperparameters are set to $k_1 = 0.7$, $k_2 = 0.15$, $k_3 = 0.15$. The length-reward penalty coefficient γ is set to 1 and 16 when $|y| \leq B$ and $|y| > B$, respectively. Thus, the length reward decays to 0 when the response exceeds B by one quarter. The choices of T , K , and the one-quarter cutoff are practical empirical settings rather than theoretically optimal constants. We have not evaluated insertion at semantically meaningful reasoning boundaries.

During evaluation, we generate N candidate solutions using the specified sampling temperature and top- p settings. We report pass@1 accuracy, with each question sampled N times and the resulting accuracy averaged. Following DeepSeek-R1 (DeepSeek-AI, 2025), we set N according to the test dataset: $N = 1$ for MATH-500 (temperature 0, top- p 1) and $N = 64$ for AMC 2023 (or $N = 8$ for the 7B model because of computational constraints).

A.8 Latency Overhead

The following Table 6 compares the latency overhead differences between our method and the c2f, Fixed Intervals, and Thought Terminator (Pu et al., 2025) approaches. It shows that our method uses significantly fewer additional tokens compared to Fixed Intervals and Thought Terminator, while being particularly close to c2f.

A.9 Latency Analysis

To validate the practical meaning of token budgets, we measure end-to-end latency on a single A800 GPU using vLLM with continuous batching enabled. As shown in Table 7, latency scales almost linearly with the generated-token budget. The average decoding cost is approximately 5.2 ms/token in this setting, which supports our use of token budget as a practical proxy for inference latency. Since BudgetThinker introduces only a small fixed number of inserted control tokens, this

Budget	Original Model			Laser			BudgetThinker		
	Acc. (%)	Avg. Tokens	Follow (%)	Acc. (%)	Avg. Tokens	Follow (%)	Acc. (%)	Avg. Tokens	Follow (%)
500	56.80	451.454	22.60	48.40	508.766	1.80	59.40	430.246	76.40
1000	70.00	789.172	43.20	70.20	931.906	32.80	75.00	897.880	69.40
2000	77.40	1349.304	60.80	80.20	1377.398	68.80	81.60	1743.494	78.60
4000	80.80	2037.216	70.00	82.60	1807.042	84.00	84.80	2798.166	94.00
6000	81.80	2592.142	74.80	83.80	2073.518	88.80	85.80	3405.768	94.80
10000	80.00	3188.208	80.40	83.20	2490.626	90.20	87.40	3691.484	97.60

Table 4: Additional budget-adherence metrics on MATH-500 under different reasoning budgets. We report accuracy, average model-generated reasoning tokens, and budget follow rate for the original model, Laser, and BudgetThinker. Control tokens and the fixed final-answer suffix are excluded from the token counts. BudgetThinker consistently achieves higher budget follow rates across all budgets while maintaining strong accuracy, indicating more reliable adherence to the requested reasoning budget.

Budget	pass@1 (%)		pass@8 (%)	
	No Control	BudgetThinker	No Control	BudgetThinker
500	3.20	4.80	9.15	7.93
1K	14.63	18.83	32.93	33.54
2K	34.76	41.08	63.41	64.02
4K	47.79	49.39	75.00	77.44
6K	52.82	52.06	81.71	82.93
8K	54.42	57.09	84.15	84.76

Table 5: HumanEval pass@1 and pass@8 (%) under different reasoning budgets. Both variants share the complete SFT+GRPO pipeline; only online ratio signaling differs.

overhead is small relative to the evaluated reasoning lengths. The actual end-to-end sequence also includes the fixed final-answer suffix, so the measurements should be interpreted as latency under the stated reasoning-budget protocol rather than a strict total-output cap.

A.10 Additional Budget-Adherence Metrics

In addition to accuracy, we report detailed budget-adherence metrics in Table 4, including the average number of model-generated reasoning tokens and the budget follow rate. The average excludes externally inserted control tokens and the fixed final-answer suffix. The budget follow rate measures the percentage of responses that terminate naturally within the specified reasoning budget. Across all evaluated budgets, BudgetThinker achieves substantially higher budget follow rates than the original model and Laser while maintaining strong accuracy. For example, at $B = 10,000$, BudgetThinker reaches a follow rate of 97.6%, compared with 80.4% for the original model and 90.2% for Laser. At this budget, BudgetThinker uses 3,691 reasoning tokens on average (36.9% of B), confirming that adherence does not imply exhausting the full ceiling.

