BudgetThinker: Empowering Budget-Aware LLM Reasoning with Control Tokens

Hao Wen[1,*], **Xinrui Wu**[1,*,‡], **Yi Sun**[1], **Feifei Zhang**[1], **Liye Chen**[1,‡], **Jie Wang**[1,‡], **Yunxin Liu**[1], **Yunhao Liu**[2], **Ya-Qin Zhang**[1], **Yuanchun Li**[1,†]

[1]Institute for AI Industry Research (AIR), Tsinghua University
[2]Global Innovation Exchange & Department of Automation, Tsinghua University

[*]Equal contribution.
[†]Corresponding author: Yuanchun Li (liyuanchun@air.tsinghua.edu.cn).
[‡]Work done during internships at AIR, Tsinghua University.
Source Code: https://github.com/MobileLLM/BudgetThinker

Recent advancements in Large Language Models (LLMs) have leveraged increased test-time computation to enhance reasoning capabilities, a strategy that, while effective, incurs significant latency and resource costs, limiting their applicability in real-world time-constrained or cost-sensitive scenarios. This paper introduces BudgetThinker, a novel framework designed to empower LLMs with budget-aware reasoning, enabling precise control over the length of their thought processes. We propose a methodology that periodically inserts special control tokens during inference to continuously inform the model of its remaining token budget. This approach is coupled with a comprehensive two-stage training pipeline, beginning with Supervised Fine-Tuning (SFT) to familiarize the model with budget constraints, followed by a curriculum-based Reinforcement Learning (RL) phase that utilizes a length-aware reward function to optimize for both accuracy and budget adherence. We demonstrate that BudgetThinker significantly surpasses strong baselines in maintaining performance across a variety of reasoning budgets on challenging mathematical benchmarks. Our method provides a scalable and effective solution for developing efficient and controllable LLM reasoning, making advanced models more practical for deployment in resource-constrained and real-time environments.

1. Introduction

A key recent progress in Large Language Models (LLMs) is test-time scaling, a paradigm where models are prompted to "think longer" by generating extended Chain-of-Thought (CoT) reasoning before delivering a final answer (DeepSeek-AI, 2025; OpenAI, 2024). While this approach has unlocked state-of-the-art performance in complex domains like advanced mathematics (DeepSeek-AI, 2025; Google, 2025) and coding (Hui et al., 2024; Yang et al., 2025), it comes at the significant cost of increased latency and computational overhead. This reliance on increasingly long reasoning makes simple scaling strategies impractical for real-world systems with strict performance constraints, such as vehicle control, robotics, and AI agents.

This trade-off between performance and efficiency highlights a critical need for **controllable CoT reasoning**, where a model can tailor its reasoning process to conclude within a specified computational budget. Such a budget could be dictated by an application's hard limits (*e.g.*, under 100ms for autonomous driving (Lin et al., 2018; Chen et al., 2025a) or drone navigation (Chen et al., 2023)) or by user tolerance for cost and delay in long-running agentic systems. We frame this crucial instruction-following capability as **budget following**.

Existing approaches to control the reasoning budget, however, have proven insufficient. Directly

inserting budget constraints into prompts (Pu et al., 2025; Sun et al., 2025) often fails to reliably control output length (Han et al., 2024). Other methods that toggle between "thinking" and "non-thinking" modes (Yang et al., 2025; OpenAI, 2025) lack the fine-grained control necessary for variable budgets. Even more advanced training-based techniques using Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL) (Hou et al., 2025; Aggarwal & Welleck, 2025; Wu et al., 2025) struggle to enforce strict adherence to a specified constraint, a limitation our work confirms and aims to address.

In this paper, we address the challenge of precisely controlling the length of a model's thought process. 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 its response. To achieve this, we develop a novel training and inference framework that enables precise control over the length of the CoT. Our method periodically inserts special **Control Tokens** that act as explicit signals of the remaining budget, 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 teaching budget adherence.

