SELFBUDGETER: ADAPTIVE TOKEN ALLOCATION FOR EFFICIENT LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently, large reasoning models demonstrate exceptional performance on various tasks. However, reasoning models inefficiently over-process both trivial and complex queries, leading to resource waste and prolonged user latency. To address this challenge, we propose SelfBudgeter - a self-adaptive controllable reasoning strategy for efficient reasoning. Our approach adopts a dual-phase training paradigm: first, the model learns to pre-estimate the reasoning cost based on the difficulty of the query. Then, we introduce budget-guided GPRO for reinforcement learning, which effectively maintains accuracy while reducing output length. SelfBudgeter allows users to anticipate generation time and make informed decisions about continuing or interrupting the process. Furthermore, our method enables direct manipulation of reasoning length via pre-filling token budget. Experimental results demonstrate that SelfBudgeter can dynamically allocate budgets according to problem complexity, yielding an average response length compression of 61% for the 1.5B model on GSM8K, MATH500, and AIME2025, and 48% for the 7B model, while maintaining nearly undiminished accuracy.

1 INTRODUCTION

Recent large reasoning models, such as O1 (OpenAI, 2024), has shown remarkable performance in various complex reasoning tasks (DeepSeek-AI et al., 2025; Qwen, 2024). The primary success factor lies in the long chain of thought (CoT) process learned through reinforcement learning (RL), which allows the model to break down reasoning steps and scaling test-time compute (Snell et al., 2024; Luo et al., 2025b).

However, reasoning models tend to use overly long thought processes even for simple questions. This "overthinking" phenomenon leads to a waste of computational resources and excessive user waiting times (Chen et al., 2024; Sui et al., 2025). For example, when answering the simple questions such as "What is the answer of 2+3?", the QwQ-32B model provides 13 different solutions and generates 100 times more tokens than Qwen2.5-72B-Instruct model (Qwen et al., 2025).

Prior studies have explored various approaches to mitigate overthinking through response length control and computation routing. Existing methods mainly include: (1) Prompt-based approaches (Lee et al., 2025; Xu et al., 2025a) that implicitly guide length through instructions, (2) Integrated training strategies that teach models to adaptively determine reasoning steps via SFT (Munkhbat et al., 2025; Ma et al., 2025) or RL with length penalties (Aggarwal & Welleck, 2025; Arora & Zanette, 2025), and (3) Router-based (Aytes et al., 2025; Chuang et al., 2025) architectures employing classifiers to allocate computation paths. While achieving partial progress, these methods either lack precise length control, require additional computational overhead, or fail to explicitly output optimal reasoning lengths (Aggarwal & Welleck, 2025; Xu et al., 2025b).

We propose **SelfBudgeter** that enables reasoning models to (1) estimate the minimal token budget required for correct responses when users do not specify token constraints, and (2) generate responses of corresponding lengths while adhering to either self-estimated or user-defined token budgets. SelfBudgeter aims to mitigate the overthinking issue by predicting the minimal possible token budget, thereby significantly reducing user waiting time. As shown in Figure 1, SelfBudgeter can provide a relatively accurate token budget estimation before generating responses, users can precisely anticipate the waiting time and decide whether to wait for the full output or terminate early based on their needs.

Additionally, when specific requirements arise, users can pre-fill the token budget field to constrain the model's response within the given limit, thereby improving interaction efficiency.

Our training framework consists of two main stages. During the Cold-Start stage, we fine-tune the model to learn how to first output its estimated token budget within <budget> tags. Subsequently, in the RL training stage, we optimize SelfBudgeter using the GRPO algorithm. For this stage, we design a reward function that primarily focuses on three key aspects: (1) answer correctness, (2) minimal achievable token budget, and (3) consistency between response length and the allocated token budget.

We conduct full-parameter training of Deepseek-R1-Distill-Qwen-1.5B using SelfBudgeter and evaluate its performance on the GSM8K, MATH500 and AIME2025 datasets. Experimental results demonstrate that SelfBudgeter achieves an average response length compression of 61% with the 1.5B model, while maintaining nearly equivalent accuracy. Furthermore, on GSM8K and MATH500, SelfBudgeter simultaneously reduces response length while improving accuracy. SelfBudgeter also exhibits excellent capability in predicting output length and, when provided with pre-filled <budget> tags, consistently adheres to the specified token budget constraints. In addition, experiments on Deepseek-R1-Distill-Qwen-7B show an average compression of 48%, further validating the scalability of SelfBudgeter to larger model sizes.

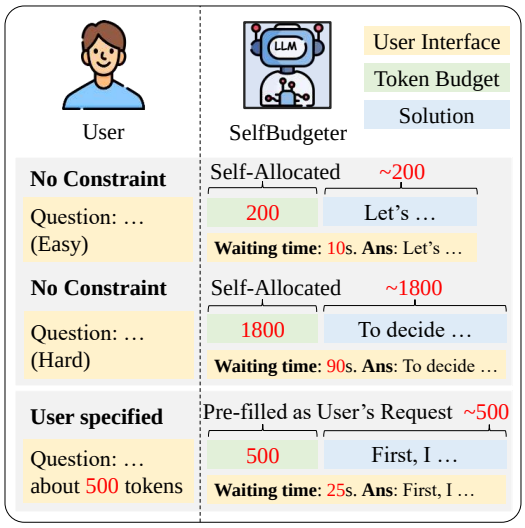

Figure 1: Overview of the SelfBudgeter. SelfBudgeter's responses comprise two sections: **Token Budget** and **Solution**. For unrestricted queries, it estimates tokens needed for the **Solution** based on problem complexity. When users specify requirements, it pre-fills the **Token Budget** accordingly. The **Solution** generation strictly adheres to **Token Budget** limits, whose numerical value indicates anticipated response duration.

