ADAPTIVE TEST-TIME COMPUTE ALLOCATION FOR REASONING LLMS VIA CONSTRAINED POLICY OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Spending more compute at inference time can improve large language model (LLM) accuracy, but not every query benefits equally. Under a limited budget, which queries should receive more compute, and which can be answered with less? We formulate this decision as a constrained optimization problem: assign a compute budget to each query to maximize expected accuracy, subject to a limit on average compute cost. To solve this problem, we propose a two-stage SOLVE-THEN-LEARN (STL) framework. In the *solve* stage, we use Lagrangian relaxation to construct a budget-allocation oracle with provable optimality guarantees on a calibration set. In the *learn* stage, we use the oracle allocations as supervision to train a lightweight classifier that predicts a compute budget for each new query before generation. For LLM reasoning, STL assigns different token caps to different questions to improve overall accuracy under a shared average token budget, while keeping the underlying LLM unchanged. Experiments across two benchmarks and three LLMs show that STL outperforms the strongest evaluated baseline in 17 of 18 settings, with an average accuracy gain of 3.90 percentage points and a maximum gain of 8.46 points under the same budget constraint.

1 INTRODUCTION

Large language model (LLM) performance has improved with more pre-training compute, larger models, and more training data (Kaplan et al., 2020; Hoffmann et al., 2022). Accuracy can also improve by spending more compute at *inference time*, through chain-of-thought reasoning (Wei et al., 2022), self-consistency voting (Wang et al., 2023), tree search (Yao et al., 2023), or process-reward verification (Lightman et al., 2024). This approach, known as *test-time scaling*, is used by frontier reasoning systems (OpenAI, 2024; DeepSeek-AI, 2025).

These gains come at an inference cost. Best-of-N sampling and majority voting require multiple model responses, while longer reasoning traces require more token generation. Both increase computation and can raise memory use and latency. In deployment, these costs mean higher API bills, lower throughput, and fewer queries served under a fixed resource budget (Kwon et al., 2023; Brown et al., 2024; Snell et al., 2025). As reasoning models generate longer chains of thought (OpenAI, 2024), deciding where to spend inference compute becomes increasingly important.

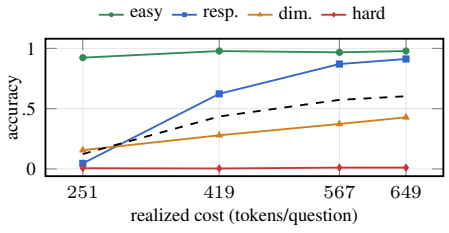

Figure 1: Token-cap accuracy curves (gpt-5.4-mini; 200 OlympiadBench calibration questions). Easy, responsive, diminishing-return, and hard groups contain 17/101/26/56 questions. Dashed: pool mean.

A common approach gives every query the same inference budget b (Wang et al., 2023; Brown et al., 2024). Depending on the method, b may be the number of sampled responses or the token limit for one generation. We study the latter: each query receives one generation with a chosen token cap. A uniform cap is simple, but queries benefit differently from extra compute.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Figure 1 illustrates this on gpt-5.4-mini and OlympiadBench (He et al., 2024). Easy queries are often solved at the smallest cap, while some hard queries remain unsolved even at the largest. Extra compute is most useful for queries whose accuracy improves substantially as the cap increases. Thus, difficulty alone is not enough: we need to estimate *how much a query benefits from extra compute*, and compare that benefit with its cost and the gains available on other queries.

At deployment, we face a practical question: how can we adaptively assign compute budgets to queries to maximize average expected accuracy under a shared budget constraint?

Mathematically, this question can be formulated as a *constrained optimization problem*:

$$\max_{\pi} \ \mathbb{E}_{x\sim\mathcal{D}}\big[\mathrm{Acc}(x,\pi(x))\big] \qquad \text{s.t.} \quad \mathbb{E}_{x\sim\mathcal{D}}\big[\mathrm{C}(\pi(x))\big] \le B. \tag{1}$$

Here, x denotes a query drawn from $\mathcal{D}$, and $\pi(x)$ is the compute budget assigned to it. $\mathrm{Acc}(x,b)$ is the probability of answering x correctly with budget b, and $\mathrm{C}(b)$ is the corresponding compute cost. The goal is to find an allocation policy π that maximizes average accuracy across queries while keeping their average compute cost at most B.

Under a shared average budget, allocating more compute to one query changes the resources available to others. We study this coupling through the constrained objective in (1) and use its Lagrangian solution to construct supervision for an allocation policy. Section 2 positions this approach relative to existing allocation and reasoning methods.

We address this problem with SOLVE-THEN-LEARN. In the **solve stage**, we use Lagrangian relaxation to obtain oracle budget allocations on a calibration set, with provable optimality guarantees. In the **learn stage**, we train a lightweight classifier to predict these allocations for new queries. At deployment, the classifier selects a token cap for each query before generation, and a budget-enforcement rule keeps the total completion-token use within the batch budget.

Our contributions are as follows.

- **Shared-budget optimization and supervision.** We study accuracy maximization under a shared average compute constraint. A Lagrangian solver coordinates allocations through a common compute price and converts calibration accuracy–cost tables into budget-specific action labels.
- **A deployable Solve-then-Learn policy.** A lightweight classifier learns these labels and selects a cap before generation, without updating the reasoning LLM. A separate execution rule keeps each batch within its token budget.
- **Allocation and learning guarantees.** We bound the accuracy gap between the returned allocation and an optimal allocation under the same budget constraint, without assuming diminishing returns in accuracy. We also bound how classification errors affect the learned policy's accuracy.
- **Budget-controlled evaluation.** We compare with Fixed-b, Random, and Heuristic on two datasets and three LLMs. STL outperforms the strongest of these baselines in $17/18$ primary settings, averaging 3.90 percentage points. Relative to Random, the mean and maximum gains are 4.05 and 8.46 points.

2 RELATED WORK

Our focus is *maximizing expected answer accuracy under a shared average compute constraint*. Solve-then-Learn connects this constrained objective to policy supervision: a common Lagrange multiplier coordinates calibration-set allocations, whose budget-specific action labels train a policy for new questions.