(a) Answer example of our method

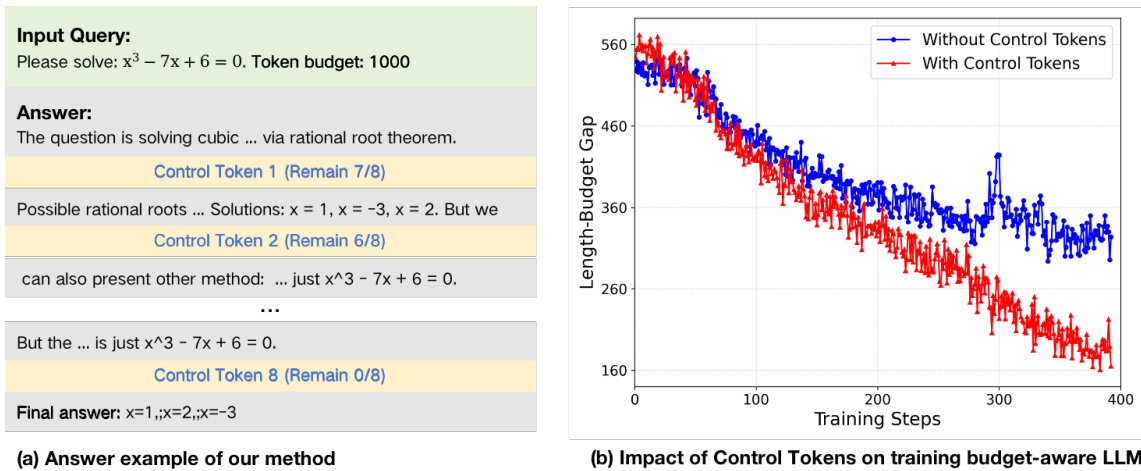**(b) Impact of Control Tokens on training budget-aware LLM**

Figure 1 | Illustration of the BudgetThinker framework and the impact of Control Tokens. (a) Example of budget-aware generation: Shows how BudgetThinker inserts control tokens into its reasoning process to stay aware of the remaining token budget. (b) Effect of Control Tokens on training: The gap between the generated answer length and the target budget during reinforcement learning.

Our main contributions are:

- We introduce a method that inserts special tokens during the inference process to achieve controllable output length. This technique minimizes the number of inserted tokens and their impact on the model's output quality.
- We design a comprehensive training process, progressing from SFT to RL, which incorporates a length-aware reward function. This allows the model to adapt to control via special tokens and produce higher-quality, budget-aware outputs. Our method is designed as a plug-and-play module that complements existing test-time scaling training procedures.
- We demonstrate that our approach surpasses strong baselines across multiple foundational models and datasets under various reasoning budgets.

We evaluate our method based on DeepSeek distilled Qwen 2.5 1.5B and 7B (DeepSeek-AI, 2025), on challenging reasoning benchmarks including MATH-500 (Hendrycks et al., 2021), AMC 2023, and AIME 2024. Compared to baselines such as the original reasoning models and a state-of-the-art efficient reasoning method (Hou et al., 2025), our approach demonstrates a better trade-off between accuracy and token budget. Specifically, our method improves accuracy by an average of 4.9% across all tested budgets and exhibits more precise adherence to the specified constraints.

2. Special-Token Conditioned Budget-Aware Reasoning

We present BudgetThinker, a budget-aware generation framework that guides LLMs to follow a predefined token budget by conditioning them on explicit control signals. Our method enhances budget adherence by (i) introducing a dynamic control token insertion strategy that continuously reminds the model of the remaining budget and (ii) developing a two-stage training pipeline combining supervised fine-tuning (SFT) and reinforcement learning (RL) to teach the model to follow these control tokens.

2.1. Special-Token Budget Signaling

Our method enforces a target output length, or budget, by augmenting the standard autoregressive generation process. The core idea is to *insert a small, fixed set of special control tokens into the sequence at positions corresponding to fractions of the total budget*. These tokens act as explicit signals to the model, indicating how much of its 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, $C = \{c_1, c_2, \ldots, c_K\}$, 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 $K=10$). During generation, the token y_t at timestep t is determined by the following piecewise function:

$$y_t = \begin{cases} c_{k+1} & \text{if } t = k \cdot \lfloor B/K \rfloor \text{ for an integer } k \in \{0, 1, \ldots, K-1\} \\ y_t' \mid (y_t' \sim \pi_\theta(\cdot|x, y_{<t})) & \text{otherwise} \end{cases}$$

At each timestep t that corresponds to the end of a budget fraction $\lfloor k \cdot B/K \rfloor$, 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.