2 Related Work

Overthinking in LLMs The emergence of the reasoning models like O1, Deepseek-R1 and QwQ advanced complex problem-solving through RL-enhanced CoT (Wei et al., 2022; OpenAI, 2024; DeepSeek-AI et al., 2025; Qwen, 2024). However, researchers observed a tendency for reasoning models to overthink simple problems—expending unnecessary computational effort on trivial queries (Chen et al., 2024; Sui et al., 2025). Excessive long CoT may lead to a decrease in accuracy (Wu et al., 2025). Current solutions for overthinking mainly involve following three strategies. Prompt-based methods try to control response length by adding instructions in prompts, but cannot control the length accurately (Lee et al., 2025; Renze & Guven, 2024; Xu et al., 2025a; Nayab et al., 2024). Integated Training-based methods try to teach model decide the length by the difficulty of the problems. Supervised fine-tuning(SFT)-based methods collect the dataset with variable length (Munkhbat et al., 2025; Ma et al., 2025; Liu et al., 2024; Han et al., 2024; Kang et al., 2024; Xia et al., 2025; Yang et al., 2025b). RL-based methods incorporate length penalties into the reward function (Aggarwal & Welleck, 2025; Arora & Zanette, 2025; Luo et al., 2025a; Chen et al., 2025a; Chang et al., 2025; Xu et al., 2025b; Yang et al., 2025a). These methods fail to control the length as users' requirements. And Router-based methods train another model as a classifier (Aytes et al., 2025; Chuang et al., 2025; 2024; Ong et al., 2024; Pan et al., 2025). The classifier decide to route the query to fast models or reasoning models. However, an extra classifier means more computation resources are needed. Current methods either sacrifice precise control, require extra computation, or fail to bridge autonomous budget estimation with strict adherence.

Token Budget In addressing the issue of overthinking, a highly intuitive approach involves directly constraining the output length. CCoT (Nayab et al., 2024) attempt to achieve this by incorporating a word budget into the prompt, various approaches—including character, token, and step budgets (Lee

et al., 2025)—have been attempted by directly incorporating them into prompts, yet achieving precise control over the model's output behavior remains challenging. TALE (Han et al., 2024) introduce, for the first time, the concept of a token budget. TOPS (Yang et al., 2025b) attempt to enable the model to autonomously determine the required effort for solving a given task. However, both TALE and TOPS fail to explicitly guide the model to produce the optimal token budget. They also fail to effectively control the output length according to a given token budget. L1 (Aggarwal & Welleck, 2025) and Elastic Reasoning (Xu et al., 2025b) can more precisely control the output length under a given token budget, yet they fail to enable the model to autonomously estimate an appropriate response length. Our proposed method enables the model to autonomously estimate the optimal token budget and subsequently generate text in strict adherence to it.

3 METHOD

To minimize the overthinking problem in LLMs, we propose SelfBudgeter for efficient reasoning. Our method aims to enable the model to autonomously determine an appropriate token budget and generate responses of corresponding length while adhering to this budget. Although reasoning models may occasionally overthink simple problems, their response lengths generally increase with problem difficulty. This phenomenon demonstrates that the model possesses the capability to allocate token quantities reasonably based on problem complexity. Previous works such as L1 (Aggarwal & Welleck, 2025) and Elastic Reasoning (Xu et al., 2025b) have also demonstrated that models can generate responses of appropriate length according to a given token budget.

Therefore, we design SelfBudgeter, which employs a reward function to guide the model in: (1) learning an output format where it first predicts a token budget before generating the answer, (2) allocating appropriate token budgets based on its own capabilities and question difficulty, and (3) generating solutions with optimal length while ensuring answer accuracy.

3.1 SELFBUDGETER

SelfBudgeter is a concise and efficient method for automatic precise length controlled. We design the Precise Budget Control Reward (PreB Reward) to achieve precise control over length. The detailed introduction of PreB Reward can be found in Section 3.3. We employ GRPO algorithm to train the model in predicting appropriate token budgets based on problem difficulty and generating responses with lengths conforming to the specified budget.

Our reward function is formally defined as Formula 1:

$$\mathrm{R}(C, F, \ell, b, b_{\text{max}}) = \begin{cases} r_f, & \text{if } F = 0, \\ \mathrm{P_B}(b, b_{\text{max}}) + \mathrm{PreB}(s^W_{\min}, s^W_{\max}, \ell, b, \alpha, b^W_{\text{best}}), & \text{if } F = 1 \text{ and } C = 0, \\ \mathrm{P_B}(b, b_{\text{max}}) + \mathrm{PreB}(s^C_{\min}, s^C_{\max}, \ell, b, \alpha, b^C_{\text{best}}), & \text{if } F = 1 \text{ and } C = 1. \end{cases} \tag{1}$$

where

$$b^C_{\text{best}} = (1 - \alpha) \cdot b, \quad b^W_{\text{best}} = (1 + \alpha) \cdot b \tag{2}$$

The inputs and hyperparameters in the reward function are listed in Table 1. To ensure stable prediction of the token budget prior to response generation, any responses deviating from the prescribed format will be assigned the minimum reward score of r_f.

3.2 BUDGET PENALTY

To enable the model to learn token budget allocation, we introduce a budget penalty module defined by Formula 3. The model incurs a penalty r_b when its estimated token budget exceeds the maximum acceptable budget b_{max}. No penalty is applied when the estimated token budget remains within b_{max}. A detailed introduction of b_{max} is presented in Section 4.2. Briefly stated, for a given question, b_{max} equals the response length if the base model can answer it correctly; otherwise, b_{max} is set to ∞.

$$\mathrm{P_B}(b, b_{\text{max}}) = \begin{cases} 0, & \text{if } b \le b_{\text{max}}, \\ r_b, & \text{else.} \end{cases} \tag{3}$$

Table 1: Input and Hyperparameters (HPs) in the reward function

Input	Description	HPs	Description
C	Correctness for answer	r_f	Penalty for format error
F	Correctness for format	$s_{\text{min}}^{W/C}$	Minimum reward (wrong/correct)
l	Response length	$s_{\text{max}}^{W/C}$	Maximum reward (wrong/correct)
b	Model's budget	α	Tightness coefficient of budget
b_{max}	Maximum acceptable budget	r_b	Penalty for excessive budget

3.3 Precise Budget Control Reward

Inspired by the cosine reward (Chang et al., 2025), we propose the Precise Budget Control Reward (PreB Reward). While the cosine reward helps mitigate overthinking tendencies, it lacks precise control over output length, as it only constrains the upper bound of the response. To address this limitation, we introduce a tightness coefficient α to better align the response length with the specified token budget.