Test-time scaling and reasoning-length control. Chain-of-thought, self-consistency, search, and verification improve reasoning by spending additional inference compute (Wei et al., 2022; Wang et al., 2023; Yao et al., 2023; Lightman et al., 2024). Scaling studies show that the value of this compute depends on the query and inference strategy (Brown et al., 2024; Snell et al., 2025; Wu et al., 2025). SelfBudgeter learns to predict and follow token budgets (Li et al., 2025); L1, TALE, and s1 control reasoning length through training, prompting, or decoding (Aggarwal and Welleck, 2025; Han et al., 2025; Muennighoff et al., 2025). These methods provide ways to change the compute used on a query. Our optimization couples decisions across queries: assigning more compute to one query

reduces what remains for others. We select allocations by their contribution to aggregate accuracy under the shared constraint, with the reasoning model held fixed.

Input-adaptive allocation and stopping. Damani et al. (2025) predict marginal rewards to allocate compute under a shared average budget. PETS allocates samples to match the infinite-budget majority vote (Liu et al., 2026), while UAB uses response uncertainty to optimize a coverage surrogate (Nguyen et al., 2026). Adaptive-consistency and early-stopping methods allocate further compute from accumulated answer evidence (Aggarwal et al., 2023; Li et al., 2024; Wan et al., 2025). STL uses Lagrangian optimization of calibration accuracy–cost tables to construct budget-specific cap labels, then learns to predict these labels from question features. This connects the shared-budget accuracy objective to policy supervision, accommodates finite cap sets without diminishing-return assumptions, and separates allocation and imitation error. Deployment requires one capped generation per question without preliminary sampling.

Adaptive computation and routing. Adaptive computation time, early-exit networks, and dynamic networks vary internal computation across inputs (Graves, 2016; Teerapittayanon et al., 2016; Huang et al., 2018; Han et al., 2022); mixture-of-experts models select parameter subsets (Shazeer et al., 2017; Fedus et al., 2022). LLM routers and cascades choose among models with different capabilities and costs (Chen et al., 2023; Ong et al., 2024; Ding et al., 2024; Panda et al., 2025). STL shares the use of an input-dependent resource policy, with generation caps for one fixed backbone as its experimental action space. Its contribution centers on deriving policy targets from the shared constrained accuracy objective: Lagrangian allocation determines which actions to imitate, supervised learning transfers those decisions to new queries, and a separate execution rule enforces the realized batch budget.

3 Problem Formulation: Budget-Constrained Compute Allocation

Setup. Let $\mathcal{X}$ be the input space of queries and $\mathcal{D}$ the unknown distribution over $\mathcal{X}$. Queries may vary substantially in difficulty, e.g. from elementary to graduate-level math. A reasoning LLM serves these queries using a finite set of K candidate budgets, $\mathcal{B} = \{b_1, \ldots, b_K\}$. For example, b_k may be a cap on the tokens a single chain-of-thought may use—the instantiation of this paper—or the number of candidate responses in best-of-b_k decoding or self-consistency, or a search depth. Each mode incurs a real computational cost $\mathrm{C}(b_k)$, and without loss of generality the modes are ordered by cost, $\mathrm{C}(b_1) \leq \cdots \leq \mathrm{C}(b_K)$. For clarity, the formulation below writes cost as $\mathrm{C}(b)$; the same finite-table solver and gap bound apply to question-dependent costs $\mathrm{C}(x, b)$ by using each row's costs. In the token-cap experiments, these are estimated mean completion-token counts, not the caps themselves. A cap b bounds a generation's realized cost $c(x, b) \leq b$. Calibration-table feasibility does not guarantee feasibility for new responses; the executor in Section 6 enforces the latter separately.

For each input x and mode b, let $\mathrm{Acc}(x, b) \in [0, 1]$ be the expected task utility over sampled model responses. For answer correctness, it is the probability of a correct answer at cap b. This quantity must be estimated from generations (§7.1). A *budget allocation policy* is a measurable map $\pi : \mathcal{X} \to \mathcal{B}$ that assigns a mode to each input. For an allowed average cost $B \in (0, \infty)$, we seek the policy with the highest expected accuracy:

$$V(B) \;:=\; \max_{\pi \in \Pi} \; \mathbb{E}_{x \sim \mathcal{D}}\big[\mathrm{Acc}(x, \pi(x))\big] \qquad \text{s.t.} \quad \mathbb{E}_{x \sim \mathcal{D}}\big[\mathrm{C}(\pi(x))\big] \leq B, \tag{2}$$

where Π is the class of all measurable maps from $\mathcal{X}$ to $\mathcal{B}$ and $V(B)$ denotes the optimal value.

Challenges. Problem (2) is difficult to solve for three main reasons. *(i) Unknown accuracy*: $\mathrm{Acc}(x, b)$ must be estimated by running the model. *(ii) Unseen queries*: we need a policy that assigns budgets to new inputs, not just the calibration set. *(iii) Shared budget*: maximizing each query's accuracy separately can overspend, so gains must be compared across queries.

4 SOLVE STAGE: LAGRANGIAN OPTIMIZATION

Figure 2 illustrates our SOLVE-THEN-LEARN framework. In the solve stage, we use Lagrangian relaxation to obtain oracle budget allocations under a shared average compute constraint. In the learn stage, a lightweight classifier learns to predict these allocations from question features. At deployment, the classifier selects a token cap before generation, and a budget check keeps the batch within its total token budget. Sections 4 and 5 describe the solve and learn stages, respectively.