For comparison, we define a naive baseline: *Fixed-Interval Signaling*. This simpler strategy inserts control tokens at constant, fixed intervals rather than at budget ratios. Given a fixed interval length I, a control token is inserted every I steps (*i.e.*, when $t = k \cdot I$). However, this approach lacks scalability with respect to the budget B. For example, if $I = 100$, a budget of $B = 500$ would require 5 control tokens, while a budget of $B = 1000$ would require 10. The model must be trained to recognize a potentially unbounded number of control tokens as the budget grows. In contrast, our ratio-based method always uses the same fixed set of K tokens, making it far more robust to 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* function adds the control token c_{k+1} at each position t when $t = k \cdot I$. Then, we get a reconstructed dataset $\{(\hat{x}, \hat{y})\}$, which we use to fine-tune LLM.

Dataset Balancing. We balance the dataset to ensure the model's budget following skill is generalizable across different task types and output lengths. We achieve this by creating a mixture of two data categories: **(1) Complex Reasoning Data**: Samples sourced from datasets like s1k (Muennighoff et al., 2025), LIMO (Ye et al., 2025), and Bespoke-Stratos-17k (Labs, 2025), which typically require long, step-by-step thought processes and generate lengthy responses. **(2) Short CoT Data**: Samples from datasets like NuminaMath (LI et al., 2024) and MATH training set (Hendrycks et al., 2021), where tasks often have short chain-of-thought extended reasoning (most answers less than 1000 tokens). By blending these sources, we ensure our SFT dataset has a well-balanced distribution of target answer lengths.

2.3. Reinforcement Learning

Following supervised fine-tuning (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.

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:

$$R(y, y_{gold}, B) == k_1 \cdot \mathbf{1}\{y = y_{\text{gold}}\} + k_2 \cdot \mathbf{1}\{\text{format}(y)\} + k_3 \cdot \underbrace{\max(1 - \gamma \cdot \left(\frac{B - |y|}{B}\right)^2, 0)}_{\text{Length Reward}}$$

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 substantially larger penalty for exceeding the budget than for falling short of it. This design encourages the model to generate responses that are not only correct but also precisely tailored to the length constraint.

Iterative Reinforcement Learning. To ensure the model performs robustly across a wide range of budgets, we implement a iterative learning strategy within the RL pipeline. Our approach is based on the hypothesis that generating concise, complete answers under tight budget constraints is more difficult than generating longer, more verbose ones. The iterative, therefore, progresses from easier to harder tasks. The training is structured in sequential stages. In each stage k, the model is trained to generate responses within a specific budget B_k. The curriculum proceeds by systematically decreasing the budget across stages, following the sequence $B_1 > B_2 > \cdots > B_n$.

Finally, to prevent catastrophic forgetting and maintain proficiency across all learned budget levels, we conclude with a **mixed-budget training phase**. In this final stage, a budget B_k is randomly sampled from the set of all previously used budgets $\{B_j\}_{j=1}^n$ for each training batch. This compels the model to retain its ability to generate high-quality responses for any length within the learned spectrum.

3. Experiments

In this section, we present a comprehensive empirical evaluation of BudgetThinker. We investigate its performance under various computational budgets and analyze its ability to adhere to these budgets (Section 3.2). We then analyze the impact of different control token insertion strategies (Section 3.3), iterative training (Section 3.4), and reinforcement learning of BudgetThinker (Section 3.5).

3.1. Experiment Setup

Training Details. Our experiments 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 Supervised Fine-Tuning (SFT), we create a dataset of 41k problem-solution pairs as described in Section 2.2. Initially, 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(LI et al., 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. We train all SFT models for 6 epochs 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). The training data is a set of 14k math problems, comprising two parts: one part from the math_splits category of the prm800k dataset (Lightman et al., 2023), and the other part from the numina_amc_aime labeled data in the PRIME-RL/Eurus-2-RL-Data(Cui et al., 2025) dataset. 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 $K = 8$. This corresponds to

inserting 8 special tokens during decoding, indicating remaining budget fractions of $7/8, 6/8, \ldots, 1/8$. 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 as:

$$\gamma = \begin{cases} 1 & \text{if } |y| \leq B \\ 16 & \text{if } |y| > B \end{cases}$$

Namely, the length reward will decade to 0 when the length of answer y exceeds the budget B by $1/4$. For the iterative training curriculum, we start from a budget of 6000, then reducing it to 4000, 3000, and finally 2000. Following this curriculum, we conduct a final mixed-budget training phase by randomly sampling $B \in \{6000, 4000, 3000, 2000\}$.