Given the inherent challenge for models to precisely comply with token budgets, we relax the length constraint to require only approximate adherence within $\alpha \cdot b$ around the target budget b. As shown in Formula 4, when the model's response length falls outside the specified range, the corresponding reward score plummets to its minimum value s_{min}.

For incorrect responses, the function incentivizes longer reasoning chains (increasing length ℓ) to encourage deeper analysis that might lead to correct conclusions. Conversely, for correct answers, the reward peaks at the minimally sufficient length $(1-\alpha) \cdot b$ to prevent unnecessary computational overhead while maintaining accuracy. This explains why in Formula 2, the value of b_{best} differs between correct and incorrect responses from the model.This dual mechanism promotes efficient reasoning by adaptively modulating response lengths based on answer correctness.

$$\text{PreB}(s_{\min}, s_{\max}, \ell, b, \alpha, b_{\text{best}}) = \begin{cases} s_{\min}, & \text{if } \dfrac{|\ell - b|}{b} > \alpha, \\ s_{\min} + (s_{\max} - s_{\min}) \times & \\ \quad \dfrac{1}{2}\left(1 + \cos\left(\pi \cdot \dfrac{|\ell - b_{\text{best}}|}{2\alpha b}\right)\right), & \text{else.} \end{cases} \tag{4}$$

3.4 Accuracy Reward

To ensure the model's post-training accuracy does not degrade below its initial performance, we configure hyperparameters to guarantee that the minimum reward for correct responses always exceeds the maximum reward for incorrect responses. Specifically, our design ensures that: A correct response, which has a token budget exceeding b_{max} and receives the lowest budget following reward s_{min}^{C}, will yield a higher total reward than an incorrect response that has a token budget within b_{max} and receives the highest budget following reward s_{max}^{W}. This constraint is formally expressed as: $s_{\text{min}}^{C} + r_b \geq s_{\text{max}}^{W}$.

Overall, the core design of SelfBudgeter consists of three key modules: Budget Penalty, Preb Reward, and Accuracy Reward, which collectively balance length compression, correctness, and precise length control–ultimately delivering a better user experience.

4 Experiment

4.1 Training Template

The existing reasoning models utilize a pair of `<think></think>` tags to demarcate the thinking process from the final solution output. Building upon this format, we have further incorporated a token budget component.

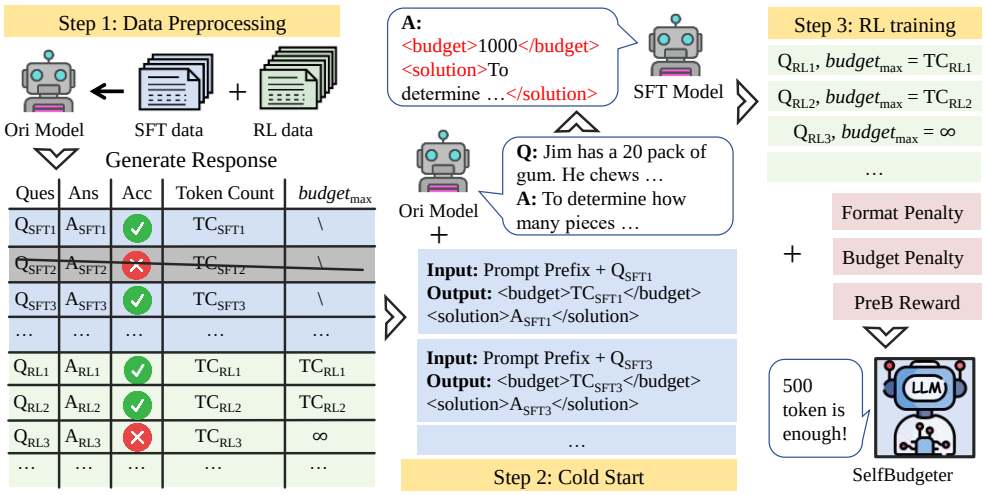

Figure 2: Overview of the SelfBudgeter training framework. The workflow consists of three key steps: (1) **Data preprocessing**: The initial model preprocesses the data to compute token budgets for subsequent training; (2) **Cold-start**: The model is fine-tuned to adopt the new output format; (3) **RL Training**: Through reward functions, the model learns to allocate token budgets and generate compliant outputs.

To enable the model to dynamically allocate token usage based on question difficulty, we design an output format as follows:

```
<budget>an integer</budget><solution>response</solution>
```

The format requires the model to first estimate the required token budget before providing the answer to the question. When no user constraint exists, the model autonomously predicts the token budget. When users specify a token limit, we pre-fill the <budget> field and let the model generate the <solution> within this constraint.

4.2 Data Preprocessing

At this stage, we collect model's responses to the test questions used in both the cold-start and RL training phases, and then evaluate the correctness and length of the responses.

For the cold-start data, we retain the model's correct responses along with their lengths and discard incorrect answers to prevent reinforcing the model's memory of wrong responses.

For the RL training data, we calculate $budget_{\max}$ (for convenience, we will refer to it as $b_{\max}$ in the following sections) using Formula 5, representing the maximum acceptable token budget for a given question. When the model answers correctly, the correctness of the response indicates that the minimum token budget required for a correct answer does not exceed the current length. Therefore, we encourage the model to further compress the response length and set $b_{\max}$ to the current response length. When the model answers incorrectly, the relationship between the minimum token budget needed for correctness and the current length remains unclear, so any token budget is acceptable.

$$b_{\max} = \begin{cases} \text{response length}, & \text{if model answers correctly}, \\ \infty, & \text{else}. \end{cases} \tag{5}$$

4.3 Cold Start

In our actual RL training process, we observe that requiring the model to simultaneously master multiple objectives - learning the new output format, providing appropriate token budgets, generating solutions of corresponding lengths according to the budget, while maintaining or improving accuracy

- proved excessively challenging. After extended training periods, the model often only succeeds in adopting the output format without achieving the other goals. Inspired by the Deepseek-R1 training methodology, we introduce a cold-start phase to accelerate training and enable the model to first learn the new output format before proceeding to more complex tasks. The overall training framework is illustrated in Figure 2.

