Algorithm 1 Budget Search

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

Output: searched budget β

1: **function** SEARCH(`isFeasible`,$\mathcal{M}, \boldsymbol{x}, y$)
2: $right \leftarrow$ the actual token costs of $\mathcal{M}$ with vanilla CoT prompt on $\boldsymbol{x}$
3: $\beta \leftarrow \lfloor (0 + right)/2 \rfloor$
4: $\beta_0 \leftarrow right$
5: **while** `True` **do**
6: **if** `isFeasible`$(\mathcal{M}, \boldsymbol{x}, y, \beta_0, \beta)$ **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** β

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

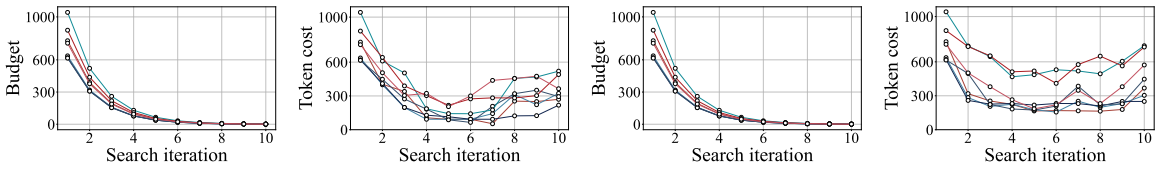
Vanilla Method for Optimal Budget Search. An intuitive method is finding the minimal needed tokens as the budget, ensuring that the LLM can still produce correct and accurate responses within this constraint. The goal of the search algorithm is not to directly determine task difficulty, but to identify 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 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.

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($*\beta^*$)	2^{-2}	2^{-1}	1	2^1	2^2
Prediction	False	False	True	True	True

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 $\boldsymbol{x}$ 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.

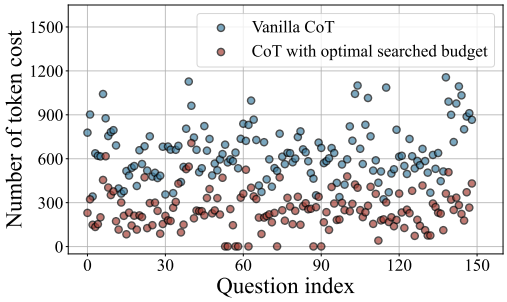


Figure 3: The effects of optimal searched budget. CoT, with our optimal searched budget, reduces the token costs significantly without influencing the accuracy. We conduct it on MathBench-College (Liu et al., 2024).

Algorithm 2 Greedy Feasibility Function

Input: a large language model $\mathcal{M}$, a question $\boldsymbol{x}$ 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`$(\mathcal{M}, \boldsymbol{x}, y, \beta_0, \beta)$
2: $t, t_0 \leftarrow$ gets the actual token costs under budgets of β and β_0
3: **if** $\mathcal{M}(\boldsymbol{x}, \beta) == y$ **and** $t < t_0$ **then**
4: **return** `True`
5: **return** `False`

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 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

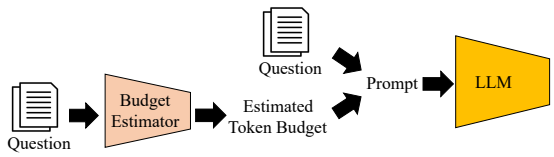


Figure 4: The workflow of TALE-EP. Given a question, TALE-EP first estimates the token budget using a budget estimator. It then crafts a token-budget-aware prompt by combining the question with the estimated budget. Finally, the prompt is input to the LLM to generate the answer as the final output. By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

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]].

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. Figure 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 $\mathcal{M}_\theta$ 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 $\mathcal{M}_\theta$ 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 $\boldsymbol{x}_i$ (see search process in Algo-