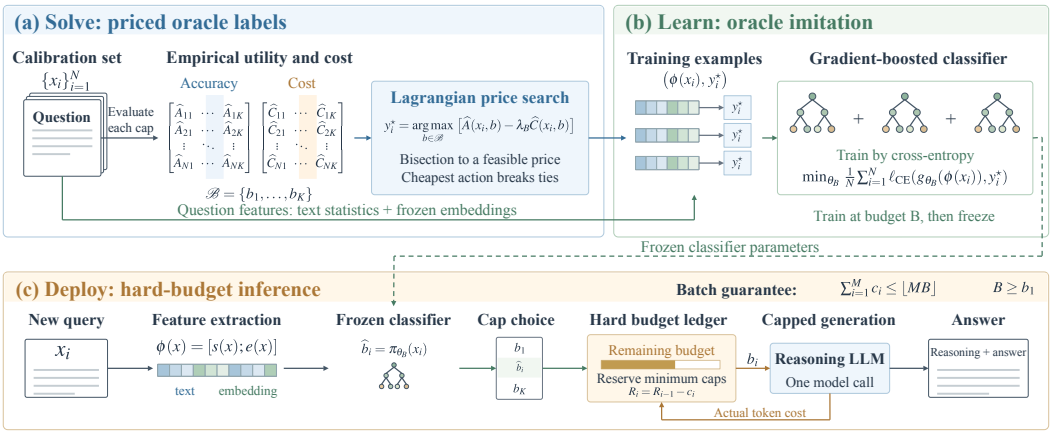

Figure 2: SOLVE-THEN-LEARN. Solve: optimize calibration-set allocations to obtain cap labels. Learn: train a classifier from question features to those labels. Deploy: predict a cap, lower it if required by the remaining budget, and generate one answer. Each target budget has its own classifier.

We begin with an offline dataset of N questions. For each question, we evaluate the model at each candidate token cap. These evaluations provide the input to the allocation problem. Replacing the expectation in (2) with an average over these questions gives

$$\hat{V}(B) \;:=\; \max_{\pi} \frac{1}{N}\sum_{i=1}^{N} \mathrm{Acc}\big(x_i, \pi(x_i)\big) \qquad \text{s.t.} \quad \frac{1}{N}\sum_{i=1}^{N} \mathrm{C}\big(\pi(x_i)\big) \le B. \tag{3}$$

We assume $B \ge \mathrm{C}(b_1)$, so a feasible allocation exists. Introducing a Lagrange multiplier $\lambda \ge 0$ for the budget constraint gives

$$\mathcal{L}(\pi, \lambda) \;=\; \frac{1}{N}\sum_{i=1}^{N} \big[\mathrm{Acc}(x_i, \pi(x_i)) - \lambda \mathrm{C}(\pi(x_i))\big] \;+\; \lambda\, B. \tag{4}$$

For fixed λ, the terms in (4) can be maximized separately. Each question is assigned the budget with the largest accuracy minus cost penalty:

$$b^\star(x; \lambda) \;:=\; \arg\max_{b \in \mathcal{B}} \big[\mathrm{Acc}(x, b) - \lambda\, \mathrm{C}(b)\big]. \tag{5}$$

For each question, we compare the scores of all candidate budgets in $\mathcal{B}$. Substituting these choices into the Lagrangian leaves a one-dimensional minimization over λ:

$$\min_{\lambda \ge 0} G(\lambda), \qquad G(\lambda) := \max_{\pi} \mathcal{L}(\pi, \lambda) = \frac{1}{N}\sum_{i=1}^{N} \max_{b \in \mathcal{B}} \big[\mathrm{Acc}(x_i, b) - \lambda \mathrm{C}(b)\big] + \lambda B.$$

For target budget B, let λ_B denote the multiplier obtained by binary search, or zero if the allocation at $\lambda = 0$ already satisfies the budget constraint. To solve for λ, let $\bar{C}(\lambda) := N^{-1} \sum_i \mathrm{C}(b^\star(x_i; \lambda))$ be the average allocation cost. This cost is non-increasing in λ (Appendix A.1), so we use binary search. If $\bar{C}(0) \le B$, set $\lambda_B = 0$. Otherwise, start with $\lambda_{\mathrm{lo}} = 0$ and increase λ_{hi} until $\bar{C}(\lambda_{\mathrm{hi}}) \le B$. At each midpoint, recompute the budgets using (5). If their average cost exceeds B, move the lower endpoint to the midpoint; otherwise, move the upper endpoint. Once the interval width is at most η, return

$\lambda_B = \lambda_{\text{hi}}$. Reducing an initial interval of width L to tolerance η costs $O(NK \log K[1 + \log(L/\eta)])$, including candidate ordering (Appendix A.2).

Finally, set $y_i^\star = b^\star(x_i; \lambda_B)$. These allocations satisfy the calibration budget and serve as the oracle labels for the classifier in Section 5. The following theorem bounds the difference in average accuracy between the returned allocation and an optimal allocation under the same budget constraint.

Theorem 1 (Optimality of the returned allocation)**.** *Let $\lambda_B \geq 0$ and $y_i^\star = b^\star(x_i; \lambda_B)$ satisfy $\bar{C}(\lambda_B) \leq B$, and write $\bar{A} := N^{-1} \sum_i \text{Acc}(x_i, y_i^\star)$. Then*

$$0 \leq \hat{V}(B) - \bar{A} \leq G(\lambda_B) - \bar{A} = \lambda_B \big[B - \bar{C}(\lambda_B)\big] =: \delta_B.$$

If $\bar{C}(\lambda_B) = B$ or $\lambda_B = 0$, the returned allocation is optimal for (3).

Here, δ_B denotes this accuracy-gap bound. Appendix A.3 provides the proof of Theorem 1.

5 Learn Stage: Amortized Policy Learning

The learn stage trains a classifier to predict the oracle label $y_i^\star := b^\star(x_i; \lambda_B)$ from the question x_i (Figure 2). It replaces calibration-table lookup with a prediction for unseen queries. Each backbone and target budget has its own fitted policy π_{θ_B}: changing the price generally changes the labels and requires refitting. The classifier should be inexpensive relative to LLM generation.

Let $\phi(x) \in \mathbb{R}^d$ be features computed from the question before generation. They should be inexpensive and useful for predicting its cap. Possible features include input length, vocabulary diversity, the number of sub-questions, and embeddings from a frozen encoder. Our implementation uses text statistics and frozen question embeddings; it uses no generated answer or confidence score.