A.11 Systematic Failure-Mode Analysis

We analyze incorrect MATH-500 responses at $B \in \{500, 2K, 4K\}$ for the Original model, No Control Tokens, and BudgetThinker. A response is marked as budget-related if it exceeds B , is truncated or incomplete at the boundary, or fails to produce a valid final answer. Percentages in Table 8 are normalized by the number of incorrect responses in each method–budget cell.

BudgetThinker substantially reduces budget-related failures at all three budgets. At $B = 500$, only 27.6% of its incorrect responses involve a budget violation or incomplete answer, compared with 96.7% for the Original model and 63.3% for No Control Tokens.

We additionally assign every incorrect response one mutually exclusive primary reasoning-error label by inspecting the failed trace. Termination denotes an incomplete derivation; Formulation denotes a misunderstanding of the problem or constraints; Strategy denotes an invalid solution approach; Coverage denotes omitted steps, cases, or constraints; Execution denotes arithmetic, algebraic, or logical errors; and Finalization denotes a correct derivation with an incorrectly extracted or formatted final answer. Table 9 reports the percentage of incorrect responses in each category.

These results show a shift in the remaining errors rather than their elimination. BudgetThinker converts many over-budget or incomplete responses into completed traces; its residual errors are predominantly complete-but-wrong execution, coverage, or strategy errors. Under very small budgets, the model can still omit necessary constraints or commit too early to a compressed route. In the qualitative example of Figure 8, the 500-token trace follows one partition path and omits

Model	Latency Overhead (tokens)					
	$B = 0.5K$	$B = 1K$	$B = 2K$	$B = 4K$	$B = 6K$	$B = 10K$
c2f	7	7	7	7	7	7
Fixed Intervals (250 Tokens)	9	11	15	23	31	47
Thought Terminator	51	99	195	387	579	963
BudgetThinker	14	14	14	14	14	14

Table 6: Latency overhead of different methods under multiple budget settings.

Budget	Original Latency (ms/sample)
250	1302.08
500	2624.45
1000	5202.57
2000	10410.78
4000	20835.37
6000	31295.96
10000	52523.27

Table 7: End-to-end latency measured on a single A800 GPU using vLLM with continuous batching enabled. Latency scales approximately linearly with the generation budget, at around 5.2 ms/token in this setting, supporting the use of token budget as a practical proxy for deployment latency.

Budget	Original	No Control	BudgetThinker
500	96.7	63.3	27.6
2K	84.8	63.0	46.4
4K	76.8	38.3	22.4

Table 8: Budget-related failures among incorrect MATH-500 responses (%).

the cross-checks retained at larger budgets, illustrating this coverage risk.

A.12 Hard-Problem Boundary: AIME 2024

On AIME 2024, BudgetThinker modestly outperforms ThinkPrune at every evaluated budget, but the margins remain small because some problems require a minimum reasoning length.

For AIME24 problem index 4 at $B = 2K$, BudgetThinker preserves the final verification and obtains the correct answer 110. Conversely, for problem index 1 with answer 113, no correct sample appears at $B = 2K$ or $4K$, whereas a correct sample appears at $B = 6K$ after the longer geometric derivation. These cases illustrate that control tokens can improve reasoning pacing but cannot remove the intrinsic minimum reasoning length of hard problems.

