

Token-Budget-Aware LLM Reasoning

Tingxu Han^{*1}, Zhenting Wang^{*†2}, Chunrong Fang^{‡1},
 Shiyu Zhao², Shiqing Ma³, Zhenyu Chen¹

¹Nanjing University ²Rutgers University ³UMass Amherst

Abstract

Reasoning is critical for large language models (LLMs) to excel in a wide range of tasks. While methods like Chain-of-Thought (CoT) reasoning and enhance LLM performance by decomposing problems into intermediate steps, they also incur significant overhead in token usage, leading to increased costs. We find that the reasoning process of current LLMs is unnecessarily lengthy and it can be compressed by including a reasonable token budget in the prompt, but the choice of token budget plays a crucial role in the actual compression effectiveness. We then propose a token-budget-aware LLM reasoning framework that dynamically adjusts the number of reasoning tokens based on the reasoning complexity of each problem. Experiments show that our method effectively reduces token costs in CoT reasoning with only a slight performance reduction, offering a practical solution to balance efficiency and accuracy in LLM reasoning. Code: <https://github.com/GeniusHTX/TALE¹>.

*“It is not enough to have a good mind;
 the main thing is to use it well.”*

— René Descartes

1 Introduction

Reasoning plays a crucial role in enabling large language models (LLM) to perform effectively across a wide range of tasks (Zhou et al., 2022; Hao et al., 2023, 2024a; Jin et al., 2024a; Wang et al., 2024b, 2025). A variety of methods have been proposed to enhance the reasoning capabilities of large language models (Suzgun et al., 2022; Wang et al., 2023; Feng et al., 2023; Xie et al., 2024). Among these, Chain-of-Thought (CoT) (Wei et al., 2022)

is the most representative and widely adopted approach. It enhances the reliability of the model’s answers by guiding large language models with the prompt “Let’s think step by step”, encouraging them to decompose the problem into intermediate steps and solve each before arriving at the final answer. [Figure 1a](#) and [Figure 1b](#) illustrate an intuitive example. Observe that without CoT, the LLM produces incorrect answers to the question. With a CoT-enhanced prompt, the LLM systematically breaks the question into multiple steps and reasons through each step sequentially. By addressing each step incrementally, the LLM eventually arrives at the correct answer. Recent reasoning models, such as OpenAI O1 ([OpenAI, 2024c](#)) and DeepSeek R1 ([Guo et al., 2025](#)), integrate CoT into their design. Notably, these models can perform CoT reasoning even without explicit prompting.

Although reasoning enhancement approaches such as CoT impressively improve LLM performance, they produce substantial additional overhead, specifically in the form of the increased number of tokens produced (Wei et al., 2022; Feng et al., 2023; Yao et al., 2024a; Jin et al., 2024b). As shown in [Figure 1b](#), the answer to prompt with CoT has notably higher token costs due to the detailed intermediate reasoning steps included in the output. Such high token costs can lead to significant expenses, including increased computational resource usage and longer running times during the LLM inference, ultimately resulting in significant additional monetary and energy costs.

This raises an important question: *“Is the reasoning process of current LLMs unnecessarily lengthy, and how can it be compressed?”* Nayab et al. (2024) demonstrate that LLM has the potential to follow a length constraint in the prompt. Building on this, we find that *including a token budget (see [Table 1](#)) in the prompts is a promising approach to compressing the CoT reasoning tokens. However, the choice of token budget plays a crucial role in*

^{*}Equal Contribution.

[†]Start the project and propose the idea.

[‡]Corresponding Author.

¹Also available at <https://www.gitlink.org.cn/txhan/TALE>

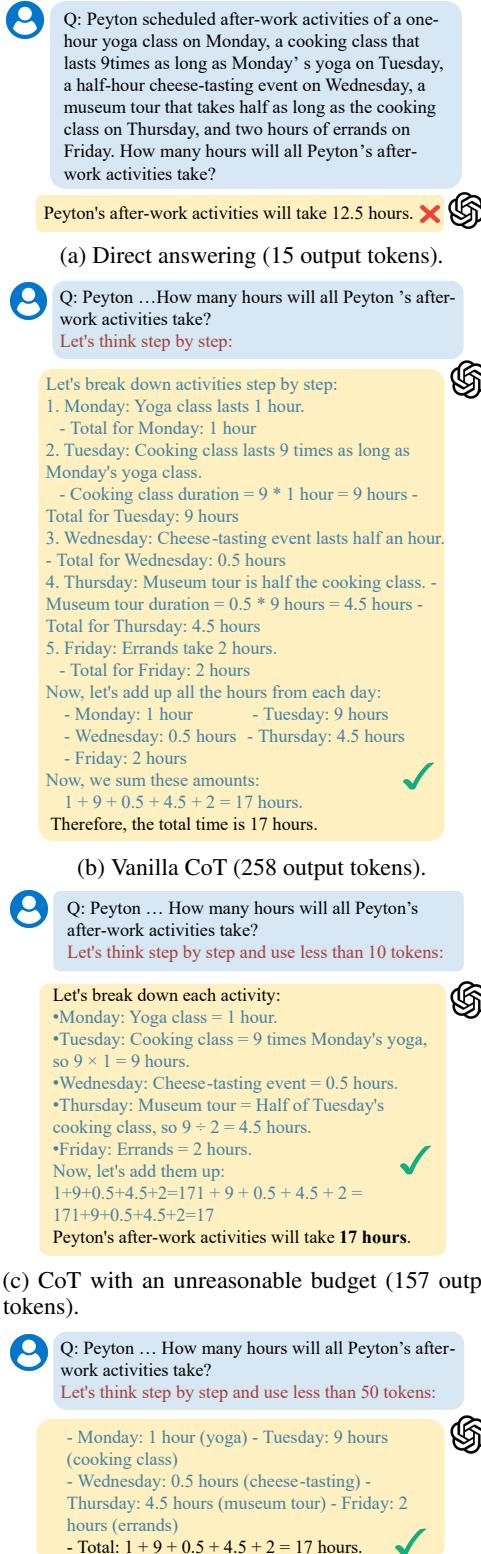


Figure 1: Examples of different problem solving paradigms. The reasoning processes are highlighted. We conduct this experiment on GPT-4o-mini.

the actual compression effectiveness. For example, Figure 1d illustrates that including a reasonable token budget (e.g., 50 tokens in this case) in the instructions reduces the token cost in the chain-of-

thought (CoT) process from 258 output tokens to 86 output tokens, while still enabling the LLM to arrive at the correct answer. However, when the token budget is set to a different smaller value (e.g., 10 tokens), the output token reduction is less effective, resulting in 157 output tokens—nearly twice as many as with a 50-token budget. In other words, when the token budget is relatively small, LLMs often fail to follow the given token budget. In such cases, the actual token usage significantly exceeds the given budget—even much larger than the token costs with larger token budgets. We refer to this phenomenon as the “Token Elasticity” in the CoT process with token budgeting. To address this, the optimal token budget for a specific LLM and a particular question can be searched by gradually reducing the budget specified in the prompt, identifying the smallest token budget that achieves both the correct answer and the lowest actual token cost.