Each oracle label is one of K budgets, so learning the allocation is a K-class prediction task. A scoring function $g_{\theta_B} : \mathbb{R}^d \to \mathbb{R}^K$ defines the policy by selecting the highest-scoring cap:

$$\pi_{\theta_B}(x) = b_{\arg\max_k \; g_{\theta_B}(\phi(x))_k}. \tag{6}$$

The classifier can be any standard architecture, including logistic regression, gradient-boosted trees, or a small MLP. The parameters θ_B are learned by minimizing the cross-entropy loss with respect to the oracle labels:

$$\min_{\theta_B} \; \frac{1}{N} \sum_{i=1}^{N} \ell_{\text{CE}}\big(g_{\theta_B}(\phi(x_i)),\; y_i^\star\big). \tag{7}$$

The experiments use unweighted cross-entropy and a histogram gradient-boosted classifier.

The following theorem bounds the accuracy lost when the classifier predicts oracle allocations.

Theorem 2 (Accuracy guarantee of the learned policy)**.** *Fix a reasoning LLM and a query distribution $\mathcal{D}$. Let $\pi_B^\star(x) = b^\star(x; \lambda_B)$ be the oracle allocation rule and $\hat{\pi}_B$ the learned policy. Define their average accuracy as $A(\pi) := \mathbb{E}_{x \sim \mathcal{D}}[\text{Acc}(x, \pi(x))]$, and let $\varepsilon_B := \mathbb{P}_{x \sim \mathcal{D}}(\hat{\pi}_B(x) \neq \pi_B^\star(x))$ be the oracle-label prediction error. If $\text{Acc}(x, b) \in [0, 1]$, then*

$$A(\hat{\pi}_B) \geq A(\pi_B^\star) - \varepsilon_B. \tag{8}$$

In particular, when $\mathcal{D}$ is the empirical distribution over the questions in Theorem 1,

$$A(\hat{\pi}_B) \geq \hat{V}(B) - \delta_B - \varepsilon_B. \tag{9}$$

Learning the oracle allocation rule reduces average accuracy by at most ε_B. Combined with Theorem 1 on the same question set, the accuracy gap is bounded by the optimization gap plus the label-prediction error. Appendix A.4 provides the proof.

6 Deployment

At deployment, compute question features and request a cap $\hat{b}_i = \pi_{\theta_B}(x_i)$ for each of M queries. Cost is actual completion tokens. Starting with $H = \lfloor MB \rfloor$ tokens, process queries in a fixed order. If R_i tokens remain before query i, reserve $(M - i)b_1$ for later queries and execute

$$b_i = \max\{b \in \mathcal{B} : b \leq \hat{b}_i,\; b \leq R_i - (M - i)b_1\}.$$

Only after this decision do we generate an answer and observe its cost $c_i \leq b_i$, updating $R_{i+1} = R_i - c_i$. Unused tokens remain available for later questions. With $B \geq b_1$, the minimum reservation guarantees $\sum_i c_i \leq H$ (Appendix A.5). The same causal rule executes STL, Random, and Heuristic requests; Fixed-b is already feasible because its uniform cap is at most B. With actual completion tokens as cost, this procedure generates answers sequentially and updates the remaining budget after each answer. To support parallel generation, we can instead count each assigned cap as its cost and ensure that the sum of caps stays within the total budget. All caps can then be fixed before rollout, allowing answers to be generated in parallel.

7 Experiments

We test whether STL answers more questions correctly when all methods receive the same total token allowance. We compare three models on two mathematics datasets, vary the available budget, and test transfer between datasets.

Draft status. The setup below describes direct-budget training. Table 1, Figure 3, and Sections 7.2–7.3 retain results from the earlier procedure. Section 7.4 reports the completed direct-budget cross-domain experiment; the primary table and budget sweep have not yet been replaced with results from this revised procedure.

7.1 Experimental Setup

Data and models. We use **OlympiadBench** (He et al., 2024) and **AIME-extended** (1983–2025). Training questions are used to learn how to allocate tokens; separate test questions are used to measure accuracy. OlympiadBench has 160 training and 474 test questions; AIME has 200 training and 600 test questions. We evaluate **GPT-5.4 mini**, **GPT-5.4 nano**, and **Qwen3-235B-A22B-Instruct-2507** (Qwen3). All three models use the same training and test questions. Each method produces one answer per test question. STL chooses a token cap from 256, 512, 1024, or 2048, with 4096 also available on AIME. A cap is the maximum number of tokens an answer may use.

Training. For each training question, we collect 16 answers at each cap and record whether each answer is correct and how many tokens it uses. These records let us estimate accuracy and average token cost for each question at each cap. For a target average budget B, we use Lagrangian relaxation to assign a cap to each training question, aiming to maximize accuracy while keeping the estimated average cost within B. We then train a small gradient-boosted classifier to predict these assigned caps from question features. Only the allocation classifier is trained; the reasoning LLM is unchanged.

Methods compared. **Fixed-b** uses the same token cap for every question. **Heuristic** assigns larger caps to longer questions. **Random** assigns caps randomly. All three methods satisfy the same total budget constraint. **Oracle** knows every response outcome in advance and finds the most correct answers within the budget. It provides an offline upper reference. Appendix B.2 gives the implementation details.

Evaluation. The budget B is the allowed number of generated tokens per question, on average. For example, 100 questions at $B = 350$ share 35,000 tokens; individual questions may use more or fewer than 350. We count the tokens actually generated. Before each answer, the execution rule reserves enough tokens to give all remaining questions the smallest cap. It lowers the current cap if needed, then subtracts the tokens used. Short answers leave more tokens for later questions. Every batch stays within its total allowance (Section 6). We evaluate accuracy and actual token use on the test questions using previously generated answers. Deployable methods choose caps under the same total budget; the reservation rule enforces this limit for adaptive methods. All methods use the same answer-checking rule. Appendix B.2 gives the evaluation details. We use budgets $290/350/450$ on OlympiadBench and $294/460/650$ on AIME, giving 18 settings across three models. We report accuracy gains in percentage points with 95% confidence intervals and apply Holm's correction (Holm, 1979) to the STL–Random comparisons. Appendix B.2 describes the statistical procedure.