To prevent the model from losing its original reasoning capability during the cold-start phase, fine-tuning must be performed using either the model's own generated responses or datasets containing long CoT responses. In our approach, we pre-populate the `<budget>` section with token counts obtained during the preprocessing stage. The `<solution>` section is filled with the model's generated responses. And the instruction prefix we prepend to each question can be found in Appendix B .

4.4 Experiment Settings

We conduct experiments on the DeepSeek-R1-Distill-Qwen-1.5B (R1-1.5B) model. We reproduce L1-Max using R1-1.5B, and select R1-1.5B and L1-Max as baseline methods for comparative evaluation against SelfBudgeter. In addition, we extend our experiments to the larger DeepSeek-R1-Distill-Qwen-7B (R1-7B) model. For more comprehensive comparison, we also include E1-Math-1.5B, R1-7B, Eurus-2-7B-PRIME (Cui et al., 2025), and Qwen-2.5-7B-Simple-RL (Shao et al., 2024) as additional baselines.

During the cold-start phase, we employ three datasets of varying difficulty—GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and s1k-1.1 (Muennighoff et al., 2025)—to help the model learn the new output format while producing token budgets with diverse distributions. The s1k-1.1 dataset contains 1,000 challenging mathematical problems with long reasoning chains generated by DeepSeek-R1, which support both reasoning ability and format adaptation. For GSM8K and MATH, we select 1,500 training samples each that the model can answer correctly. For s1k-1.1, we directly use the native responses and compute the corresponding token counts with the model's tokenizer to populate our designed template; in total, we retain 630 problems that DeepSeek-R1 answered correctly. This yields a training set of 3,630 samples. Following the preprocessing protocol in Sections 4.2 and 4.3, we fine-tune the model for one epoch. Throughout data collection and training, the model's temperature is consistently set to 0.6.

During the reinforcement learning phase, we use STILL-3-Preview-RL-Data (Chen et al., 2025b) dataset. It also serves as the training dataset for reproducing L1-max. This dataset collects 30K high-quality samples based on the MATH (Hendrycks et al., 2021), NuminaMathCoT (LI et al., 2024), and AIME 1983-2023 (Veeraboina, 2023) datasets. It includes problems of varying difficulty levels, which also helps the model learn to allocate token counts adaptively based on difficulty. As described in Section 4.2, we compute the maximum acceptable budget ($b_{\max}$) based on the model's responses, then train the model for 3 epochs on this dataset. More detailed information can be found in Appendix A.

4.5 Main Results

Table 2 presents a comprehensive comparison of model performance on the GSM8K, MATH500, and AIME2025 test sets, evaluated in terms of accuracy (Acc) and average response length (Len). The table contrasts baseline models with different variants of the SelfBudgeter framework across varying model scales. For clarity, the best performance is highlighted in bold, while the second-best performance is indicated with underline. It is worth noting that token limits for L1 are explicitly specified through prompt templates, whereas those for E1 are enforced via hard truncation. In contrast, SelfBudgeter autonomously estimates its token constraints during inference. All reported results are averaged over three runs with different random seeds.

Baseline Comparison Although the Deepseek-R1-Distill-Qwen-1.5B baseline demonstrates strong accuracy, it requires substantially longer responses. On GSM8K, our method improves accuracy by 11.01 percentage points while compressing response length to 43% of the original. On MATH500, it achieves a 3.54-point accuracy gain with response length reduced to 44%. On AIME2025, our approach compresses response length to 30% of the original while maintaining comparable accuracy. In contrast, although L1 and E1 attain stronger compression on certain datasets, they incur larger

Table 2: Performance comparison on GSM8K, MATH500, and AIME2025. Accuracy (Acc) is reported in percentage, and length (Len) in tokens.

Models	GSM8K		MATH500		AIME2025	
	Acc	Len	Acc	Len	Acc	Len
DeepSeek-R1-Distill-Qwen-1.5B	73.09	2865.08	74.93	5327.12	**22.22**	14444.03
E1-Math-1.5B(0.5K,1K)	60.20	1205.21	35.53	1499.54	4.44	3008.44
E1-Math-1.5B(4K,1K)	72.10	1299.62	72.47	2088.44	21.11	5578.13
L1-Max(3600)	79.56	571.72	76.73	1753.42	17.88	5213.89
SelfBudgeter-1.5B	**84.10**	1231.79	**78.47**	2326.85	21.11	4288.10
DeepSeek-R1-Distill-Qwen-7B	87.09	1918.21	86.73	5387.19	28.89	22158.79
Eurus-2-7B-PRIME	**90.98**	302.72	79.73	582.58	15.56	1254.52
Qwen-2.5-7B-Simple-RL	75.94	519.07	61.13	823.89	6.67	1429.94
SelfBudgeter-7B	90.30	991.13	**86.87**	2666.58	**30.00**	12241.84

accuracy losses—L1 performs poorly on the challenging AIME2025 benchmark, while E1 suffers more pronounced accuracy degradation on the simpler GSM8K and MATH500 datasets.

In addition, Table 2 highlights that SelfBudgeter consistently strikes a better balance between accuracy and response length than existing baselines. Unlike L1, which enforces explicit length limits but collapses on AIME2025, or E1, which relies on hard truncation and severely harms accuracy, SelfBudgeter autonomously learns effective token budgeting. As a result, it achieves the best or second-best accuracy across all datasets while simultaneously reducing response length substantially.