Based on the above observations and analysis, we propose a token-budget-aware LLM reasoning framework that dynamically adjusts the number of reasoning tokens based on the reasoning complexity of each problem. We call our method TALE (Token-Budget-Aware LLM rEasoning), which includes two implementations: token budget estimation and prompting (TALE-EP) and token budget awareness internalization via post-training (TALE-PT). TALE-EP estimates a reasonable token budget for each problem using zero-shot prompting and incorporates it into the reasoning process, while TALE-PT internalizes token-budget awareness through post-training, enabling the LLM to generate more token-efficient responses without explicit token constraints in the prompt. We discuss both implementations in Section 5. Experiment results show that TALE significantly reduces token costs in LLM chain-of-thought (CoT) reasoning while largely maintaining answer correctness. On average, TALE-EP achieves a 67% reduction in token usage while maintaining accuracy with less than a 3% decrease. TALE-PT cuts token usage by around 50% compared to Vanilla CoT and achieves competitive performance.

2 Related Work

LLM Reasoning. Reasoning in LLMs has seen substantial advancements through techniques that generate intermediate steps, enabling more accurate and effective performance across diverse domains (Wu et al., 2022; Yang et al., 2022; Zhou et al., 2022; Sun et al., 2024; OpenAI, 2024c). Var-

ious LLM reasoning techniques are proposed to improve the LLM performance. Chen et al. (2024) formulates reasoning as sampling from a latent distribution and optimizing it via variational approaches. Ho et al. (2022) utilizes LLM as reasoning teachers, improving the reasoning abilities of smaller models through knowledge distillation. Among them, Chain-of-Thought (CoT) prompting has emerged as a key technique for improving LLM reasoning by breaking problems into intermediate steps, enabling better performance on multiple tasks (Wei et al., 2022; Lyu et al., 2023; Li et al., 2023; Feng et al., 2024). Extensions of CoT include self-consistency, which aggregates multiple reasoning paths to improve robustness (Wang et al., 2022), and Tree-of-Thoughts, which explores reasoning steps in a tree-like structure for more complex tasks (Yao et al., 2024b). Reflexion introduces iterative refinement, where the model critiques and updates its intermediate steps (Shinn et al., 2024). **Token Cost of LLM.** Although the above methods enhance reasoning accuracy, they often increase token usages, posing challenges to efficiency (Wang et al., 2024a; Chiang and Lee, 2024; Bhargava et al., 2023). Consequently, it is important to mitigate token consumption while maintaining the model performance. To address this issue, Li et al. (2021) introduces a multi-hop processing technique designed to filter out irrelevant reasoning. While effective, this approach is limited to traditional neural networks, such as PALM (Bi et al., 2020), and lacks adaptability to large language models (LLMs). Speculative decoding (Leviathan et al., 2023) aims to accelerate decoding by generating drafts using smaller models and verifying them with larger models, which is over-dependent on the alternative small approximation model. LLM routing (Ding et al., 2024) queries to different LLMs based on quality-cost trade-offs, but it cannot reduce the token usage on the specific LLM for a given query. Zheng et al. (2024) aims to improve LLM inference speed by predicting response lengths and applying a scheduling algorithm to enhance efficiency. However, it is constrained to scheduling level, and it does not reduce the actual token costs. Hao et al. (2024b) reduces token usage by substituting decoded text tokens with continuous latent tokens. However, its application is currently restricted to small-scale, early language models like GPT-2 (Radford et al., 2019). Additionally, it significantly impacts reasoning accuracy, resulting in over a 20% relative accuracy reduction on

Table 1: Illustrations of the vanilla CoT prompt and the token-budget-aware prompt.

Prompt method	Content
Vanilla CoT	Let’s think step by step:
CoT with Token Budget	Let’s think step by step and use less than budget tokens:
Example	Let’s think step by step and use less than 50 tokens:

benchmarks such as GSM8K (Cobbe et al., 2021).

3 Token Redundancy in LLM Reasoning

Token Budget. Previous research (Nayab et al., 2024) demonstrates that LLM has the potential to follow a length constraint in the prompt. Table 1 shows the difference between the vanilla CoT and the CoT with token budget. For instance, by including a token budget (50 tokens) within the prompt, as illustrated in Figure 1d, the LLM adjusts the length of its output (86 output tokens), trying to align with the specified budget. This indicates that LLMs have a certain capability in following prompts with an explicit token budget.

Token Redundancy Phenomenon. We find that providing a reasonable token budget can significantly reduce the token cost during reasoning. As shown in Figure 1d, including a token budget in the instructions reduces the token cost in the chain-of-thought (CoT) process by several times, but the LLM still gets the correct answer. Our results in Figure 2 and Table 3 also confirm there are a large number of redundant tokens in the reasoning process of the state-of-the-art LLMs.

Causes of Token Redundancy in LLM Reasoning. A possible explanation for this token redundancy is that during the post-training phase, such as the RLHF process (Ouyang et al., 2022), annotators might favor more detailed responses from LLMs, marking them as preferred. As a result, the model learns to associate longer, more detailed responses with alignment to human preferences and tends to produce such outputs during reasoning. However, in many scenarios, we primarily need LLMs to provide the correct answer and make accurate decisions, rather than elaborate extensively with detailed explanations. This motivates the need to eliminate redundant tokens in the LLM reasoning process in many cases.

4 Searching Optimal Token Budget

As demonstrated in Figure 1, different token budgets have different effects. Therefore, it is natural to investigate the following question: “How to search

Algorithm 1 Budget Search

Input: feasibility checking function `isFeasible`, a large language model \mathcal{M} , a given question x and the ground truth label y

Output: searched budget β

```
1: function SEARCH(isFeasible,  $\mathcal{M}$ ,  $x$ ,  $y$ )
2:    $right \leftarrow$  the actual token costs of  $\mathcal{M}$  with
      vanilla CoT prompt on  $x$ 
3:    $\beta \leftarrow \lfloor (0 + right) / 2 \rfloor$ 
4:    $\beta_0 \leftarrow right$ 
5:   while True do
6:     if isFeasible( $\mathcal{M}$ ,  $x$ ,  $y$ ,  $\beta_0$ ,  $\beta$ ) then
        ▷ Update the searched budget
7:        $\beta \leftarrow \lfloor (0 + right) / 2 \rfloor$ 
        ▷ Record previous searched budget
8:        $\beta_0 \leftarrow right$ 
        ▷ Update the search range
9:        $right \leftarrow \beta$ 
10:    else
11:      break
12:   return  $\beta$ 
```

the optimal token budget for a specific question and a particular LLM?”