Table 1: Accuracy under a shared average token budget B. Values are accuracy (%), with actual tokens/question in parentheses. Bold marks the highest accuracy excluding Oracle. Δ is STL minus Random in percentage points; CI is the 95% confidence interval. A † marks a gain with Holm-adjusted $p < 0.05$ across eighteen comparisons. Oracle is the offline optimum given the full response outcomes of each replay.

	OlympiadBench								
	GPT-5.4 mini			**GPT-5.4 nano**			**Qwen3-235B-A22B**		
Method	$B = 290$	$B = 350$	$B = 450$	$B = 290$	$B = 350$	$B = 450$	$B = 290$	$B = 350$	$B = 450$
Fixed-b	14.6 (279.7)	19.5 (312.9)	33.6 (397.2)	14.2 (278.2)	17.7 (310.9)	28.8 (399.2)	0.9 (289.5)	2.5 (328.9)	6.0 (444.5)
Random	15.7 (288.8)	25.4 (349.2)	38.8 (449.3)	16.0 (288.2)	22.8 (349.1)	33.0 (449.4)	1.4 (288.2)	3.1 (347.5)	6.3 (446.9)
Heuristic	11.4 (264.2)	15.8 (348.9)	24.5 (448.1)	13.2 (270.0)	15.8 (349.0)	21.6 (449.0)	0.8 (286.7)	1.5 (340.4)	2.7 (449.7)
STL	**16.9** (271.3)	**33.7** (348.3)	**40.9** (448.9)	**20.2** (289.4)	**26.1** (349.4)	**33.8** (449.3)	**4.7** (289.7)	**9.1** (349.7)	**14.7** (446.5)
Δ (pp)	+1.18	+8.29†	+2.12	+4.21†	+3.33	+0.81	+3.31†	+6.02†	+8.46†
95% CI	[−0.95, +3.27]	[+4.22, +11.39]	[−2.43, +6.12]	[+2.00, +6.96]	[+0.32, +6.75]	[−2.64, +4.32]	[+1.90, +4.75]	[+3.80, +8.44]	[+5.27, +11.18]
Oracle	52.2 (289.7)	64.7 (349.1)	66.8 (370.7)	47.4 (289.6)	57.4 (349.0)	62.4 (412.1)	15.8 (289.6)	26.2 (349.2)	35.8 (448.5)

	AIME-extended								
	GPT-5.4 mini			**GPT-5.4 nano**			**Qwen3-235B-A22B**		
Method	$B = 294$	$B = 460$	$B = 650$	$B = 294$	$B = 460$	$B = 650$	$B = 294$	$B = 460$	$B = 650$
Fixed-b	2.2 (292.8)	12.4 (443.9)	18.7 (519.4)	1.5 (292.5)	8.8 (447.4)	13.8 (532.6)	0.0 (294.0)	0.2 (460.0)	0.8 (559.7)
Random	3.0 (291.2)	13.2 (452.8)	31.6 (644.6)	**2.6** (291.9)	9.2 (454.4)	20.0 (640.7)	0.1 (292.3)	0.3 (458.0)	2.2 (644.1)
Heuristic	2.6 (293.8)	12.6 (459.6)	27.8 (648.1)	2.3 (293.8)	8.2 (459.8)	17.0 (649.6)	0.2 (292.8)	1.9 (459.8)	3.2 (649.8)
STL	**3.3** (273.4)	**19.9** (426.9)	**35.3** (628.4)	2.0 (270.3)	**13.1** (401.5)	**26.7** (594.6)	**0.6** (274.9)	**6.1** (438.3)	**10.4** (612.6)
Δ (pp)	+0.30	+6.73†	+3.69	−0.61	+3.93†	+6.73†	+0.55	+5.77†	+8.13†
95% CI	[−0.92, +1.67]	[+3.92, +9.58]	[+0.75, +6.42]	[−1.50, +0.25]	[+1.42, +6.17]	[+3.92, +9.33]	[+0.17, +1.17]	[+4.42, +7.33]	[+6.00, +10.00]
Oracle	27.8 (293.8)	58.3 (459.0)	60.3 (487.9)	23.9 (293.7)	50.2 (459.1)	60.7 (624.7)	7.5 (293.4)	22.0 (458.7)	31.9 (648.0)

7.2 Main Results

STL beats the strongest baseline in 17 of 18 settings. In Table 1, the main number is accuracy and the number in parentheses is actual tokens used per question. Compared with the best of Fixed-b, Random, and Heuristic in each setting, STL wins $17/18$ times, with a mean gain of 3.90 percentage points and a maximum of 8.46 points. Random is the strongest baseline in 15 settings; Heuristic is strongest in the three Qwen3 settings on AIME. Relative to Random, STL gains 4.05 points on average. The table's confidence intervals and significance tests compare STL with Random; ten gains are significant after Holm correction.

The gain over Random averages 3.39pp for mini and nano across their twelve settings. For mini on OlympiadBench at $B = 350$, accuracy rises from 25.41% to 33.70%, a gain of 8.29pp. Qwen3 improves in all six settings, averaging 5.37pp; five gains are significant after correction. Its largest gain is on OlympiadBench at $B = 450$: accuracy rises from 6.28% to 14.74%, while spending is nearly the same (446.92 versus 446.49 tokens/question).

STL can improve accuracy while using fewer tokens. On AIME at $B = 460$ and 650, this holds for all three models compared with Random. For Qwen3 at $B = 650$, STL reaches 10.37% accuracy using 612.61 tokens/question; Random reaches 2.25% using 644.12. The single negative main result is nano on AIME at $B = 294$: STL loses 0.61pp, with a 95% interval of $[-1.50, +0.25]$. Qwen3's 0.55pp gain at the same budget is also not significant after correction. The medium- and high-budget AIME results show that STL improves accuracy over Random while using fewer tokens across all three models.

7.3 Results Across Budgets

Figure 3 compares 57 model–dataset–budget settings. STL outperforms the strongest baseline in 50 settings, with an average gain of 3.84 percentage points and a maximum gain of 8.98 points. Accuracy generally increases with the budget, and gains appear across different budget ranges. On AIME, STL leads in 25 of 27 settings; on OlympiadBench, Qwen3 leads at all ten budgets.