A.13 Detailed Analysis

For the qualitative analysis of the reasoning strategy variations of the BudgetThinker model under

Method / B	Term.	Form.	Strat.	Cov.	Exec.	Final.
Original / 500	5.0	3.3	2.1	2.5	82.0	5.0
No Control / 500	0.0	2.9	6.3	13.5	74.4	2.9
BudgetThinker/ 500	0.0	3.4	9.4	11.3	73.4	2.5
Original / 2K	6.2	6.2	5.4	5.4	72.3	4.5
No Control / 2K	0.0	3.6	13.4	5.4	73.2	4.5
BudgetThinker/ 2K	0.0	3.3	2.2	7.6	82.6	4.3
Original / 4K	4.0	6.1	7.1	7.1	69.7	6.1
No Control / 4K	0.0	6.2	39.5	12.3	38.3	3.7
BudgetThinker/ 4K	0.0	3.9	5.3	9.2	76.3	5.3

Table 9: Primary reasoning-error categories among incorrect MATH-500 responses (%). Categories are mutually exclusive within each response. Term., termination; Form., formulation; Strat., strategy; Cov., coverage; Exec., execution; Final., finalization.

Method	2K	4K	6K	10K
Original	14.80	23.28	25.71	28.88
ThinkPrune	15.22	22.78	25.22	27.80
BudgetThinker	16.25	23.80	25.63	28.90

Table 10: AIME 2024 accuracy (%) under different reasoning budgets.

different budget constraints, Figure 7 presents multiple responses generated by the BudgetThinker model for the same problem, Question-Math1, across various budget allocations.

It can be observed that as the reasoning budget is systematically reduced from 6000 to 500, the thought process exhibits a progressive compression in both methodological diversity and explanatory depth, along with the omission of certain steps. For a detailed analysis of the reasoning process, refer to Figure 8.

At the highest budget of 6000, the reasoning is most comprehensive and expanded, covering multiple methods (e.g., ordered pair enumeration, unordered pair deduplication, integer partitions, symmetry analysis, generating functions) and containing extensive metacognitive statements and pedagogical explanations. For instance, the generating function method is fully elaborated: *Alternatively, maybe using generating functions The generating function for indistinct objects coefficient of x^5 in*

$(1+x+\dots+x^5)^2$, alongside dense self-verification loops such as *Let me verify this.* and *Wait, but let me think again..*

When the budget is reduced to 4000, the reasoning shows multi-route, multi-round verification characteristics, but the depth and cyclicity of each method chain are compressed compared to 6000; method coverage remains broad (including ordered pairs, partitions, combinatorial formulas, test cases, etc.), but advanced methods like generating functions are only mentioned without elaboration, e.g., *But wait, let me verify this.* and *Suppose instead of 5 balls 1 ball the answer should be 1* demonstrate light reflection and the use of test cases.

Further compression occurs at a budget of 2000, reasoning retains multiple paths (e.g., full ordered pair enumeration, symmetric halving, partition verification) but each path is more linear, such as completely listing ordered pairs: $x + y = 5$ *ordered pairs: (0,5), (1,4), (2,3), (3,2), (4,1), (5,0). There are 6.*, followed by symmetric grouping: *Since all are of the form $a \neq b$, unordered pairs = $6 / 2 = 3$.*, yet lacks advanced methods (e.g., generating functions) and complex background explanations.

At a budget of 1000, reasoning is further compressed, retaining only core branching and slight hesitant language, e.g., *But wait, no, wait If the boxes are indistinct the problem reduces to partitions*, and introduces a second method (unordered pairs) for one-time verification: $(0,5),(1,4),(2,3)$ *Since the boxes are indistinct we have three distinct cases.*, but verification is limited and lacks redundant loops.

Finally, at the most constrained budget of 500, reasoning is extremely highly compressed, adopting a single linear path with highly structured language and no hesitation or alternative method discussion, for example, directly listing partitions: *Partitions of 5 into at most 2 parts: $5 = 5 + 0$, $4 + 1$, $3 + 2$* , and performing minimal verification: *Verification: $5+0$ corresponds to $4+1$ corresponds to $3+2$ corresponds to*, with the entire logical path being linear and without branching.

Lastly, a comparative analysis of the responses generated by the BudgetThinker model against other models on both the VQA and mathematical reasoning tasks is presented in Figures 11, 9, and 10, accompanied by a detailed examination of BudgetThinker’s advantages in handling diverse question types.