Evaluation Details. We evaluate our methods and baselines on three challenging math benchmarks: MATH-500 (Hendrycks et al., 2021), AIME 2024, and AMC 2023. To assess budget control, we evaluate performance across a range of reasoning budgets, from $B = 500$ 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 the final answer generation before terminating the process entirely. For each test question, we generate N candidate solutions using sampling with a temperature of 0 and a top-p value of 1 for MATH-500, and with a temperature of 0.6 and a top-p value of 0.95 for AIME 2024 and AMC 2023. We report the pass@1 accuracy. We set the number of samples N to 1 for MATH-500, and to 64 for AIME 2024 and AMC 2023.

Baselines. We benchmark BudgetThinker against two baselines: **(1) ThinkPrune** (Hou et al., 2025): A state-of-the-art efficient reasoning framework that also uses GRPO for budget control but does not incorporate explicit control tokens. As the official ThinkPrune repository does not provide a 7B model, we limit our comparison to the 1.5B model scale. **(2) 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. Main Performance

We first evaluate the accuracy of BudgetThinker against the baselines across various token budgets, with results shown in Figure 2 and 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 4.2% over the original model and 5.7% over ThinkPrune on average. However, on the AIME 2024 benchmark, the performance of all three methods was comparable. This is likely because the inherent difficulty of AIME problems demands longer, more complex chains of thought; simply restricting the generation length does not necessarily yield better solutions, as producing a concise yet correct proof requires a higher level of reasoning.

We also analyze the budget-following capabilities of each model, as illustrated in Figure 3a. Besides, in Figure 3b, we calculate the average relative length at each budget ($|y|/B$) for every answer that is within budget, which shows the capability for LLM to understand and fully use budgets. The original model, lacking any budget-specific training, exhibits poor capability to follow the specified limits. While ThinkPrune (Hou et al., 2025) shows improved budget awareness, it often underutilizes the allocated budget, prematurely concluding its reasoning process, which can negatively impact performance on complex problems. In contrast, BudgetThinker demonstrates superior budget adherence. Guided by the