Beyond its effectiveness at the 1.5B scale, our method also delivers efficient reasoning with larger models. SelfBudgeter-7B achieves the highest accuracy on MATH500 and AIME2025, and the second-best accuracy on GSM8K—only 0.68 points lower than the best-performing model. Meanwhile, SelfBudgeter-7B attains an average compression ratio of 48%, further demonstrating the generality of our approach and its effectiveness at larger model scales. Compared with Eurus-2-7B-PRIME, which excels only on GSM8K but falls behind on harder reasoning tasks, and Qwen-2.5-7B-Simple-RL, which underperforms across all benchmarks, SelfBudgeter exhibits robust gains across datasets of varying difficulty.

4.6 Dynamic Alpha Schedule

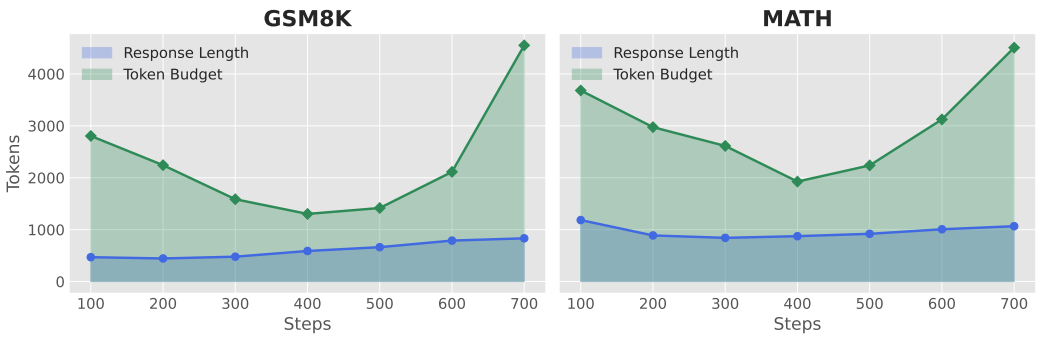

Figure 3: Response length and token budget on GSM8K and MATH benchmarks across training steps with $\alpha = 0.5$. The curves show how the average response length (solid circles) and allocated token budget (solid diamonds) evolve during training.

In SelfBudgeter, α serves as a critical hyperparameter. As shown in Figure 3, we observe that using a fixed and relatively loose α can lead to *reward hacking*: once the model learns to align the budget with the actual response length, it tends to inflate the predicted budget during later training stages, pushing the output length toward the lower bound of the acceptable range to obtain higher PreB

scores. Conversely, when α is fixed but relatively tight, the token budget quickly collapses to the response length, which hinders the model from learning an optimal budgeting strategy. To address these issues, we introduce a *dynamic alpha schedule*, where α is linearly decreased over training steps. This gradually tightens the tolerance range for acceptable response lengths and encourages closer convergence between the predicted budget and the actual output length. Consequently, the optimal α is not static but evolves throughout the training process.

Formally, the dynamic α is defined by a linear schedule:

$$\alpha_{\text{now}} = \alpha_{\text{start}} - (\alpha_{\text{start}} - \alpha_{\text{end}}) \cdot \frac{\text{step}_{\text{now}}}{\text{Total steps}}. \tag{6}$$

This schedule only requires specifying the starting and ending values of α (i.e., α_{start} and α_{end}), which are set to 6.0 and 0.1, respectively.

5 Discussion

In the Analysis section, we systematically examine SelfBudgeter's adaptive computation allocation mechanism through two pivotal aspects: its ability to dynamically adjust budgets according to problem complexity, and compliance with token constraints while preserving response quality. This holistic evaluation reveals fundamental insights into how adaptive language models negotiate computational efficiency with task requirements, informing both theoretical understanding and practical deployment considerations.

5.1 Adaptive Budget Allocation

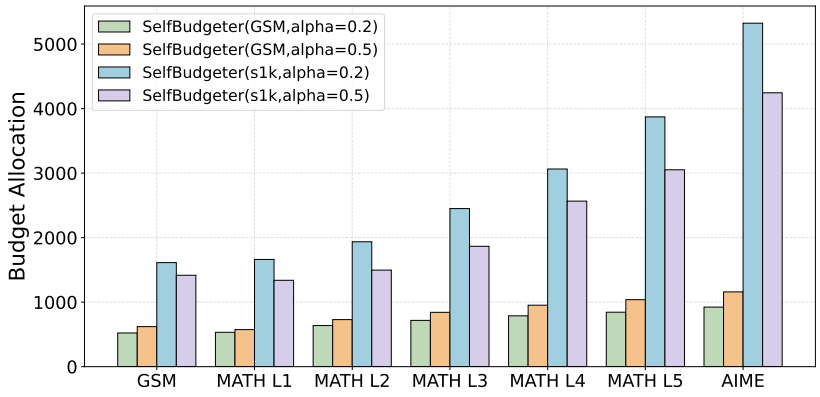

Figure 4: Token budget allocation patterns across problem difficulty levels for four SelfBudgeter-1.5B configurations (initialized on GSM8K/s1k with α=0.2/0.5). All variants exhibit monotonic budget escalation with increasing task complexity (GSM8K, MATH Level 1-5, AIME2024), confirming robust cross-configuration alignment between computational investment and intrinsic problem difficulty.

To investigate SelfBudgeter's capacity for difficulty-aware budget allocation, we conduct empirical evaluations across three mathematical reasoning benchmarks with inherent complexity gradients: GSM8K, MATH, and AIME 2024. Our experimental framework systematically evaluates four architectural variants combining cold-start initialization strategies (GSM8K vs. s1k) with α hyperparameter values (0.2 vs. 0.5).

Figure 4 shows a consistent positive correlation between problem complexity and allocated token budgets across all model variants, demonstrating SelfBudgeter's ability to scale computation with task difficulty. The near-linear allocation across difficulty tiers highlights its emergent capacity for intrinsic difficulty estimation, while the minimal variance across configurations indicates robust and generalized learning of task-complexity metrics rather than configuration-specific artifacts.