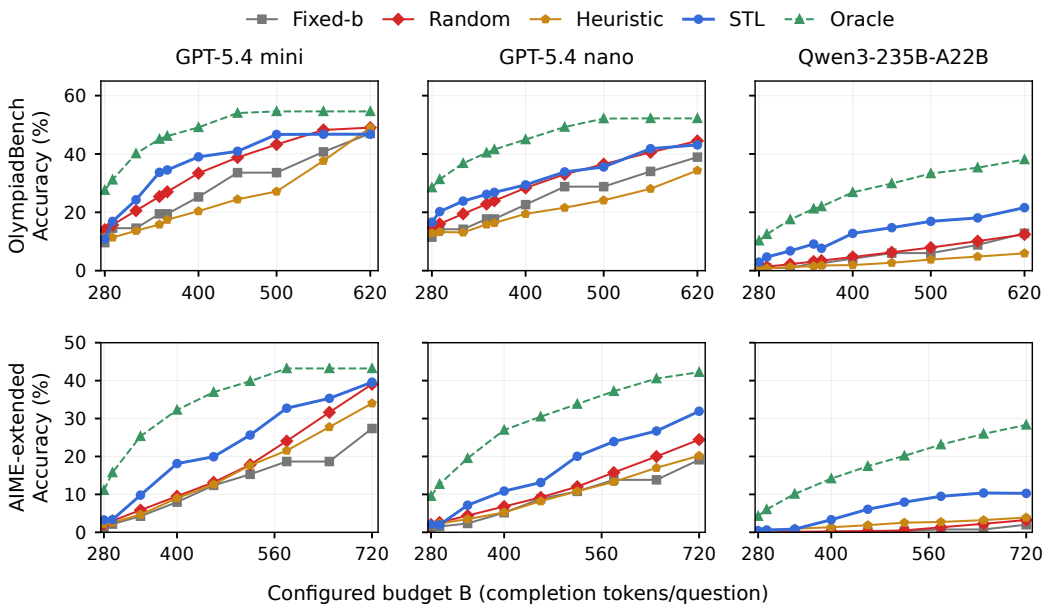

Figure 3: Accuracy as the allowed average budget increases. Each panel shows one model on one dataset. The horizontal axis is the allowed tokens per question; the vertical axis is the percentage of questions answered correctly. All methods use the same budget points within a panel. Oracle uses extra test-question answers and is shown as a reference.

STL can also achieve similar accuracy with a smaller budget. For mini on AIME, it reaches 18.13% accuracy at $B = 400$, compared with the strongest baseline's 17.83% at $B = 520$, using a 23% smaller configured budget.

7.4 Cross-Domain Generalization

We test whether an STL classifier trained on one dataset can allocate tokens effectively on another. For each model and budget, we train on OlympiadBench and test on AIME, then reverse the direction, without retraining on the new dataset. Figure 4 shows three models at eight budgets on OlympiadBench and nine on AIME, giving 51 settings. Each panel compares STL trained on the other dataset, STL trained on the dataset being tested, and the strongest of Fixed-b, Random, and Heuristic, using the same test questions and budget. Appendix B.3 gives the detailed settings.

Transferred STL outperforms the strongest baseline in 33 of 51 settings, averaging a 1.90-point accuracy gain. Transfer from OlympiadBench to AIME is particularly effective: STL leads in 25 of 27 settings by 3.85 points on average, while its mean accuracy is 0.66 points below STL trained on AIME. In the reverse direction, Qwen3 leads at six of eight budgets with a mean gain of 2.46 points; the average difference across all three models is -0.28 points. Accuracy generally rises with the available budget in both directions. These results show that learned allocation can remain useful across datasets, with the benefit depending on the transfer direction and model.

7.5 Classifier Ablation

We compare five classifiers within STL: gradient boosting, logistic regression, RBF SVM, random forest, and MLP. For each model and budget, we solve the allocation problem at the target budget and train each classifier on the same cap labels. We keep the training questions, features, cap choices, and execution rule fixed. Figure 5 covers the same 51 settings as Figure 4; detailed classifier settings are provided in Appendix B.3.

Logistic regression achieves the highest average accuracy and ranks first in 29 of 51 settings. Averaged across settings, logistic regression and MLP improve accuracy over gradient boosting by 0.98 and 0.65 percentage points, respectively. Several classifiers reach similar accuracy at larger budgets on

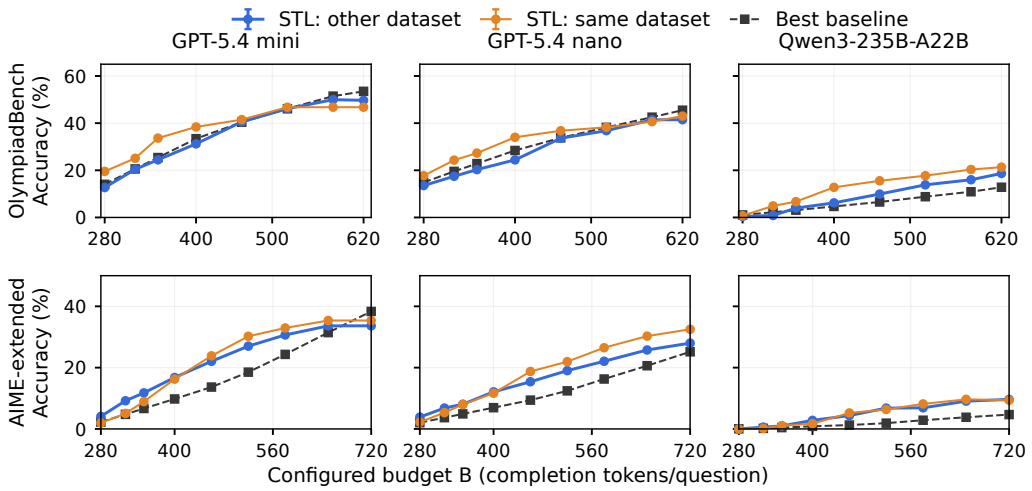

Figure 4: Cross-domain generalization. Top: AIME to OlympiadBench; bottom: the reverse. Blue and orange show STL trained on the other and the same dataset, respectively. Dashed black shows the strongest baseline. Points and error bars show the mean and standard deviation over three training seeds.