Vanilla Method for Optimal Budget Search. An intuitive method is finding the minimal needed tokens as the budget, ensuring that the LLM can still produce correct and accurate responses within this constraint. The goal of the search algorithm is not to directly determine task difficulty, but to identify the minimum token budget under which the model can still produce a correct answer. Specifically, the algorithm conducts a binary search over different token budgets and selects the shortest budget that maintains correctness. This approximates the minimum reasoning length required for the model to solve the given problem. To find the minimal token budget, we first propose an “implicit monotonicity assumption” that when the model outputs a wrong prediction at a budget value, it always predicts incorrectly when under this budget value, and when the model outputs a correct prediction at a budget value, it always predicts correctly when above this budget value. To empirically assess the validity of this assumption, we conduct an additional analysis. Specifically, we define a sample as monotonic if all predictions above the optimal budget are correct, and all predictions below it are incorrect. We randomly sample from the GSM8K dataset as test data and find that 90.91% of the samples satisfy this monotonicity condition. This suggests that

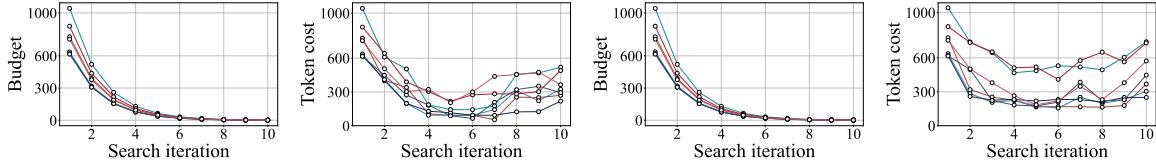
Table 2: An intuitive monotonic example. β^* is the searched optimal budget. The budget row displays scaled budgets ranging from 2^{-2} to $2^2 \cdot \beta^*$.

Budget(* β^*)	2^{-2}	2^{-1}	1	2^1	2^2
Prediction	False	False	True	True	True

while the assumption may not strictly hold in every instance, it is a reasonable and effective approximation in practice for guiding budget search. An intuitive monotonic sample is illustrated as Table 2. Based on the “implicit monotonicity assumption”, we further design a binary search-based minimal budget search algorithm detailed in Algorithm 1.

Before initiating the search process, we first apply the vanilla CoT to generate an answer for each question, as illustrated in Figure 1b. The number of tokens in the resulting answer is then calculated and designated as the right boundary for search, denoted by $right$. The function `isFeasible` is used to determine the feasibility of a budget. A budget is considered feasible here if the CoT prompt with that budget preserves the correctness of the answer. Algorithm 1 showcases the details. Given the feasibility function, large language model \mathcal{M} , question x and label y as the input, Algorithm 1 first calculates the right boundary of search (line 2). With 0 as the left boundary, the current possible budget β is computed as the midpoint of 0 and $right$ (line 3). We use β_0 to record the previously searched budget (line 4). While the current β is feasible, the algorithm updates β by recalculating the midpoint (line 7) and adjusts the search bounds accordingly to narrow the range (line 9). Once the loop ends, the final budget β is returned as the searched result (line 12). Algorithm 1 is designed to find the minimal budget efficiently. However, we observe that the minimal budget required to produce a correct answer is not necessarily the optimal budget. When the budget is unreasonably small, the actual token cost often exceeds that of cases where a larger budget is used. We also further formalize the optimal budget search process detailed in Section A.6.

Observation of Token Elasticity. During our minimal budget search process, we observe a “*token elasticity*” phenomenon as we approach the minimal budget. Specifically, as Algorithm 1 progresses, we aim to identify the minimal budget that still ensures the answer’s correctness. However, we find that if the budget is reduced beyond a certain range, the token cost increases, indicating that further reductions in the budget lead to increasing to-



(a) GPT-4o-mini budget search. (b) GPT-4o-mini token cost. (c) Yi-lightning budget search. (d) Yi-lightning token cost.

Figure 2: Token elasticity phenomenon. The x-axis denotes the budget search iteration. The y-axis denotes the searched budget (Figure 2a and Figure 2c) or the real token costs for each searched budget (Figure 2b and Figure 2d). Different colors denote different samples randomly selected from MathBench-College (Liu et al., 2024). The token cost is significantly lower in a reasonable token budget range. When the token budget is smaller than the reasonable range, the token cost gradually increases.

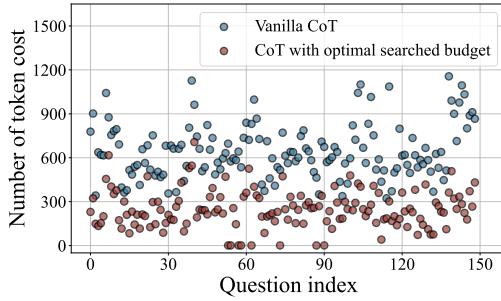


Figure 3: The effects of optimal searched budget. CoT, with our optimal searched budget, reduces the token costs significantly without influencing the accuracy. We conduct it on MathBench-College (Liu et al., 2024).

ken consumption. Figure 2 showcases the evidence. The x-axis represents the iterations of the budget binary search, with the budget values decreasing progressively. The y-axis in Figure 2b and Figure 2d show the corresponding token costs at each budget search iteration. When the searched budget drops below a reasonable range, the token cost increases. This happens because the model, unable to meet the tight constraint, ignores it and reverts to longer reasoning. In other words, the model effectively “gives up” on complying with the instruction, resulting in longer outputs and redundant token costs. This explains the non-monotonicity observed in Figure 2: token usage initially decreases as the budget tightens but eventually rebounds when the budget becomes too small to be feasible. We will make this intuition clearer in the revised version.

Figure 1c also shows an example. As observed, when a small token budget (e.g., 10 tokens) is used, the real token cost is significantly higher compared to scenarios where a reasonable token budget is allocated (i.e., Figure 1d).

Token Elasticity based Optimal Budget Search. The token elasticity observation shows that while a minimal budget may keep the correctness of the answer, it does not necessarily minimize the token

Algorithm 2 Greedy Feasibility Function

Input: a large language model \mathcal{M} , a question x and the ground truth label y , previous and current budget: β_0, β

Output: True if the budget satisfies the requirements, False otherwise

```

1: function isFeasible( $\mathcal{M}, x, y, \beta_0, \beta$ )
2:    $t, t_0 \leftarrow$  gets the actual token costs under budgets of  $\beta$  and  $\beta_0$ 
3:   if  $\mathcal{M}(x, \beta) == y$  and  $t < t_0$  then
4:     return True
5:   return False

```

cost. Figure 1c and Figure 1d illustrate an intuitive example. To address this, we enhance Algorithm 1 by incorporating a greedy search strategy aimed at finding the optimal budget that simultaneously minimizes token cost and preserves answer correctness. Specifically, we introduce an additional constraint to the isFeasible condition. Beyond ensuring correctness, the updated budget must result in a lower token cost compared to the previously searched budget. Algorithm 2 outlines the feasibility function employed during the search process. Initially, the actual token cost is computed for both the current and previously evaluated budgets (line 2). Next, feasibility is assessed based on two criteria: the answer correctness and greedy token reduction (line 3). The search process is terminated if either condition fails.

Overhead of Budget Search. For the overhead, note that the budget search mainly serves to reveal token compression potential, motivating TALE’s design. For TALE-PT, the budget search is used once offline to generate training targets for post-training. Since this is a one-time pre-processing step, it does not incur additional costs during actual model deployment. For TALE-EP, we clarify that

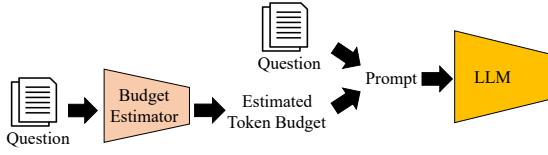


Figure 4: The workflow of TALE-EP. Given a question, TALE-EP first estimates the token budget using a budget estimator. It then crafts a token-budget-aware prompt by combining the question with the estimated budget. Finally, the prompt is input to the LLM to generate the answer as the final output. By default, we use the reasoning LLM itself with zero-shot estimation prompt as the budget estimator.