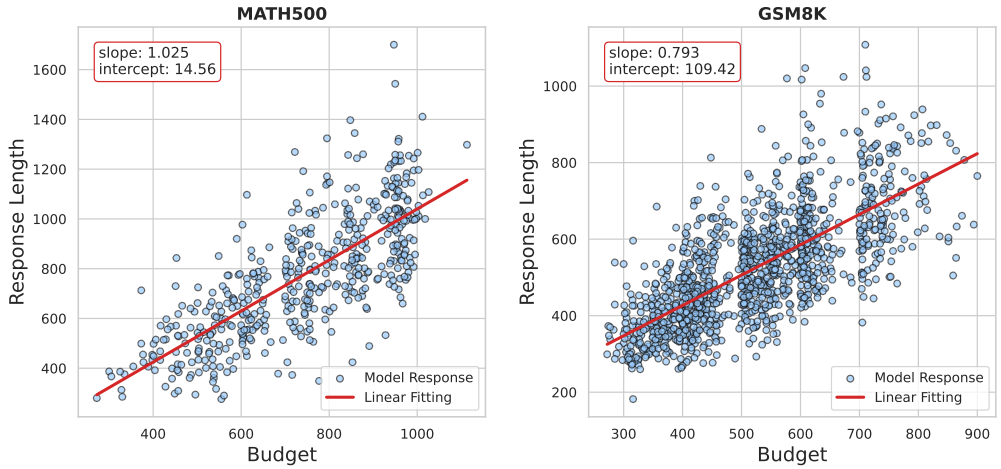

Figure 5: Linear regression analysis of token budget prediction versus actual response length for SelfBudgeter on the MATH500 dataset(left) and GSM8K test set(right). The figure demonstrates SelfBudgeter with GSM initialization and hyperparameter $\alpha = 0.2$.

5.2 Budget-Constrained Generation

To systematically evaluate the generation capability of SelfBudgeter under budget constraints, this study employs linear regression modeling to quantitatively analyze the mapping relationship between predicted token budgets and actual response lengths. We conduct a quantitative analysis on the MATH500 dataset and GSM8K test set using linear regression to investigate the mapping between predicted budgets and actual response lengths (as shown in the Figure 5). On MATH500 dataset, the least squares fitting yields a slope of 1.025 (95% CI [0.9466, 1.1042]). And on GSM8K test set, the least squares fitting yields a slope of 0.793 (95% CI [0.7512, 0.8354]). The slope coefficient approaching unity validates the efficacy of the budget control mechanism, indicating that each 1-token increase in the predicted budget corresponds to an average increase of about 1-token in output.

Quantitative results demonstrate that 96% of generated responses exhibit relative deviations $\leq 50\%$ from the target token budget, with 65.40% achieving tighter deviations $\leq 20\%$. Extended experiments on full benchmark datasets reveal that 97.65% (GSM8K) and 95.82% (MATH) of samples satisfy the $\leq 50\%$ relative deviation constraint. Notably, the model's budget adherence is influenced by the cold-start dataset and hyperparameter α. The optimized SelfBudgeter configuration (initialized with GSM8K and $\alpha = 0.2$), which balances generation quality and budget compliance, is reported here as the best-performing variant.

We further validate SelfBudgeter's adherence to *user-defined* token budgets through controlled experiments. The results indicate that the actual generated length follows a linear functional relationship with user-defined budgets, demonstrating robust alignment even under explicit external constraints. Details are provided in Appendix C.

6 Conclusion

We propose the SelfBudgeter framework, which autonomously predicts required token budgets for reasoning while effectively adhering to self-imposed constraints, successfully optimizing the accuracy-response length trade-off. By leveraging SelfBudgeter's token budget predictions, users can anticipate total inference duration in advance, significantly enhancing user experience. In resource-efficient reasoning, SelfBudgeter demonstrates performance comparable to several existing methods, highlighting its potential for deployment in resource-constrained environments. Additionally, output length can be dynamically regulated through transformation functions when required. SelfBudgeter paves a promising pathway toward more efficient, controllable, and user-friendly reasoning models.

ETHICS STATEMENT

This work uses only publicly available datasets under their original licenses, and does not involve human subjects, private data, or personally identifiable information. Our contributions are methodological, focusing on improving reasoning efficiency, and do not amplify risks of harmful or biased content. We declare no conflicts of interest or ethical concerns, and we have complied with the ICLR Code of Ethics throughout the research and submission process. Additional details regarding the use of large language models (LLMs) are provided in Appendix D.

REPRODUCIBILITY STATEMENT

We have made extensive efforts to ensure the reproducibility of our work. The datasets used in our experiments are publicly available. Detailed descriptions of data preprocessing, training settings, and evaluation protocols are provided in Section 4, with additional implementation details and hyperparameters included in the appendix. We will release anonymous source code and scripts for training and evaluation as supplementary material.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.

Daman Arora and Andrea Zanette. Training language models to reason efficiently. *CoRR*, abs/2502.04463, 2025. doi: 10.48550/ARXIV.2502.04463. URL https://doi.org/10.48550/arXiv.2502.04463.

Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*, 2025.

Edward Y. Chang, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. *CoRR*, abs/2502.03373, 2025. doi: 10.48550/ARXIV.2502.03373. URL https://doi.org/10.48550/arXiv.2502.03373.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wangxiang Che. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*, 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, 2024. URL https://arxiv.org/abs/2412.21187.

Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. An empirical study on eliciting and improving r1-like reasoning models. *arXiv preprint arXiv:2503.04548*, 2025b.

Yu-Neng Chuang, Helen Zhou, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, and Xia Hu. Learning to route with confidence tokens. *CoRR*, abs/2410.13284, 2024. doi: 10.48550/ARXIV.2410.13284. URL https://doi.org/10.48550/arXiv.2410.13284.

Yu-Neng Chuang, Leisheng Yu, Guanchu Wang, Lizhe Zhang, Zirui Liu, Xuanting Cai, Yang Sui, Vladimir Braverman, and Xia Hu. Confident or seek stronger: Exploring uncertainty-based on-device llm routing from benchmarking to generalization. *arXiv preprint arXiv:2502.04428*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

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.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

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. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness. *CoRR*, abs/2412.11664, 2024. doi: 10.48550/ARXIV.2412.11664. URL https://doi.org/10.48550/arXiv.2412.11664.