(a) 1.5B LLMs on MATH-500

(b) 7B LLMs on MATH-500

(c) 1.5B LLMs on AMC 2023
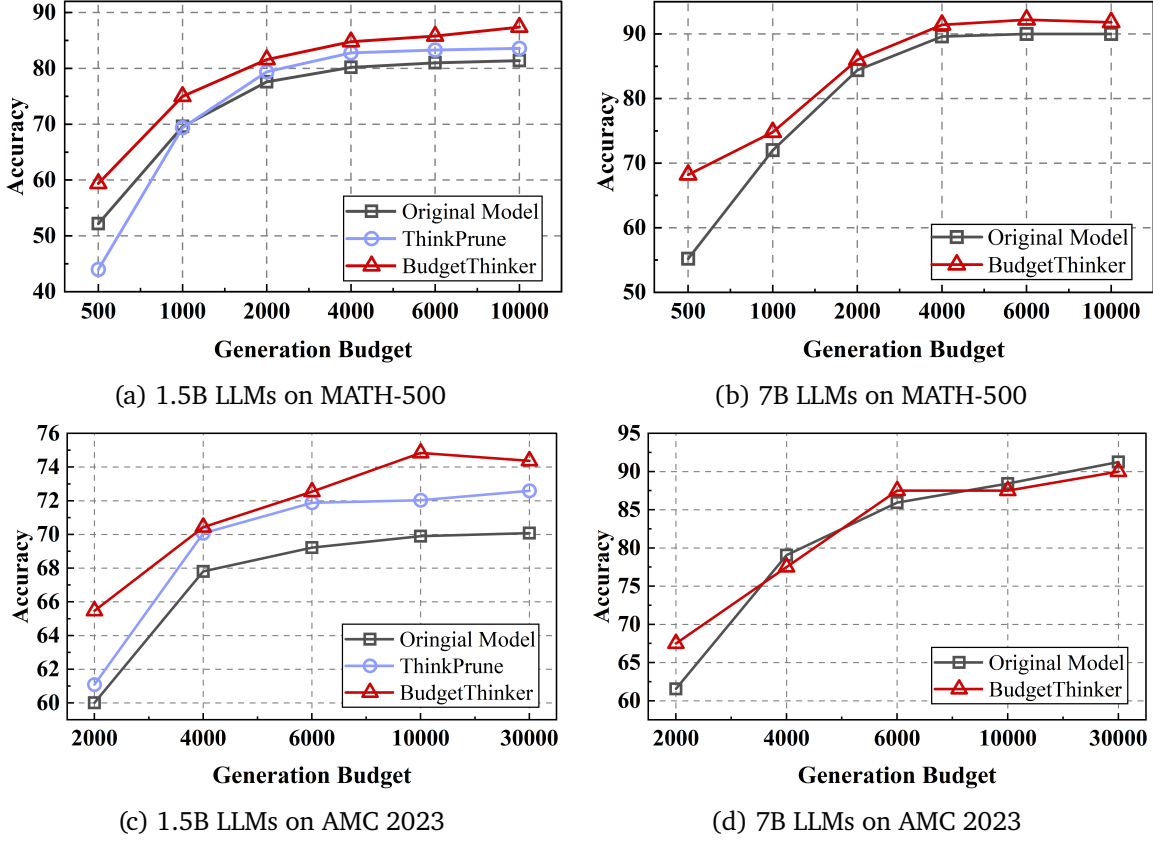
(d) 7B LLMs on AMC 2023

Figure 2 | Pass@1 accuracy of BudgetThinker vs. baselines across various generation budgets on MATH-500 and AMC 2023. Accuracy is plotted against the maximum generation budget, a metric more applicable to real-time scenarios than average reasoning length.

(a) Budget following ratio.
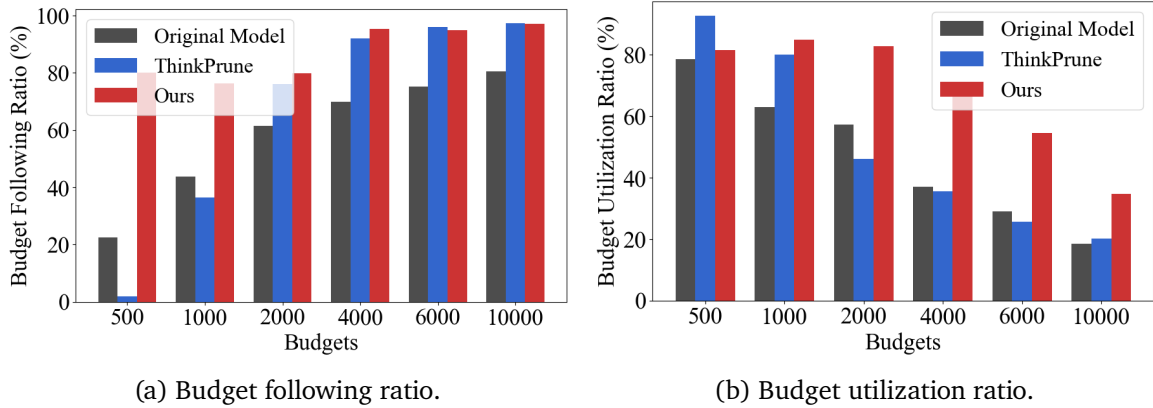
(b) Budget utilization ratio.

Figure 3 | Budget following and utilization analysis on MATH-500. We compare BudgetThinker-1.5B, ThinkPrune-1.5B, and the original DeepSeek-Distilled-Qwen-2.5-1.5B on MATH-500. (a): The percentage of generated responses that terminate naturally within the specified budget limit (B). (b): For responses that finish within the budget, this shows the average generated length as a percentage of the total allocated budget ($|y|/B$). This shows BudgetThinker has a superior ability to follow budget constraints while also making more effective use of the allocated tokens compared to baselines.