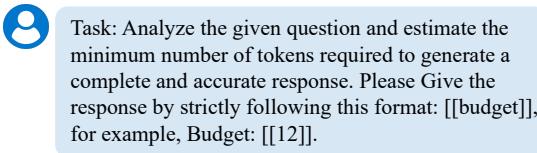


Figure 5: The prompt for zero-shot budget estimation.

the budget search is not needed at all. TALE-EP relies on a lightweight, zero-shot budget estimator that directly predicts a reasonable token budget without any iterative search, making it training-free and highly efficient at inference time. To quantify the budget search overhead for TALE-PT data generation, we measured the total time needed to run the search algorithm and produce the optimal budget dataset. On GSM8K (7473 samples), this process takes approximately 354 minutes on an A100 GPU, which we consider acceptable given that it is a one-time offline cost for training.

5 Methodology

5.1 Overview

Based on the above analysis, we designed our method TALE for token-budget-aware reasoning in LLMs. Two solutions, i.e., estimation&prompting (TALE-EP, see Figure 4) and post-training (TALE-PT, see Figure 6), are proposed.

5.2 Estimation and Prompting (TALE-EP)

Our observations on token elasticity (Section 4) indicate that only a well-chosen budget within a reasonable range can effectively minimize token costs while preserving LLM performance. The optimal budget, found using Algorithm 1 and Algorithm 2, lies within this range and achieves a satisfying trade-off between efficiency and performance. Building on this insight, we introduce a token budget aware reasoning method by zero-shot-based token budget estimation and prompting the reasoning LLM. TALE-EP leverages the reasoning capabilities of the LLM as an estimator. Fig-

ure 4 provides an overview of TALE-EP’s workflow. The goal of TALE-EP is to construct a token-budget-aware prompt that maintains performance comparable to vanilla CoT while reducing token costs. To achieve this balance, TALE-EP follows a two-phase approach: budget estimation and prompt construction. Given a question, TALE-EP first estimates a reasonable token budget that closely aligns with the optimal searched budget. By default, we use the reasoning LLM itself with a zero-shot estimation prompt as the budget estimator. Figure 5 demonstrates the budget estimation prompt, which will guide the model to evaluate the question as a whole.. Using this estimate, it then crafts a token-budget-aware prompt and feeds it into the LLM to generate the final answer. Figure 7c illustrates this process with a concrete example. The key intuition behind TALE-EP is inspired by human-like thinking. When solving a mathematical problem, a person may take time to compute the exact answer but can quickly estimate the effort required to solve it. For instance, when comparing a primary school arithmetic question to a college-level calculus problem, one may not immediately provide the solutions but can easily infer that the former takes only seconds while the latter requires significantly more time. Section A.2 evaluates the effectiveness of our budget estimation approach, demonstrating that the budgets estimated by advanced LLMs (e.g., GPT-4o-mini) are generally close to the optimal searched budget and deliver competitive performance.

5.3 TALE Post-Training (TALE-PT)

Another approach for obtaining an LLM with token-budget awareness is post-training it to incorporate this awareness into its inference process, enabling it to generate more token-efficient reasoning responses. Specifically, we post-train the LLM \mathcal{M}_θ to produce answers that adhere to the token budget. This process is divided into two key stages: target output generation and LLM post-training.

Target Output Generation. In the target output generation stage, we craft the target output y_i by prompting \mathcal{M}_θ with a Chain-of-Thought (CoT) prompt that incorporates our searched optimal token budget. The prompt is formatted as follows:

“Let’s think step by step and use less than β_i^* tokens:”

where β_i^* is the searched optimal budget for the given question x_i (see search process in Algo-

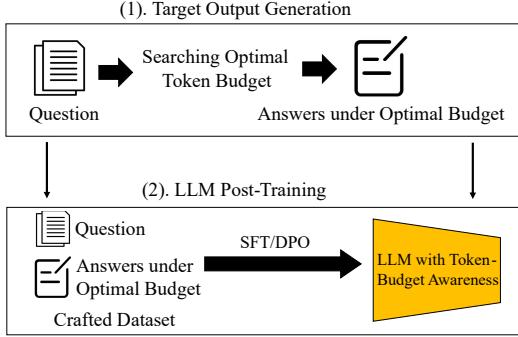


Figure 6: The workflow of TALE-PT. Given a set of questions, TALE-PT first generates target outputs in phase (1), searching the answers under optimal budget. In phase (2), TALE-PT uses the searched target outputs to craft a specialized dataset, then post-train the LLM (via SFT/DPO) to internalize token-budget awareness.

rithm 1 and Algorithm 2). Figure 1d illustrates an example. The resulting LLM output, constrained by the token budget specified in the prompt, is taken as the crafted target output y_i . This target output not only leads to the correct answer but also has minimal actual output token cost among our token elasticity-based search process, as described in Section 4. In the LLM post-training stage, we train the LLM \mathcal{M}_θ using the crafted target outputs from the first stage. We introduce two ways to conduct the token-budget awareness internalization during post-training, i.e., SFT-based and DPO-based method. Details of the hyper parameters are in Section A.3.

SFT-based Internalization. To inject token-budget awareness into \mathcal{M}_θ , we perform supervised fine-tuning with these target outputs. We post-train \mathcal{M}_θ to generate token-efficient outputs by minimizing the cross-entropy loss between the model’s predictions and the target outputs. Given an input x and a target output y from the first stage (which reflects token-budget awareness), the cross-entropy loss is defined as:

$$\mathcal{L}_{CE}(\theta) = -\frac{1}{N} \sum_{i=1}^N \sum_{t=1}^{T_i} \log \mathbb{P}(y_{i,t}|y_{i,<t}, x_i),$$

where T_i means the length of the target sequence y_i for the i -th training example, $y_{i,t}$ the target token at position t of y_i , $y_{i,<t}$ means the sequence of tokens preceding the current token $y_{i,t}$, representing the context up to time step t for the i -th sample. $\mathbb{P}(y_{i,t}|y_{i,<t}, x_i)$ represents the conditional probability predicted by the model \mathcal{M}_θ for the token $y_{i,t}$, given the input x_i and the preceding tokens $y_{i,<t}$. The loss is based on the next token prediction. The goal is to adjust the model parameters

θ such that it produces concise and accurate responses that adhere to the token budget constraint. This is achieved through gradient descent, forcing the model to internalize the compact reasoning patterns from the token-efficient target outputs.