Return your final response within `\boxed{}`. How many ways are there to put 5 balls in 2 boxes if the balls are not distinguishable and neither are the boxes? `\n(Complete thinking within xxx tokens or fewer)<think>`

BudgetMind (budget=500):

BudgetMind (budget=1000):

BudgetMind (budget=2000):

Answer**\boxed{3}

BudgetMind (budget=6000)

ways**,\n<remaining>1/8</remaining>\n\n

31201

Budget	Number of reasoning steps	Primary steps	Verification
500	3 steps	1. Reformulate the problem as integer partitions. 2. Enumerate the three partitions. 3. Briefly verify the semantic meaning of each partition and count them.	Minimal – a single brief verification (e.g., “one box containing 5... 4+1... 3+2..”).
1000	7 steps	1. Explain the meaning of “indistinguishable.” 2. Temporarily list incorrect partitions (e.g., 3+1+1). 3. Exclude partitions with >2 parts. 4. Re-list the valid partitions. 5. Re-derive the result using unordered-pair reasoning. 6. Symmetry check (odd-case midpoint). 7. Conclude.	Moderate – at least two reflections / re-derivations (explicit “wait, no... let me check again” phenomenon).
2000	10 steps	1. Consider distinguishable boxes (solve $x+y=5$ → 6 nonnegative solutions). 2. Address indistinguishability and justify dividing by 2. 3. Enumerate ordered pairs. 4. Symmetry analysis ($x = y$ impossible here). 5. Conclude unordered count = 3. 6. Return to partition-based explanation. 7. List all 7 partitions of 5. 8. Filter to “at most 2 parts” → 3. 9. Revisit combinatorial formula and reflect. 10. Multiple cross-checks.	Extensive verification – ≈ three repeated derivations + multiple-method cross-validation.
4000	12 steps	1. Explicitly contrast distinguishable vs. indistinguishable boxes (“boxes are indistinct... adjust for that symmetry”). 2. Apply stars-and-bars / combinations for the distinguishable case: $\binom{5+2-1}{2-1} = \binom{6}{1} = 6$. 3. Motivate halving via symmetry / impose $(a \leq b)$ (list $(0,5;1,4;2,3)$). 4. Enumerate ordered pairs and group them. 5. Argue $(a=b)$ impossible since (5) is odd. 6. Give partition perspective (5, 4+1, 3+2). 7. Cross-validate enumeration against informal formula. 8. Use the bound $(a \in \{0, \dots, \lfloor \frac{5}{2} \rfloor\})$ to enumerate quickly. 9. Provide small-(n) sanity checks (e.g., $(n=1,2,3)$). 10. Discuss the interpretation of zero parts (empty boxes). 11. Reconcile multiple approaches to the same conclusion. 12. State final result (=3).	Multiple cross-validations (combinatorial formula, unordered-pair reasoning, partition listing, explicit enumeration); at least two distinct methods mutually confirm the outcome.
6000	13 steps	1. Precisely set up distinguishable vs. indistinguishable framing. 2. Provide the distinguishable-case stars-and-bars / ordered solutions: $((0,5),(1,4),\dots)$ (6 total). 3. Explain why indistinguishability collapses ordered solutions to unordered equivalence classes. 4. List unordered solutions with $(a \leq b)$: $((0,5),(1,4),(2,3))$. 5. Fully enumerate all partitions of 5 (all 7). 6. Explain the logical filter “partitions into (≤ 2) parts.” 7. Offer generating-function verification (coefficient of (x^5) in $((1+x+\dots)^2)$). 8. Clarify the relationship between allowing zero parts and the standard definition of integer partitions ($\text{parts}(\geq 1)$). 9. Reconcile stars-and-bars and partition perspectives (equivalences and differences). 10. Run multiple small-(n) sanity checks $((n=1,2,3,4))$. 11. Discuss the effect of object distinguishability on counts (comparative implications). 12. Synthesize all method outcomes into a final aggregate. 13. Reiterate the conclusion (“therefore, the answer is 3”) for emphasis.	Very extensive validation: diverse methods (enumeration, generating functions, algebraic equations, standard formulae), multiple test cases, and ≥3 independent confirmations.

Figure 8: This is a detailed analysis of the reasoning processes generated by the BudgetThinker model for Question-Math1 under different reasoning budgets.