	B=2000	B=4000	B=6000	B=10000
Original-1.5B	14.80	23.28	25.71	28.88
ThinkPrune-1.5B	15.22	22.78	25.22	27.80
BudgetThinker-1.5B	16.25	23.80	25.63	28.90

Table 1 | Performance of BudgetThinker and baselines on AIME 2024 with different budgets (B).

remaining budget tokens, it effectively utilizes the entire allocated length without significant overruns. This precise control allows it to achieve a better balance between computational cost and accuracy, leading to its state-of-the-art performance.

3.3. Analysis of Control Token Insertion Strategies

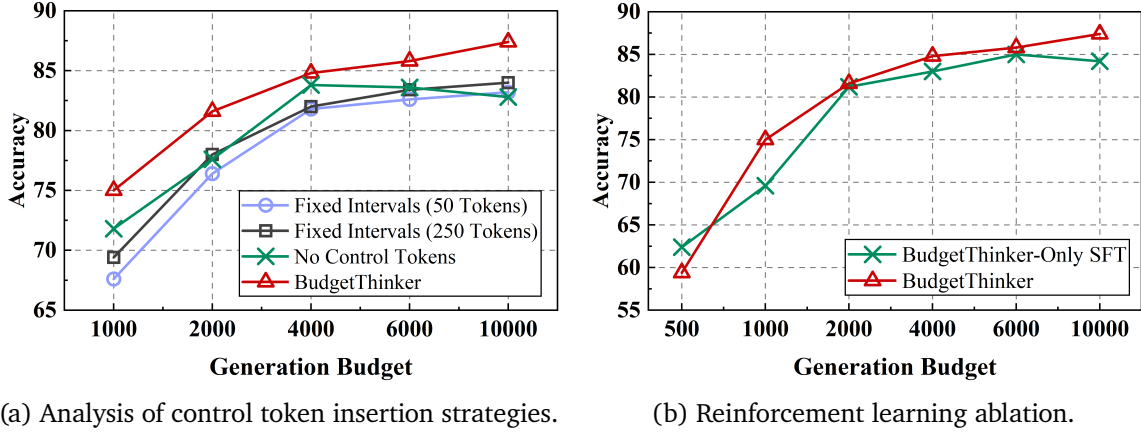(a) Analysis of control token insertion strategies.   (b) Reinforcement learning ablation.

Figure 4 | Ablation studies on control strategy and RL training (MATH-500, 1.5B Model).

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, presented in Figure 4a, indicate that the **Budget Ratio** strategy yields the best performance. This suggests that a relative, ratio-based notification of remaining resources is more intuitive for the

LLM to learn and adapt its reasoning pace, compared to fixed-interval notifications. Furthermore, the 250-token interval outperforms the 50-token interval, implying that a sparser, less intrusive set of control signals is more beneficial. The superior performance of our default strategy over the "No Control Tokens" baseline further validates the effectiveness of our core design.

3.4. Analysis of Iterative Training

Model	Accuracy (%)				Budget Following Ratio (%)			
	$B = 2K$	$B = 4K$	$B = 6K$	$B = 10K$	$B = 2K$	$B = 4K$	$B = 6K$	$B = 10K$
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 | This table tracks the 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.

To investigate the impact of our iterative training curriculum, we evaluated 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, the LLM achieves a balanced capability to handle all budgets, even if it is not individually optimal for every single budget constraint.

3.5. 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 4b demonstrate that the full RL-tuned model consistently surpasses the SFT-only model across all 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 discover more effective reasoning strategies that optimize the reward under specific budget constraints, ultimately leading to higher accuracy.