DPO-based Internalization. Another way to incentivize \mathcal{M}_θ to learn the token-budget preference is applying the DPO algorithm (Rafailov et al., 2023) to post-train the model. DPO directly refines the policy through a classification objective, aligning the model’s behavior with the desired preferences. The goal of DPO here is to refine \mathcal{M}_θ so it can accurately solve a given problem x while adhering to an internalized token budget. We use the target outputs y_i from the searched optimal budget as positive samples, while outputs y'_i generated with the vanilla CoT prompt serve as negative samples. These positive-negative pairs are then used to create the pairwise preference data for DPO training. Given the crafted dataset $\mathcal{D} = \{(x_i, y_i, y'_i)\}_{i=1}^N$, the objective is to maximize the likelihood that the model ranks the positive samples higher than the negative ones. Formally, we aim to optimize the following objective:

$$\mathcal{L}_{DPO}(\theta) = -\frac{1}{N} \sum_{i=1}^N \log P_\theta(y_i \succ y'_i), \quad \text{where}$$

$$P_\theta(y_i \succ y'_i) = \frac{\exp(s(y_i, x_i))}{\exp(s(y_i, x_i)) + \exp(s(y'_i, x_i))}.$$

$P_\theta(y_i \succ y'_i)$ is the preference function. Here, $s(y_i, x_i)$ is defined as $\sum_{t=1}^{T_i} \log \mathbb{P}(y_{i,t}|y_{i,<t}, x_i)$, and it represents the log-probability of the model generating y_i for input x_i , which serves as the preference score assigned to y_i . This score measures how strongly the model favors that output. The objective ensures that the model prioritizes concise and token-efficient outputs while maintaining high-quality reasoning and correctness. During training, the LLM is encouraged to internalize the token budget constraint and adopt a more compact reasoning process guided by the target outputs generated in the first stage. This two-stage process effectively trains the LLM to produce concise yet accurate responses, striking a balance between reasoning quality and token efficiency during inference. More details are in Section A.3.

6 Evaluation

In this section, we provide the experiment results to evaluate the effectiveness of two versions of TALE,

Table 3: Comparison of TALE-EP (estimation and prompting) and other prompt engineering methods. “Directly Answering” means prompting LLM without any reasoning process. “Vanilla CoT” means the vanilla CoT prompting without budget. The model used in our evaluation is GPT-4o-mini (OpenAI, 2024a). Observe that TALE-EP achieves an average accuracy (ACC) of 80.22%, with an average output token cost of 138.53 and an average expense of 118.46. TALE-EP reduces output token costs by 67%, lowers expenses by 59%, and maintains competitive performance compared to the vanilla CoT approach. ACC \uparrow , Output Tokens \downarrow , Expense (10^{-5} \$ / sample) \downarrow .

Dataset	Directly Answering			Vanilla CoT			TALE-EP		
	ACC \uparrow	Output Tokens \downarrow	Expense \downarrow	ACC \uparrow	Output Tokens \downarrow	Expense \downarrow	ACC \uparrow	Output Tokens \downarrow	Expense \downarrow
GSM8K	28.29%	12.46	39.43	81.35%	318.10	541.09	84.46%	77.26	279.84
GSM8K-Zero	97.21%	18.85	91.69	99.50%	252.96	886.79	98.72%	22.67	276.12
MathBench-Arithmetic	59.67%	41.10	9.78	75.00%	313.51	78.58	73.67%	39.60	18.62
MathBench-Middle	33.33%	5.00	3.58	84.67%	553.93	68.22	79.33%	238.14	42.95
MathBench-High	51.33%	5.00	4.07	84.00%	653.24	82.44	80.00%	254.82	47.61
MathBench-College	44.00%	5.00	3.68	78.00%	675.78	81.56	70.00%	259.85	45.60
Average	52.31%	14.57	25.37	83.75%	461.25	289.78	81.03%	148.72	118.46

Table 4: The generalization of TALE-EP (estimation and prompting) across different LLMs. Yi-lightning (Wake et al., 2024), GPT-4o-mini (OpenAI, 2024a), GPT-4o (OpenAI, 2024b) and o3-mini (OpenAI, 2025) are taken into consideration. We conduct following evaluations on the MathBench-College dataset. ACC \uparrow , Output Tokens \downarrow , Expense (10^{-5} \$ / sample) \downarrow .

LLM	Directly Answering			Vanilla CoT			TALE-EP		
	ACC \uparrow	Output Tokens \downarrow	Expense \downarrow	ACC \uparrow	Output Tokens \downarrow	Expense \downarrow	ACC \uparrow	Output Tokens \downarrow	Expense \downarrow
Yi-lightning	66.67%	80.01	3.09	79.33%	998.10	21.55	76.67%	373.52	17.25
GPT-4o-mini	44.00%	5.00	3.68	78.00%	675.78	81.56	70.00%	259.85	45.60
GPT-4o	57.33%	5.00	61.34	84.00%	602.29	1359.42	80.00%	181.61	759.95
o3-mini	96.00%	601.51	336.69	97.33%	1163.55	638.46	96.66%	677.65	385.12

TALE-EP and TALE-PT. The comparisons of the two implementations are detailed in Section A.6.

6.1 Experiment Setup

Datasets. To evaluate the LLM performance, three most challenging mathematical datasets are taken into consideration: GSM8K (Cobbe et al., 2021), GSM8K-Zero (Chiang and Lee, 2024), and Math-Bench (Liu et al., 2024). GSM8K-Zero, derived from the GSM8K dataset, specifically targets the analysis of over-reasoning and redundancy in LLM-generated outputs. In short, GSM8K-Zero is designed so that the answers are embedded within the questions themselves. LLMs can easily generate correct responses without complicated additional reasoning or redundant calculations.

Models. We conduct experiments on five state-of-the-art LLMs (i.e., GPT-4o (OpenAI, 2024b), GPT-4o-mini (OpenAI, 2024a), Yi-lightning (Wake et al., 2024), o3-mini (OpenAI, 2025)), and Llama-3.1-8B-Instruct (Dubey et al., 2024).

Metrics. The target of TALE is to balance the LLM correctness performance and extra redundant token costs. Specifically, TALE seeks to minimize *Number of Output Tokens* while maintaining comparable *Accuracy (Acc)* simultaneously.

Accuracy (Acc). This metric is calculated as the following: $Accuracy = \frac{1}{N} \sum_{i=1}^N \mathbb{I}\{\mathcal{M}(\mathbf{x}_i) = y_i\}$, where $(\mathbf{x}_i, y_i) \in \mathcal{X}$. \mathbf{x}_i is the math question from

dataset \mathcal{X} and y_i the ground truth answer. $\mathcal{M}(\cdot)$ returns the answer for a given question. $\mathbb{I}\{\cdot\}$ represents an indicator function. This function evaluates whether the inside given condition holds. Specifically, it returns **1** if the condition is true and **0** if the condition is false. For a better evaluation, we format the LLM output by crafting an elaborate instruction detailed in Figure 8.

Number of Output Tokens. We evaluate the token costs by calculating the average output token consumption for each specific task. The output token costs are measured as follows: $Number of Output Tokens = \frac{1}{N} \sum_{i=1}^N \mathbb{T}(\mathcal{M}(\mathbf{x}_i))$, where \mathbf{x}_i represents the given question, and \mathbb{T} is a function that measures the number of tokens. Intuitively, the more output tokens, the higher the costs incurred by \mathcal{M} . To evaluate costs more precisely, we calculate the average expense per sample. The total token expense includes both input and output tokens used during the query process.