Ayeong Lee, Ethan Che, and Tianyi Peng. How well do llms compress their own chain-of-thought? A token complexity approach. *CoRR*, abs/2503.01141, 2025. doi: 10.48550/ARXIV.2503.01141. URL https://doi.org/10.48550/arXiv.2503.01141.

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.

Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng Zhang. Can language models learn to skip steps? In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/504fa7e518da9d1b53a233ed20a38b46-Abstract-Conference.html.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *CoRR*, abs/2501.12570, 2025a. doi: 10.48550/ARXIV.2501.12570. URL https://doi.org/10.48550/arXiv.2501.12570.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl, 2025b. Notion Blog.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. Cot-valve: Length-compressible chain-of-thought tuning. *CoRR*, abs/2502.09601, 2025. doi: 10.48550/ARXIV.2502.09601. URL https://doi.org/10.48550/arXiv.2502.09601.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *CoRR*, abs/2501.19393, 2025. doi: 10.48550/ARXIV.2501.19393. URL https://doi.org/10.48550/arXiv.2501.19393.

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. Self-training elicits concise reasoning in large language models. *CoRR*, abs/2502.20122, 2025. doi: 10.48550/ARXIV.2502.20122. URL https://doi.org/10.48550/arXiv.2502.20122.

Sania Nayab, Giulio Rossolini, Giorgio C. Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. Concise thoughts: Impact of output length on LLM reasoning and cost. *CoRR*, abs/2407.19825, 2024. doi: 10.48550/ARXIV.2407.19825. URL https://doi.org/10.48550/arXiv.2407.19825.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *CoRR*, abs/2406.18665, 2024. doi: 10.48550/ARXIV.2406.18665. URL https://doi.org/10.48550/arXiv.2406.18665.

OpenAI. Learning to reason with llms, September 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason: Fast and accurate inference-time compute via speculative reasoning, 2025. URL https://arxiv.org/abs/2504.07891.

Qwen. Qwq: Reflect deeply on the boundaries of the unknown, November 2024. URL https://qwenlm.github.io/blog/qwq-32b-preview/.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In *2nd International Conference on Foundation and Large Language Models, FLLM 2024, Dubai, United Arab Emirates, November 26-29, 2024*, pp. 476–483. IEEE, 2024. doi: 10.1109/FLLM63129.2024.10852493. URL https://doi.org/10.1109/FLLM63129.2024.10852493.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024. doi: 10.48550/ARXIV.2408.03314. URL https://doi.org/10.48550/arXiv.2408.03314.

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

Hemish Veeraboina. Aime problem set 1983-2024, 2023. URL https://www.kaggle.com/datasets/hemishveeraboina/aime-problem-set-1983-2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in llms. *CoRR*, abs/2502.07266, 2025. doi: 10.48550/ARXIV.2502.07266. URL https://doi.org/10.48550/arXiv.2502.07266.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. *CoRR*, abs/2502.12067, 2025. doi: 10.48550/ARXIV.2502.12067. URL https://doi.org/10.48550/arXiv.2502.12067.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *CoRR*, abs/2502.18600, 2025a. doi: 10.48550/ARXIV.2502.18600. URL https://doi.org/10.48550/arXiv.2502.18600.

Yuhui Xu, Hanze Dong, Lei Wang, Doyen Sahoo, Junnan Li, and Caiming Xiong. Scalable chain of thoughts via elastic reasoning, 2025b. URL https://arxiv.org/abs/2505.05315.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Zheng Lin, Li Cao, and Weiping Wang. Dynamic early exit in reasoning models, 2025a. URL https://arxiv.org/abs/2504.15895.

Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. Towards thinking-optimal scaling of test-time compute for llm reasoning. *arXiv preprint arXiv:2502.18080*, 2025b.

A Training Details

A.1 Experimental Environments

Our server is equipped with two 80GB A100 GPUs and two 45GB A40 GPUs. We conducted fine-tuning experiments and inference tests on the two A40 GPUs, while the GRPO training was performed on the two A100 GPUs.

A.2 Parameter Settings

In the fine-tuning training during the cold-start phase, our parameter settings are configured as follows. The sequence length is capped at 16,384, with a per-device training and evaluation batch size of 1, while gradient accumulation (2 steps) is employed to alleviate GPU memory constraints. A cosine learning rate scheduler is adopted with a 10% warm-up ratio and a base learning rate of 5e-5. The model is trained for 1 epoch, with 10% of the training set allocated for validation. The model checkpoints are saved and evaluated every 500 steps, and the best-performing checkpoint is retained.

In the GRPO (Global Reward Policy Optimization) training, our parameter configuration is set as follows. The training and validation batch sizes are set to 128 and 1,250, respectively, with maximum prompt and response lengths of 1,024 and 32,000 tokens. The Actor model employs a learning rate of 1e-6, dynamic batching (up to 24K tokens per GPU), and a KL divergence loss (coefficient 0.001), with gradient checkpointing and FSDP (Fully Sharded Data Parallel) distributed training enabled (parameter offloading disabled). During the Rollout phase, the vLLM inference engine is utilized with tensor parallelism (TP=2) and 80% GPU memory utilization, generating 5 responses per round. Global settings include 3 training epochs, a checkpoint-saving interval of 50 steps, and a KL control coefficient of 0.001, executed on a single node with dual GPUs. And key hyperparameters involved in the reward function are specified in Table 3.

Table 3: Hyperparameters Settings

Parameters	$C = 0$	$C = 1$	**Parameters**	**Value**
s_{min}	-0.5	0.5	r_f	-1
s_{max}	0	1	r_b	-0.4

For the GSM-initialized SelfBudgeter, we select the checkpoint after 699 training steps when alpha was set to 0.2, and the checkpoint after 575 steps when alpha was 0.5. For the s1k-initialized SelfBudgeter, we choose the checkpoint after 475 training steps with alpha=0.2, and the checkpoint after 500 steps with alpha=0.5. For L1-Max, we choose the checkpoint after 280 training steps.