Return your final response within `\boxed{}`. How many ways are there to put 5 balls in 2 boxes if the balls are not distinguishable and neither are the boxes? `\n(Complete thinking within 500/1000 tokens or fewer)<think>`

[illegible]

think=OKay, so I have this problem: I need to figure out how many ways there are to put 5 balls into 2 boxes. The balls are not distinguishable, and neither are the boxes. Hmm, let me think about this step by step. $\text{len}(\text{first})$, since the balls are not labeled, it doesn't matter which specific ball goes into which box. Similarly, the boxes themselves aren't labeled, so swapping the contents of the boxes doesn't create a new arrangement. That simplifies things a bit. Now, how many ways are there to put 5 balls into 2 boxes? Well, there are 5 possible distributions. Each ball has two choices: it can go into Box 1 or Box 2. Since there are 5 balls, each with 2 choices, the total number of ways without considering the boxes being indistinct would be 2^5 , which is 32. But wait, that's if the boxes are distinguishable. Since they aren't, some of these 32 arrangements are duplicates. $\text{len}(\text{first})$ the boxes were distinguishable, like if one box was red and the other was blue, each ball independently could go into either box, so there would be 32 unique arrangements. But since the boxes are indistinct, some of these are the same. For example, putting all 5 balls in Box 1 is the same as putting all 5 balls in Box 2 because the boxes are indistinct. $\text{len}(\text{no})$, how many unique distributions are there? Well, each distribution can be characterized by how many balls are in Box 1 and how many are in Box 2. Since the total is 5, the number of balls in Box 1 can range from 0 to 5. For each number of balls in Box 1, there's exactly one way to put the remaining balls in Box 2, because the boxes are indistinct. $\text{len}(\text{putting } k \text{ balls in Box 1 is the same as putting } 5 - k \text{ balls in Box 1})$, so for $k=0$, it's the same as $k=5$; for $k=1$, it's the same as $k=4$; and for $k=2$, it's the same as $k=3$. The middle case, $k=2.5$, $\text{len}(\text{cut})$

[illegible]

think/nokay, so I have this problem: I need to figure out how many ways there are to put 5 balls into 2 boxes. The balls are not distinguishable, and neither are the boxes. Hm, let me think about this step by step. \nfirst, since the balls are not labeled, it doesn't matter which box I put the balls in. I can just think of putting a ball in box A is the same as putting it in box B. That simplifies things a bit. Since I don't have to worry about which box is which: I just need to count the number of ways to distribute the balls.\n\nlet me consider the possible scenarios. Each ball has two choices: it can go into box 1 or box 2. If I think of each ball as having 2 choices, then the total number of ways is 2 multiplied by itself 5 times, which is 2^5. But wait, that would be 32. But hold on, that doesn't seem right because it counts all possible assignments, including the ones where all balls are in one box. But since the boxes are indistinguishable, having both boxes must have at least one ball, so maybe that's acceptable.\n\nbut let me think again. If the boxes are indistinct, does that mean that putting all balls in box 1 is the same as putting all in box 2? So, in that case, the number of distinct distributions would be half of 32, which is 16. But wait, that's only if the boxes are distinguishable. If the boxes are indistinct, each ball independently goes into either box, so the total number of assignments is 2^5, but since the boxes are indistinct, some of these assignments are duplicates. For example, putting a ball in box 1 and another in box 2 is the same as putting a ball in box 2 and another in box 1. So, the reality is that the number of distinct distributions is 16, which is 2^5 / 2. The 5 balls, because each subset represents the balls in box 1, and the complement represents the balls in box 2. Since the boxes are indistinct, the order doesn't matter, so the number of ways is 2^5 / 2 = 16. But wait, actually, the number of subsets is 2^5, which is 32, but since each subset and its complement are considered the same when boxes are indistinct, the number of distinct ways is (2^5 - 1) / 2. Hm, that seems conflicting.\n\nmaybe I'm overcomplicating it. Let me think about it differently. Since the balls are indistinct, I just need to count how many ways to have 0 to 5 balls in one box. Since the boxes are indistinct, the only thing that matters is how many go into each box. So, for each possible number of balls in box 1, there's exactly one way to have that many balls in box 2. The number is the same as the complement, which is when the number is 2 or 3, so 2 and 3 are complements of each other in a 5-ball set. \n\nwait, no, that's not quite right. If I have 5 balls, the number of ways to choose k balls for box 1 is C(5, k), where C is the combination function. But since the boxes are indistinct, choosing k balls for box 1 is the same as choosing 5 - k balls for box 2. So, the number of distinct distributions is equal to the number of distinct values of k, where k ranges from 0 to 5, but considering that k and 5 - k are the same when the boxes are indistinct. \n\nfor k = 0, there's 1 way to have 0 balls in box 1 and 5 in box 2. \nfor k = 1, there's 5 ways to have 1 ball in box 1 and 4 in box 2. \nfor k = 2, there's 10 ways to have 2 balls in box 1 and 3 in box 2. \nfor k = 3, there's 10 ways to have 3 balls in box 1 and 2 in box 2. \nfor k = 4, there's 5 ways to have 4 balls in box 1 and 1 in box 2. \nfor k = 5, there's 1 way to have 5 balls in box 1 and 0 in box 2. \n\nso, the distinct distributions are determined by the number of balls in box 1, which can be 0, 1, 2, or 3. That's 4 distinct distributions. But wait, that doesn't seem right because 0 and 5 are the same, 1 and 4 are the same, and 2 and 3 are the same. So, the number of distinct distributions is 4, which is (2^5 - 1) / 2. \n\nin the other, or 2 and 3. \n\n