6.2 Effectiveness of TALE-EP

Table 3 compares TALE-EP with other prompt engineering methods across seven datasets, evaluating accuracy, output tokens, and expenses. Effective prompts should maximize accuracy while minimizing token usage and cost. Direct Answering is the most cost-efficient (14.57 tokens, 25.37 expense) but with low accuracy (52.31%). Vanilla

Table 5: Comparison of TALE-PT (post-training to internalize token-budget awareness) and other prompt engineering methods. Two different post-training methods, SFT and DPO, are taken into consideration.

LLM	Dataset	Directly Answering		Vanilla CoT		TALE-PT-SFT		TALE-PT-DPO	
		ACC ↑	Output Tokens ↓	ACC ↑	Output Tokens ↓	ACC ↑	Output Tokens ↓	ACC ↑	Output Tokens ↓
Llama-3.1-8B-Instruct	GSM8K	21.00%	38.54	77.56%	241.51	78.57%	139.63	74.11%	149.93
	GSM8K-Zero	70.32%	13.49	65.04%	251.08	78.43%	77.85	78.41%	113.41

CoT achieves the highest accuracy (83.75%) but at a high cost (461.25 tokens, 289.78 expense). TALE-EP balances performance and efficiency, achieving 81.03% accuracy while reducing token usage to 32% and expenses to 41% of Vanilla CoT. On GSM8K, it even surpasses Vanilla CoT with 84.46% accuracy. Note that expense is not directly proportional to output tokens because it also accounts for input and cached tokens. TALE-EP reduces token costs by 68.64% on average, offering a scalable, cost-effective solution for budget-constrained reasoning tasks. For resource-rich scenarios, we evaluate TALE-EP under a larger token budget as detailed in [Section A.8](#).

To further evaluate the generalization of TALE-EP across different LLMs. We conduct experiments across Yi-lightning, GPT-4o-mini, GPT-4o and o3-mini on MathBench-College. [Table 4](#) illustrates the results, showing TALE-EP’s ability to reduce output tokens and expenses while maintaining competitive accuracy significantly. TALE-EP achieves substantial token savings, reducing output tokens by 64.63% on average, compared to Vanilla CoT. Expense reductions are equally notable, with costs decreasing by 45.30% on average. Despite these cost savings, TALE-EP maintains strong accuracy, achieving 76.67% on Yi-lightning, 70.00% on GPT-4o-mini, and 80.00% on GPT-4o, comparable to Vanilla CoT. These results highlight TALE-EP’s effectiveness in balancing cost efficiency and reasoning performance across diverse LLM architectures. The observed accuracy drop is most significant for GPT-4o-mini. This could be attributed to its smaller number of parameters, which makes it more challenging to answer correctly within a limited response reasoning length. We also evaluate the applicability of TALE on more tasks, the results are illustrated in [Section A.5](#). The efficiency analysis of TALE-EP is in [Section A.7](#)

6.3 Effectiveness of TALE-PT

[Table 5](#) compares TALE-PT methods with Vanilla CoT and Direct Answering on GSM8K and GSM8K-Zero using Llama-3.1-8B-Instruct. For GSM8K, Direct Answering demonstrates the low-

est token usage (38.54) but at the cost of significantly reduced accuracy (21.00%). In contrast, Vanilla CoT achieves much higher accuracy (77.56%) but incurs a significant increase in token cost (241.51). Note that on GSM8K-Zero, the accuracy of Vanilla CoT drops below Direct Answering. This drop can be attributed to overthinking, as GSM8K-Zero is simpler, with answers often implied directly within the question. In such cases, a long reasoning process can introduce unnecessary complexity, leading to reduced accuracy. Among the TALE-PT methods, TALE-PT-SFT achieves the best accuracy (78.57%, 78.43%) with reduced tokens, while TALE-PT-DPO balances accuracy (74.11%, 78.41%) and token efficiency, cutting token consumption by over 50% on GSM8K-Zero compared to Vanilla CoT.

7 Conclusion

In this paper, we introduce TALE, a framework that reduces token redundancy in Chain-of-Thought (CoT) reasoning by incorporating token budget awareness. TALE dynamically adjusts the number of reasoning tokens based on the reasoning complexity of each problem, balancing token efficiency and answer correctness. Experiments show that TALE reduces output token usage and expense significantly with acceptable accuracy loss, outperforming Vanilla CoT in cost-effectiveness while generalizing well across various LLMs.

8 Limitations

The experiments of our proposed token-budget-aware reasoning framework currently focus on LLMs that process only text as input and output. While the results demonstrate significant improvements in efficiency and cost reduction, it does not account for models that have multimodal output content. Such as the models generate interleaved images and text as output. In future work, we will extend token-budget awareness to such LLMs with multimodal output by introducing modality-specific budget constraints and designing adaptive strategies to optimize token efficiency for different modality types, such as images and videos.

References

- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Aman Bhargava, Cameron Witkowski, Shi-Zhuo Looi, and Matt Thomson. 2023. What’s the magic word? a control theory of llm prompting. *arXiv preprint arXiv:2310.04444*.
- Bin Bi, Chenliang Li, Chen Wu, Ming Yan, Wei Wang, Songfang Huang, Fei Huang, and Luo Si. 2020. Palm: Pre-training an autoencoding&autoregressive language model for context-conditioned generation. *arXiv preprint arXiv:2004.07159*.
- Haolin Chen, Yihao Feng, Zuxin Liu, Weiran Yao, Akshara Prabhakar, Shelby Heinecke, Ricky Ho, Phil Mui, Silvio Savarese, Caiming Xiong, et al. 2024. Language models are hidden reasoners: Unlocking latent reasoning capabilities via self-rewarding. *arXiv preprint arXiv:2411.04282*.
- Cheng-Han Chiang and Hung-yi Lee. 2024. Overreasoning and redundant calculation of large language models. *arXiv preprint arXiv:2401.11467*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid llm: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Guohao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2024. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36.
- Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. 2023. AlphaZero-like tree-search can guide large language model decoding and training. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, et al. 2024a. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. *arXiv preprint arXiv:2404.05221*.
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024b. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.
- Namgyu Ho, Laura Schmid, and Se-Young Yun. 2022. Large language models are reasoning teachers. *arXiv preprint arXiv:2212.10071*.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Code-searchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.
- Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, et al. 2024a. Exploring concept depth: How large language models acquire knowledge and concept at different layers? *arXiv preprint arXiv:2404.07066*.
- Mingyu Jin, Qinkai Yu, Dong Shu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, and Mengnan Du. 2024b. The impact of reasoning step length on large language models. *arXiv preprint arXiv:2401.04925*.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.
- Chenliang Li, Bin Bi, Ming Yan, Wei Wang, and Songfang Huang. 2021. Addressing semantic drift in generative question answering with auxiliary extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 942–947.
- Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5315–5333.

- Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark. *arXiv preprint arXiv:2405.12209*.
- Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*.
- Sania Nayab, Giulio Rossolini, Giorgio Buttazzo, Nicola Maria Manes, and Fabrizio Giacomelli. 2024. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*.
- OpenAI. 2024a. [Gpt-4o mini: advancing cost-efficient intelligence](#). Technical report, OpenAI. Accessed: July 18, 2024.
- OpenAI. 2024b. [Hello gpt-4o](#). Technical report, OpenAI. Accessed: May 13, 2024.
- OpenAI. 2024c. [Learning to reason with llms](#). Technical report, OpenAI.
- OpenAI. 2025. [Openai o3-mini: Pushing the frontier of cost-effective reasoning](#). Technical report, OpenAI. Accessed: January 31, 2025.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.
- Hannah Rashkin, Eric Michael Smith, Margaret Li, and Y-Lan Boureau. 2019. Towards empathetic open-domain conversation models: A new benchmark and dataset. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5370–5381.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.
- Guangyan Sun, Mingyu Jin, Zhenting Wang, Cheng-Long Wang, Siqi Ma, Qifan Wang, Ying Nian Wu, Yongfeng Zhang, and Dongfang Liu. 2024. Visual agents as fast and slow thinkers. *arXiv preprint arXiv:2408.08862*.
- Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.
- Alan Wake, Albert Wang, Bei Chen, CX Lv, Chao Li, Chengen Huang, Chenglin Cai, Chujie Zheng, Daniel Cooper, Ethan Dai, et al. 2024. Yi-lightning technical report. *arXiv preprint arXiv:2412.01253*.
- Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. 2024a. Reasoning in token economies: Budget-aware evaluation of llm reasoning strategies. *arXiv preprint arXiv:2406.06461*.
- Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Zhenting Wang, Guofeng Cui, Kun Wan, and Wentian Zhao. 2025. Dump: Automated distribution-level curriculum learning for rl-based llm post-training. *arXiv preprint arXiv:2504.09710*.
- Zhenting Wang, Shuming Hu, Shiyu Zhao, Xiaowen Lin, Felix Juefei-Xu, Zhuowei Li, Ligong Han, Harihar Subramanyam, Li Chen, Jianfa Chen, et al. 2024b. Mllm-as-a-judge for image safety without human labeling. *arXiv preprint arXiv:2501.00192*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*, pages 1–22.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*.

Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. 2022. Seqzero: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. *arXiv preprint arXiv:2205.07381*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024b. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2024. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems*, 36.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

A Appendix

A.1 Definition of Ideal Budget Range

Ideal Budget Range. Based on the observation of token elasticity, a token cost bottom range exists during searching for the optimal budget. In this range, the token costs approach the token cost lowest bound. Before or after the range, the token cost will increase. We define such a bottom range as “ideal budget range”. It’s worth noting that the budget continuously degrades during the search. Only the token cost rebounds. That’s why we refer to this observation as token elasticity. To summarize, ideal budget range is a range that minimizes actual token consumption. Let $\beta = \{\beta_1, \beta_2, \dots, \beta_N\}$ denote all possible budgets that can maintain answer correctness. A rolling window $W \in \beta$ is applied iteratively over β . Let k represent the range size, which is adaptively determined during our evaluation as $\frac{N}{3}$, where N is the total number of possible budgets. A budget range is defined as:

$$W_k(i) = \{\beta_j \mid i \leq j \leq i + k - 1\}, \\ 1 \leq i \leq |\beta| - k + 1$$

Q: Tom's ship can travel at 50 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 30 mph. How long does it take him to get back?

It takes Tom 3 hours to get back.

(a) Direct answering (10 output tokens).

Q: Tom ... How long does it take him to get back? Let's think step by step:

Let's break the problem down step by step:
 ### Step 1: Calculate the distance Tom travels. Tom sails from point 1 to point 4 PM, so he is traveling from 1:00 PM to 4:00 PM, which is a 3-hour journey. Since Tom's ship travels at a speed of 50 miles per hour, we can calculate the distance he covers during this time: $\text{Distance} = \text{Speed} \times \text{Time} = 50 \times 3 = 150$.

Step 2: Calculate the time it takes to travel back. Tom travels back at a rate of 30 miles per hour, and the distance is the same (150 miles). We can calculate the time it takes him to return by dividing the distance by his return speed: $\text{Time to return} = \frac{\text{Distance}}{\text{Speed}} = \frac{150}{30} = 5$.

Final Answer: It takes Tom **5 hours** to get back.

(b) Vanilla CoT (271 output tokens).

Q: Tom's ship can travel at 50 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 30 mph. How long does it take him to get back? Task: Analyze the given question and estimate the minimum number of tokens required to generate a complete and accurate response. Please give the response by strictly following this format: [[budget]], for example, Budget: [[12]].

Budget: [[26]]

Budget Estimation

Please answer the above question.
 Let's think step by step and use less than 26 tokens:
 1. Time out: 3 hours (1 to 4 PM).
 2. Distance = speed × time = $50 \text{ mph} \times 3 \text{ hours} = 150 \text{ miles}$. 3. Time back = distance ÷ speed = $150 \text{ miles} \div 30 \text{ mph} = 5 \text{ hours}$.
 Answer: 5 hours.

Token-budget-aware CoT

(c) TALE (68 output tokens).

Figure 7: An intuitive example to illustrate the workflow of TALE-EP on GPT-4o-mini (OpenAI, 2024a).

The ideal budget range W^* is defined as:

$$W_k^* = \arg \min_i \left(\sum_{\beta_j \in W_k(i)} \mathbb{T}(\beta_j) \right), \quad (1)$$

where \mathbb{T} denote the actual token consumption for a given budget $\beta \in \beta$. We aim to estimate a budget located in the ideal budget ranges without any search process. In that case, TALE obtains the ideal budget within acceptable sacrifice.

A.2 Effectiveness of Budget Estimation.

In this RQ, we evaluate the effectiveness of the budget estimation performance. An ideal estimated budget should be located around the optimal

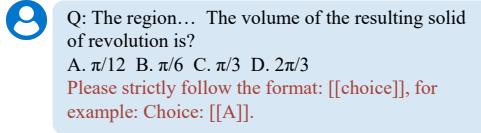


Figure 8: The instruction prompt used to format the LLM output on multiple-choice questions.

searched budget and in the bottom area of Figure 2. We further define such an area as the ideal budget range and give the formalized definition in Section A.1. A good budget should be located in the ideal budget range. Two metrics are taken into consideration: *in-range accuracy* and *out-of-range distance*. In-range accuracy determines whether the predicted budget $\hat{\beta}$ falls within the ideal budget range W_k^* . Mathematically, it can be expressed as:

$$\mathbb{I}\{\hat{\beta} \in W_k^*\} = \begin{cases} 1, & \text{if } \hat{\beta} \in W_k^*, \\ 0, & \text{otherwise.} \end{cases}$$

Out-of-range distance quantifies the distance between $\hat{\beta}$ and W_k^* if the predicted budget $\hat{\beta}$ falls outside the ideal budget range W_k^* . Let $dist(\hat{\beta}, W_k^*)$ represent the distance, defined as:

$$dist(\hat{\beta}, W_k^*) = \begin{cases} 0, & \text{if } \hat{\beta} \in W_k^*, \\ \min_{\hat{\beta} \in W_k^*} |\hat{\beta} - \beta|, & \text{if } \hat{\beta} \notin W_k^*. \end{cases}$$

Intuitively, a higher in-range accuracy and a lower out-of-range distance indicate a better estimated budget. During our evaluation, the in-range accuracy is 60.61%, and the out-of-range distance is 109.64. It indicates that more than two-thirds of estimated budgets are located in the ideal range. For those out-of-range samples, they have an offset of 109.64 tokens on average. Figure 9 illustrates the successful and failed estimated cases intuitively. The prompt we use for budget estimation is as follows:

“Task: Analyze the given question and estimate the minimum number of tokens required for reasoning.”