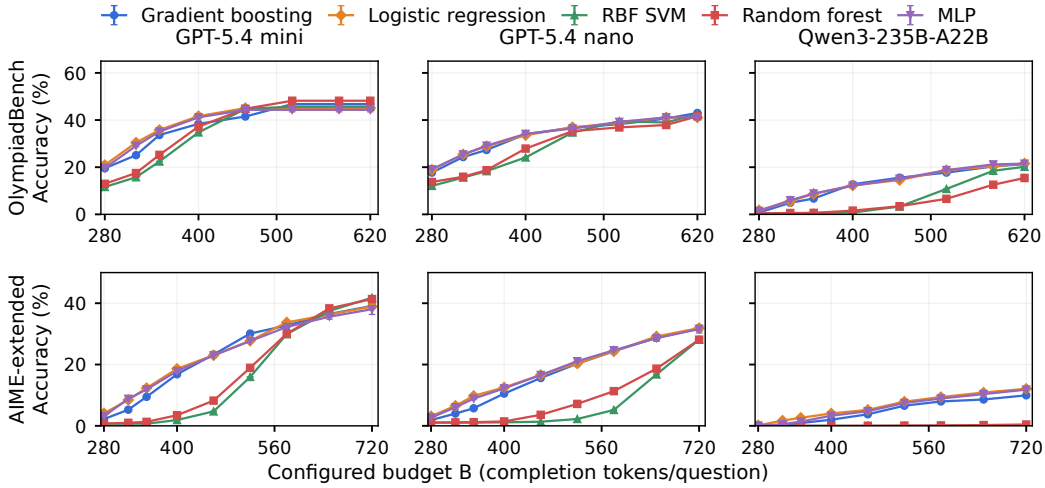

Figure 5: Classifier ablation within STL. Each curve uses a different classifier to predict token caps under the same budget. Points and error bars show the mean and standard deviation over three training seeds.

mini and nano, while their differences remain clear on Qwen3. These results show that STL supports different classifiers, with logistic regression and MLP providing strong overall performance.

8 Conclusion

We study accuracy maximization under a shared average compute budget and formulate token allocation as a constrained optimization problem. Solve-then-Learn uses Lagrangian optimization to assign training labels, then trains a classifier to select token caps for new questions. A budget-aware execution rule enforces the total allowance. Experiments across two benchmarks and three language models show that learned allocation can improve accuracy under a common budget. Cross-domain and classifier studies examine its transferability and design choices. Future work includes broader tasks, transfer across models, and end-to-end serving costs.

AI USE STATEMENT

We used AI tools to assist with coding, experimental design and analysis, and manuscript editing. The authors take responsibility for the final content, including its claims, results, and citations.

ETHICS STATEMENT

This work studies compute allocation for language-model reasoning using existing mathematical benchmarks. Our experiments focus on accuracy and computational cost. The proposed method inherits the reliability limitations of its underlying language models. Practical applications should consider domain-specific accuracy requirements and the consequences of incorrect outputs.

REPRODUCIBILITY STATEMENT

The main text describes the optimization problem, learning procedure, and budget-constrained execution. Appendix A provides theoretical assumptions and proofs. Appendix B documents the datasets, features, model configurations, and evaluation protocols, including those for the cross-domain and classifier-ablation studies.

REFERENCES

P. Aggarwal, A. Madaan, Y. Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

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

B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

L. Chen, M. Zaharia, and J. Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

M. Damani, I. Shenfeld, A. Peng, A. Bobu, and J. Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In *International Conference on Learning Representations (ICLR)*, 2025.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Rühle, L. V. S. Lakshmanan, and A. H. Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

T. Han, Z. Wang, C. Fang, S. Zhao, S. Ma, and Z. Chen. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics (ACL)*, 2025.

Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7436–7456, 2022.

C. He, R. Luo, Y. Bai, S. Hu, Z. L. Thai, J. Shen, J. Hu, X. Han, Y. Huang, Y. Zhang, J. Liu, L. Qi, Z. Liu, and M. Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

Y. Li, P. Yuan, S. Feng, B. Pan, X. Wang, B. Sun, H. Wang, and K. Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *International Conference on Learning Representations (ICLR)*, 2024.

Z. Li, Q. Dong, J. Ma, D. Zhang, K. Jia, and Z. Sui. SelfBudgeter: Adaptive token allocation for efficient LLM reasoning. *arXiv preprint arXiv:2505.11274*, 2025.

H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Z. Liu, H. Qu, X. Yin, H. Sun, Y. Han, T. Chen, and Z. Deng. PETS: A principled framework towards optimal trajectory allocation for efficient test-time self-consistency. *arXiv preprint arXiv:2602.16745*, 2026.

N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

M. Nguyen, S. Gupta, and H. Le. Uncertainty-aware budget allocation for adaptive test-time reasoning. *arXiv preprint arXiv:2605.26849*, 2026.

I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica. RouteLLM: Learning to route LLMs with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

OpenAI. Learning to reason with LLMs, September 2024. URL.

P. Panda, R. Magazine, C. Devaguptapu, S. Takemori, and V. Sharma. Adaptive LLM routing under budget constraints. *arXiv preprint arXiv:2508.21141*, 2025.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

S. Teerapittayanon, B. McDanel, and H.-T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR)*, 2016.

G. Wan, Y. Wu, J. Chen, and S. Li. Reasoning aware self-consistency: Leveraging reasoning paths for efficient LLM sampling. In *Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Y. Wu, Z. Sun, S. Li, S. Welleck, and Y. Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *International Conference on Learning Representations (ICLR)*, 2025.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

A PROOFS

We explain the solve, learn, and budget-execution guarantees in turn. Each proof uses a fixed collection or distribution of questions.[1]

A.1 MONOTONICITY OF THE ALLOCATION COST

The multiplier λ controls the penalty for cost in the score $\mathrm{Acc}(x, b) - \lambda \mathrm{C}(b)$. Increasing this penalty makes expensive options less attractive. We show that the cost of the selected option cannot increase.