4. Related Work

4.1. Test-Time Scaling for Reasoning LLMs

Recent advances in test-time scaling seek to improve LLM reasoning by increasing computational depth during decoding. Among these test-time-scaling methods, reinforcement learning encourages LLMs to explore different strategies of solving problems and allocate more reasoning times for reflection. It has

shown great potential in boosting LLM reasoning in multiple domains, including math (DeepSeek-AI, 2025; Google, 2025), coding (Hui et al., 2024; Yang et al., 2025), agentic tasks (Team, 2025a), and multimodal reasoning (Tan et al., 2025; Shen et al., 2025). Some other works propose to distill reasoning ability from long reasoning CoTs generated by large models to smaller LLMs to encourage deep thinking Muennighoff et al. (2025); Labs (2025); Team (2025b); Ye et al. (2025); Xu et al. (2025a); Geiping et al. (2025). While effective at enhancing complex problem-solving, these methods often suffer from significant inference latency due to the generation of lengthy outputs (Sun et al., 2025; Zhu & Li, 2025; Qu et al., 2025), rendering them impractical for deployment in real-time systems (Wen et al., 2023; Jiang et al., 2021; Han et al., 2021). Besides, recent works also found that lengthy outputs often result in overthinking (*e.g.* meaningless repeats) and even lead to errors (Yu et al., 2025a; Chen et al., 2025b). Our method teaches LLMs how to follow a user-specified token budget during its reasoning process. This helps the model generate more controlled and efficient reasoning.

4.2. Efficient Chain of Thoughts

Several methods have been proposed to alleviate the token efficiency problem of LLMs' reasoning. Prompt methods (Xu et al., 2025b; Han et al., 2024; Muennighoff et al., 2025) make LLMs generate less reasoning tokens directly by adding explicit length constraints into the prompts. Solution routing works (Yu et al., 2025b; Wang et al., 2025a) allow for mid-generation control to prune unpromising traces. Computation routing methods (Damani et al., 2024; Fu et al., 2025; Wang et al., 2025b; Li et al., 2025) choose to allocate just the necessary computation budget based on predicted complexity of queries, either by routing to larger models or conducting more samplings to vote for the final answer. There are also learning methods that either construct well-designed datasets to apply model fine-tuning (Chen et al., 2025b; Han et al., 2024; Xia et al., 2025; Zeng et al., 2025) or use RL methods (Yu et al., 2025a; Yang et al., 2025; Aggarwal & Welleck, 2025; Hou et al., 2025; Arora & Zanette, 2025) to encourage model to generate concise yet accurate answers. More recently, models like Qwen3 (Yang et al., 2025), Kimi k1.5 (Team et al., 2025) and GPT-5 (OpenAI, 2025) integrate hybrid thinking modes of long and short CoTs, seeking trade-off between output length and model performance. Instead, our work achieves precise token budget control. LLMs are trained to understand and follow the constraint in prompts and adaptively adjust their reasoning process to complete the task within the user-requested budgets, showcasing the flexibility and efficiency of our method.

5. Conclusion

In this work, we introduce BudgetThinker, a novel framework that manages the trade-off between reasoning quality and computational cost in LLMs. By using special control tokens and a two-stage training pipeline, BudgetThinker enables precise, budget-aware control over the model's reasoning length. Our experiments validate that this method achieves superior budget adherence while maintaining high accuracy on challenging benchmarks. Thus, BudgetThinker represents a significant step towards developing more efficient LLMs suitable for real-time, resource-constrained applications.

References

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.04697.

Daman Arora and Andrea Zanette. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*, 2025.

Guojun Chen, Xiaojing Yu, Neiwen Ling, and Lin Zhong. Typefly: Flying drones with large language model. *arXiv preprint arXiv:2312.14950*, 2023.

Jiale Chen, Duc Van Le, Yuanchun Li, Yunxin Liu, and Rui Tan. Timelynet: Adaptive neural architecture for autonomous driving with dynamic deadline. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2025a.

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. Do not think that much for 2+3=? on the overthinking of o1-like llms, 2025b. URL https://arxiv.org/abs/2412.21187.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards, 2025. URL https://arxiv.org/abs/2502.01456.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of lm computation, 2024. URL https://arxiv.org/abs/2410.04707.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. Efficiently scaling llm reasoning with certaindex, 2025. URL https://arxiv.org/abs/2412.20993.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *CoRR*, abs/2502.05171, February 2025. URL https://doi.org/10.48550/arXiv.2502.05171.

Google. Gemini 2.5: Our most intelligent ai model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking, March 2025.