B Cold-start Data Selection

Prompt Template

Answer the given question. You should first estimate the total number of tokens you will need to answer this question based on its difficulty. Then you think about the reasoning process in the mind and provide the user with the answer. The token budget and whole solution are enclosed within <budget></budget> and <solution> </solution> tags, respectively, i.e., <budget> token budget here, just an integer </budget><solution> solution here, please output the final answer within \boxed{} </solution>.
Question:

Figure 6: The prompt template used in the cold-start stage.

The choice of initialization data substantially impacts model performance. SelfBudgeters initialized with the **s1k** dataset outperform their GSM-initialized SelfBudgeters by 8.82–10.72 percentage points

Table 4: Model performance comparison on GSM8K and MATH test sets, showing accuracy (Acc/%), average response length (Len/tokens) and matching rate between token limits and response length (Mat/%). The SelfBudgeter variants with different cold-start data and α parameters are contrasted with baseline models.

Model	GSM8K			MATH		
	Acc↑	Len↓	Mat↑	Acc↑	Len↓	Mat↑
Cold Start (GSM)	71.95	1003.79	85.82	64.74	3043.29	41.16
SelfBudgeter (GSM, $\alpha = 0.2$)	76.27	523.77	**97.65**	63.46	779.54	95.82
SelfBudgeter (GSM, $\alpha = 0.5$)	74.68	520.82	96.97	63.78	777.80	**96.66**
Cold Start (s1k)	82.49	1983.29	21.76	76.64	4001.29	23.28
SelfBudgeter (s1k, $\alpha = 0.2$)	**81.50**	662.08	70.74	**74.18**	919.27	78.36
SelfBudgeter (s1k, $\alpha = 0.5$)	80.44	719.36	71.19	72.60	1022.99	79.76

on MATH (74.18% vs. 63.46% for $\alpha = 0.2$) and 5.23–5.76 percentage points on GSM8K (80.44% vs. 74.68% for $\alpha = 0.5$). While SelfBudgeters with GSM-initialized exhibit lower accuracy, they generate significantly more concise responses compared to s1k-initialized SelfBudgeters. Specifically, GSM-initialized SelfBudgeters reduces response length by approximately 15–24% on MATH and achieves 21–28% length reduction on GSM8K. This performance gap highlights the importance of high-quality initialization for the budgeting mechanism.

As shown in Table 4, significant performance variations exist between models fine-tuned with different cold-start datasets. The s1k-fine-tuned model demonstrates superior accuracy over the GSM-fine-tuned counterpart, achieving 10.54% and 11.90% higher accuracy on GSM8K and MATH respectively. However, this comes at the cost of substantially longer responses, with the s1k model generating 97.58% and 31.48% lengthier outputs on GSM8K and MATH. This discrepancy stems from the s1k dataset's responses being generated by Deepseek-R1, which produces higher-quality outputs than those self-generated by Deepseek-R1-Distill-Qwen-1.5B. Additionally, the s1k dataset's average length of 7,677.43 tokens (we only retained correct responses under 16,000 tokens) vastly exceeds GSM8K's 837.14 tokens, explaining the dramatic difference in response lengths after fine-tuning. These factors substantially influence SelfBudgeter's final performance, as evidenced by: (1) SelfBudgeter's accuracy closely mirroring that of its fine-tuned base model, and (2) the response length relationships and matching rate relationships between different SelfBudgeter variants remaining consistent with their respective cold-start models.

C Prefilled Token Budget Following

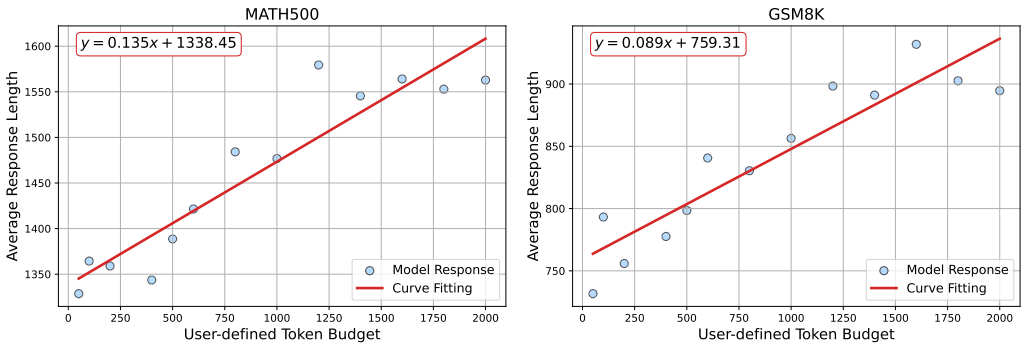

Figure 7: Relationship between user-defined token budgets and SelfBudgeter average response lengths with curve fitting analysis.

To systematically evaluate model performance under user-defined token budget constraints, we conduct quantitative analysis using SelfBudgeter with GSM initialization and hyperparameter $\alpha = 0.2$

on both MATH500 dataset and GSM8K test set. In the experimental design, fixed token budgets were pre-filled in the <budget> field of training templates, with empirical results obtained by measuring average generated response lengths. We evaluated SelfBudgeter's performance with user-defined token budgets ranging from 50 to 2000 (specifically: 50, 100, 200, 400, 500, 600, 800, 1000, 1200, 1400, 1600, 1800, and 2000), as shown in the Figure 7.

Regression intercepts effectively reflect problem complexity, where GSM8K's simpler questions yield significantly smaller intercepts. Despite a moderate slope, SelfBudgeter demonstrates robust budget adaptability, maintaining a stable positive correlation between user-defined budgets and output lengths. This linear relationship enables deterministic length control through derived transformation functions.

D THE USE OF LARGE LANGUAGE MODELS

During the preparation of this paper, large language models (LLMs) were used solely as auxiliary tools for grammar checking, text polishing, and improving clarity of exposition. No experimental design, data analysis, or substantive research conclusions were generated by LLMs. All methodological and experimental contributions are original and conducted entirely by the authors.