Above are responses from two models—BudgetMind and the Original model—to the same question under different budget constraints (500 and 1000).

It can be observed that under both budget conditions, BudgetMind successfully completes its responses within the budget limits (322 and 840 tokens, respectively) while clearly following a compact, stepwise format and producing the final answer. In contrast, the Original model exhausts the allotted budget (500+ tokens and 1000+ tokens) by expanding irrelevant lines of thought and becomes truncated, failing to present a concise, correctly framed conclusion before budget exhaustion.

BudgetThinker etMind reduces the problem immediately to integer partitions of 5 into at most 2 parts and lists the valid partitions. This direct partition viewpoint eliminates unnecessary combinatorial detours, verifies each case succinctly, and counts them to produce `(\boxed{3})`. The chain of thought is minimal, tightly scoped to the mathematical equivalence (distribution $\approx$ partition), and avoids redundancy – yielding a correct, compact answer within budget.

Original model: The Original model spends tokens on irrelevant expansions (treating each ball as independently choosing a box $\rightarrow 2^45$), then trying to reconcile subset/complement reasoning), repeatedly rephrases the same ideas, and toggles between incompatible perspectives. This redundancy both wastes budget and creates internal contradictions; as a result the output is truncated and the final answer is not reliably delivered. The process lacks the direct, partition-based insight that would have yielded a short, correct result.

Figure 9: This is a comparison between BudgetThinker and Original model’s response cases on MATH datasets.

Return your final response within `\boxed{}`. What is the least positive integer multiple of 30 that can be written with only the digits 0 and 2? `\n`(Complete thinking within 500/2000 tokens or fewer)<think>

BudgetMind:

```

think>\n1. **Verify the properties of the number:**\n - The number must be a multiple of 30.\n - The number must be written using only the digits 0 and 2.\n\n2. **Check the divisibility rules:**\n - A number is divisible by 3 if the sum of its digits is divisible by 3.\n - For a number to be divisible by 2, its last digit must be 0.\n - For a number to be divisible by 5, its last digit must be 0 or 5.\n\n3. **Construct the number:**\n - Since the number must be a multiple of 30, it must be divisible by 2, 3, and 5. Therefore, the last digit must be 0.\n - We need to find the smallest number composed of the digits 0 and 2 that is divisible by 3.\n - The sum of the digits of the number must be divisible by 3.\n - The smallest number composed of the digits 0 and 2 that satisfies this condition is 2220, where the sum of the digits is  $2 + 2 + 2 + 0 = 6$ , which is divisible by 3.\n - Therefore, 2220 is divisible by 3.\n\n4. **Conclusion:**\n - The number 2220 is the smallest number composed of the digits 0 and 2 that is divisible by 30.\n\nFinal Answer: 2220