This prompt encourages the model to evaluate the question as a whole, including reasoning depth, structure, completeness, and surface-level difficulty.

A.3 Details of TALE’s Implementation

In this section, we introduce the hyper-parameters used for TALE-EP and TALE-PT.

TALE-EP. TALE-EP uses a zero-shot mechanism to estimate the token budget and then prompts the LLM. The instruction prompts used during this

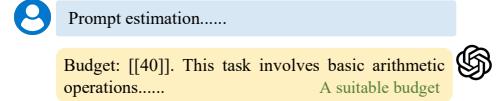


Figure 9: An intuitive example for successful and failed cases of prompt budget estimation in TALE-EP.

Table 6: Comparison of TALE-EP and TALE-PT.

Metrics	TALE-EP	TALE-PT	
		SFT	DPO
ACC	71.82	78.57	74.11
Output Tokens	112.21	139.63	149.93

process are shown in Figure 7. To ensure output consistency, we set the temperature to 0.1 and limit the model to a single reasoning path. Additionally, the random seed is fixed at 1024.

TALE-PT. TALE-PT includes two implementations: SFT and DPO. For parameter efficiency, both implementations adopt LoRA (Hu et al., 2021) for post-training, with rank set to 8 and lora alpha set to 32. For TALE-PT-SFT, we train for 3 epochs with a batch size of 16, a learning rate of 1e-4, and a weight decay of 0.01. For TALE-PT-DPO, we train for 2 epochs with a batch size of 16, a learning rate of 3e-5, and a weight decay of 0.001.

A.4 Comparison of TALE-EP and TALE-PT.

In this section, we compare the performance of TALE-EP and TALE-PT (including SFT and DPO). Specifically, we utilize Llama-3.1-8B-Instruct as both the budget estimator for TALE-EP and the base model for TALE-PT, evaluated on GSM8K. Table 6 illustrates the evidence. As the zero-shot-based estimator tends to predict relatively low budgets, TALE-EP achieves lower token usage but at the cost of slightly reduced accuracy. In contrast, both variants of TALE-PT achieve higher accuracy with more tokens, as their training data is constructed using optimal budget search, which enforces answer correctness as a strict constraint even with a higher token costs. This highlights a trade-off between strict correctness preservation (in TALE-PT) and token efficiency (in TALE-EP).

A.5 Applicability of TALE on More Tasks.

To further evaluate the applicability, we deploy

Table 7: Generalization of TALE on more tasks. Three popular LLM generative tasks, Code Summarization (Husain et al., 2019)(CS), Empathetic Response Generation (Rashkin et al., 2019)(ERG), Code Generation (Austin et al., 2021)(CG), are taken into consideration. BLEU is taken as the metric to evaluate the performance. BLEU↑. Output Tokens↑.

Tasks	TALE-EP		Vanilla CoT	
	BLEU	Output Tokens	BLEU	Output Tokens
CS	0.07	44.39	0.2	134.05
ERG	0.005	60.34	0.006	175.37
CG	0.24	171.08	0.267	461.77

TALE-EP on three additional open-ended generative tasks. As these tasks involve open-ended text generation, we adopt the BLEU metric (Papineni et al., 2002) to quantify the similarity between generated outputs and reference texts. The results in Table 4 demonstrate that TALE-EP achieves comparable or even better BLEU scores than Vanilla CoT while using only around 40% of the output tokens, validating its effectiveness and applicability in broader generative tasks.

A.6 Formalizing the Budget Search.

For a given input x , we define the search space as:

$$\mathcal{B} = \{\beta \in \mathbb{Z}^+ \mid 0 < \beta \leq T_{\text{vanilla}}(x)\}$$

where $T_{\text{vanilla}}(x)$ is the number of tokens generated by Vanilla CoT. A candidate budget $\beta \in \mathcal{B}$ is considered feasible if:

$$LLM(x, \beta) = y \text{ and } T(x, \beta) < T(x, \beta_0)$$

where y is the ground-truth answer, $T(x, \beta)$ is the actual number of output tokens when answering x under budget β , β_0 is the previously searched (larger) feasible budget. Our goal is to find:

$$\beta^* = \arg \min_{\beta \in \mathcal{B}} T(x, \beta), \text{ subject to } LLM(x, \beta) = y$$

To efficiently find β^* , we employ a binary search procedure guided by the feasibility function above, as detailed in Algorithm 1 and Algorithm 2.

A.7 Efficiency of TALE-EP.

Since TALE-EP requires one additional query, we further evaluate its end-to-end latency in this section. Specifically, we query the Llama-3.1-8B-Instruct model over the GSM8K-Zero dataset and measure both accuracy and average time cost. The budget estimation query of TALE-EP is also considered. Although TALE-EP requires one additional

Table 8: The empirical evidence for “implicit monotonicity assumption”. $\bar{\beta}$ is the budget upper bound, which is the token cost of vanilla CoT. The budget row displays scaled budgets ranging from 2^{-2} to $2^2 \cdot \beta^*$.

Budget(* β^*)	2^{-5}	2^{-4}	2^{-2}	2^0
ACC	69.23	75.82	75.82	76.92
Output Tokens	222.69	222.42	244.61	653.53

query compared to Vanilla CoT, it is significantly more efficient, taking only 2.3 seconds per sample, while Vanilla CoT takes 10.2 seconds. This is because the primary factor influencing inference time is the number of output tokens, which TALE-EP effectively reduces.

A.8 Effectiveness of Larger Token Budget.

In scenarios with ample computational resources, the token budget could be larger for better performance. We simulate such a scenario by scaling the estimated budget by a factor α ($\alpha * \text{budget}$). As shown in the table below, increasing α from 1 to 2 leads to higher accuracy (from 67.33% to 72.66%) at the cost of more tokens (from 210.97 to 279.78), demonstrating that TALE-EP can flexibly adapt to different resource scenarios.

A.9 Empirical Evidence for the “Implicit Monotonicity Assumption”.

In this section, we give empirical evidence to support the “implicit monotonicity assumption” for our search algorithm. Specifically, for a set of questions, we vary the budget across a range of values (e.g., $2^{-5}, 2^{-4}, 2^{-2}, 2^0$ times budget upper bound, which is the token cost of vanilla CoT), and record the corresponding accuracy and average output tokens at each point. Table 8 illustrates the empirical evidence. Observe that it roughly follows from the monotonicity property. The results demonstrate a consistent trend: accuracy generally increases or plateaus with increasing budgets, confirming a soft monotonicity in most cases.