Rui Han, Qinglong Zhang, Chi Harold Liu, Guoren Wang, Jian Tang, and Lydia Y. Chen. Legodnn: block-grained scaling of deep neural networks for mobile vision. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, pp. 406–419, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383424. doi: 10.1145/3447993.3483249. URL https://doi.org/10.1145/3447993.3483249.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2504.01296.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. Flexible high-resolution object detection on edge devices with tunable latency. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, pp. 559–572, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383424. doi: 10.1145/3447993.3483274. URL https://doi.org/10.1145/3447993.3483274.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Bespoke Labs. Bespoke-stratos: The unreasonable effectiveness of reasoning distillation. https://www.bespokelabs.ai/blog/bespoke-stratos-the-unreasonable-effectiveness-of-reasoning-distillation, 2025. Accessed: 2025-01-22.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, and Zhifang Sui. Selfbudgeter: Adaptive token allocation for efficient llm reasoning. *arXiv preprint arXiv:2505.11274*, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 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*, pp. 751–766, 2018.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

OpenAI. Openai o1 system card. 2024. URL https://arxiv.org/abs/2412.16720.

OpenAI. Gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf, August 2025. Accessed: 2025-08-21.

Xiao Pu, Michael Saxon, Wenyue Hua, and William Yang Wang. Thoughtterminator: Benchmarking, calibrating, and mitigating overthinking in reasoning models. *arXiv preprint arXiv:2504.13367*, 2025.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning, 2025. URL https://arxiv.org/abs/2503.07572.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, et al. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*, 2025.

Yi Sun, Han Wang, Jiaqiang Li, Jiacheng Liu, Xiangyu Li, Hao Wen, Yizhen Yuan, Huiwen Zheng, Yan Liang, Yuanchun Li, and Yunxin Liu. An empirical study of llm reasoning ability under strict output length constraint. *arXiv preprint arXiv:2504.14350*, April 2025. doi: 10.48550/arXiv.2504.14350.

Huajie Tan, Yuheng Ji, Xiaoshuai Hao, Minglan Lin, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. Reason-rft: Reinforcement fine-tuning for visual reasoning. *arXiv preprint arXiv:2503.20752*, 2025.

Kimi Team. Kimi k2: Open agentic intelligence, 2025a. URL https://arxiv.org/abs/2507.20534.

Kimi Team et al. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

NovaSky Team. Sky-t1: Train your own o1 preview model within $450. https://novasky-ai.github.io/posts/sky-t1, 2025b. Accessed: 2025-01-09.

Siyuan Wang, Enda Zhao, Zhongyu Wei, and Xiang Ren. Stepwise informativeness search for efficient and effective llm reasoning, 2025a. URL https://arxiv.org/abs/2502.15335.

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning, 2025b. URL https://arxiv.org/abs/2408.13457.

Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Yunxin Liu. Adaptivenet: Post-deployment neural architecture adaptation for diverse edge environments. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399906. doi: 10.1145/3570361.3592529. URL https://doi.org/10.1145/3570361.3592529.

Xingyu Wu, Yuchen Yan, Shangke Lyu, Linjuan Wu, Yiwen Qiu, Yongliang Shen, Weiming Lu, Jian Shao, Jun Xiao, and Yueting Zhuang. Lapo: Internalizing reasoning efficiency via length-adaptive policy optimization. *arXiv preprint arXiv:2507.15758*, 2025.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms, 2025. URL https://arxiv.org/abs/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. Redstar: Does scaling long-cot data unlock better slow-reasoning systems?, 2025a. URL https://arxiv.org/abs/2501.11284.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*, 2025b.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale, 2025. *URL https://arxiv. org/abs/2503.14476*, 2025a.

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. Think smarter not harder: Adaptive reasoning with inference aware optimization, 2025b. URL https://arxiv.org/abs/2501.17974.

Wenhao Zeng, Yaoning Wang, Chao Hu, Yuling Shi, Chengcheng Wan, Hongyu Zhang, and Xiaodong Gu. Pruning the unsurprising: Efficient code reasoning via first-token surprisal, 2025. URL https://arxiv.org/abs/2508.05988.

Jason Zhu and Hongyu Li. Towards concise and adaptive thinking in large reasoning models: A survey, 2025. URL https://arxiv.org/abs/2507.09662.