```

Total: 409 tokens

Original:

think>0,okay, so I need to find the smallest positive integer that's a multiple of 30 and only uses the digits 0 and 2. Hmm, let's break this down. First, multiples of 30 are numbers that can be divided by 30 without any remainder. So, the number must be divisible by both 3 and 10. For divisibility by 3, the sum of its digits must be divisible by 3. For divisibility by 10, the number must end with a 0. So, the number must end with a 0 and its sum of digits must be divisible by 3. Since we can only use 0 and 2, that's already satisfied because both are even. \n\nNext, divisibility by 5 means the number must end with a 0 or 5. But since we can only use 0 and 2, it must end with a 0. So the number has to end with 0. \n\nNow, for divisibility by 3, the sum of all digits must be a multiple of 3. Since we can only use 0s and 2s, the sum will be 2 times the number of 2s in the number. So, the number of 2s must be a multiple of 3. The smallest number of 2s is 3, giving us 222, but that's not divisible by 10. The next possible number is 2220, which is divisible by 10 and its sum of digits (2+2+2+0=6) is divisible by 3. \n\nSo, putting it all together, the number must end with a 0 and have a number of 2s that's a multiple of 3. Let's start with the smallest possible number. \n\nThe smallest number would be a 3-digit number because a 2-digit number can't have enough 2s to be divisible by 3. Let's see: 220. Is that divisible by 30? Let's check: 2220 divided by 30 is 74, which is an integer. So, 2220 is a multiple of 30. But wait, is there a smaller number? Let's check 2200. 2200 divided by 30 is 73.33, which is not an integer. So, 2200 is not a multiple of 30. \n\nThe next possible number of 2s is 3, so 2220 is the smallest. \n\nWait, hold on. 2220 is 222 followed by 0. But is there a smaller number? Let's think about 2220. Maybe a 4-digit number with fewer 2s? But 2220 is 4 digits, and if we try to make a CUT X

Total: 500+10 tokens

BudgetMind:

[illegible]

Total: 1723 tokens

Original:

[illegible]

Total: 2000+10 tokens

Comparison:

Above are responses from two models—BudgetMind and the Original model—to the same question under different budget constraints (500 and 2000).

It can be observed that under both budget conditions, BudgetMind successfully completes its responses within the budget limits (409 and 1723 tokens, respectively) while perfectly adhering to the specified format. In contrast, the Original model exceeds the maximum token limit (500+10 tokens and 2000+10 tokens) and fails to extract the answer in the correct format.

BudgetThinker:

the smallest number directly. The number must end in 0 and have a digit sum that is a multiple of 3. Using three 2's (the smallest number of 2's to meet this condition), it constructs 2220. This direct approach is **concise and efficient**. The reasoning process is **succinct and clear**, starting **directly** from the properties of multiples of 30 and analyzing the unit digit and digit sum conditions step by step to quickly reach the conclusion. It fully considers that a multiple of 30 must meet the conditions of being a multiple of 2, 3, and 5, and closely revolves around these conditions in each step, **without any redundant thinking**, directly arriving at the smallest number that fits the criteria, 2220.

Original model: The original model is **less efficient**. It repeatedly verifies the same conditions and explores unnecessary possibilities after identifying 2220. This redundancy wastes tokens and leads to an incomplete, erroneous output due to exceeding the token limit. The thinking process is **lengthy and complex**, with repeated checks and validations, and even unnecessary repetitions and superfluous thoughts (such as considering whether there could be a smaller number, despite having already identified the smallest one). Although it also considers the conditions for a multiple of 30, the thinking process **lacks efficiency and does not directly use the known conditions to reach a conclusion**. Instead, it tries to find the answer through continuous attempts and validations, resulting in a lengthy and error-prone process.

Figure 10: This is another comparison between BudgetThinker and Original model’s response cases on MATH datasets.



Figure 11: This is a comparison between BudgetThinker and Laser's response cases on VQA datasets.