Lemma 1 (Cost monotonicity)**.** *Fix a question x and two penalties $0 \le \lambda < \lambda'$. Let b be the option selected at λ and b' the option selected at λ'. Then $\mathrm{C}(b') \le \mathrm{C}(b)$. Averaging over questions, the allocation cost $\bar{C}(\lambda)$ is also non-increasing.*

Proof. At penalty λ, option b has a score at least as high as option b'. At penalty λ', the reverse comparison holds. Thus,

$$\mathrm{Acc}(x, b) - \lambda \mathrm{C}(b) \ge \mathrm{Acc}(x, b') - \lambda \mathrm{C}(b'),$$
$$\mathrm{Acc}(x, b') - \lambda' \mathrm{C}(b') \ge \mathrm{Acc}(x, b) - \lambda' \mathrm{C}(b).$$

Adding the two inequalities cancels the accuracy terms and gives

$$(\lambda' - \lambda)\big[\mathrm{C}(b) - \mathrm{C}(b')\big] \ge 0.$$

Since $\lambda' - \lambda > 0$, we have $\mathrm{C}(b') \le \mathrm{C}(b)$. This holds for each question, so it also holds after taking an average or expectation. The same proof applies when costs depend on the question: replace $\mathrm{C}(b)$ with $\mathrm{C}(x, b)$ throughout. □

When scores tie, we choose the option with lower cost, then the smaller index. For a fixed question, each change as λ increases is therefore to a cheaper option, giving at most $K - 1$ changes. Once λ is large enough, the selected option has minimum cost; among equally cheap options, it has the highest accuracy. This explains why increasing λ eventually gives a feasible allocation whenever the budget covers the cheapest options for the finite collection of questions.

A.2 COMPUTATIONAL COMPLEXITY OF THE LABEL SOLVER

Suppose the accuracy and cost values are already available for N questions, each with $K \ge 2$ candidate budgets. The total work is the work per search step multiplied by the number of steps. Let $L > 0$ be the initial search-interval width and $0 < \eta \le L$ the desired final width.

Proof. For one value of λ, computing all NK scores takes $O(NK)$ operations. Our solver orders the candidates by cost to apply the tie rule, which takes $O(NK \log K)$. Selecting the highest score for each question and averaging the selected costs takes another $O(NK)$. One allocation evaluation therefore costs $O(NK \log K)$.

Binary search halves the interval at each step. After t steps, its width is $L/2^t$, so the number of steps needed to reach a width of at most η is

$$T = \lceil \log_2(L/\eta) \rceil .$$

Including the evaluations at the interval endpoints, the total work is

$$O(NK \log K\,(1 + T)) = O(NK \log K\,[1 + \log(L/\eta)]) .$$

□

This calculation starts after a search interval has been found and treats arithmetic as exact. Finding the interval takes additional allocation evaluations. Generating answers and training the classifier have separate costs.

[1] Accuracy, cost, and policy functions are assumed measurable, so the expectations and probabilities used below are well defined.

A.3 Proof of Theorem 1

We bound the accuracy difference between the returned allocation and the best allocation that meets the budget. Here, $\bar{A}$ is the returned allocation's average accuracy, $\bar{C}(\lambda_B)$ is its average cost, and $\hat{V}(B)$ is the best feasible average accuracy on the same questions and accuracy table.

Proof. First, we find an upper bound on $\hat{V}(B)$. For any feasible allocation π, the Lagrangian equals its average accuracy plus λ_B times its unused average budget. This extra term is nonnegative. Also, $G(\lambda_B)$ is the largest Lagrangian value over all allocations. Hence,

$$\frac{1}{N}\sum_{i=1}^{N}\mathrm{Acc}(x_i,\pi(x_i)) \le \mathcal{L}(\pi,\lambda_B) \le G(\lambda_B).$$

This holds for every feasible allocation, including the best one, so $\hat{V}(B) \le G(\lambda_B)$. The returned allocation is itself feasible, giving $\bar{A} \le \hat{V}(B)$.

Next, we compute the difference between this upper bound and $\bar{A}$. Each returned option $y_i^\star$ has the highest score for its question at λ_B. Substituting these options into G gives

$$\begin{aligned} G(\lambda_B) &= \frac{1}{N}\sum_{i=1}^{N}\left[\mathrm{Acc}(x_i,y_i^\star) - \lambda_B \mathrm{C}(y_i^\star)\right] + \lambda_B B \\ &= \bar{A} + \lambda_B\left[B - \bar{C}(\lambda_B)\right]. \end{aligned}$$

Together, these two steps give

$$0 \le \hat{V}(B) - \bar{A} \le \lambda_B\left[B - \bar{C}(\lambda_B)\right] = \delta_B.$$

If the returned allocation uses the full average budget, or if $\lambda_B = 0$, this upper bound is zero. Its accuracy then equals the best feasible accuracy, proving optimality. □

In general, δ_B is an upper bound on the accuracy gap. It is the cost penalty multiplied by the unused average budget. Because the candidate budgets are discrete, allocation cost can change in jumps; narrowing the search interval alone need not make this gap zero.

For costs that depend on the question, the same proof uses $\mathrm{C}(x_i, b)$ in each row. A feasible allocation exists when $B \ge N^{-1}\sum_i \min_{b\in\mathcal{B}} \mathrm{C}(x_i, b)$: the budget must cover the average cost of choosing each question's cheapest option.

A.4 Proof of Theorem 2

Let $A(\pi)$ be a policy's average accuracy and ε_B the probability that the classifier predicts a different cap from the oracle. We compare both policies on the same question distribution, with the trained classifier and oracle rule held fixed.

Proof. Step 1: compare the caps for one question. If the classifier predicts the oracle cap, both policies have the same accuracy on that question. If it predicts a different cap, their accuracies differ by at most 1, because both lie in $[0, 1]$. Thus,

$$|\mathrm{Acc}(x,\pi_B^\star(x)) - \mathrm{Acc}(x,\hat{\pi}_B(x))| \le \mathbf{1}\{\hat{\pi}_B(x) \ne \pi_B^\star(x)\}.$$

The expression on the right is 1 when the caps differ and 0 when they agree.

Step 2: average over questions. The difference between the average accuracies is at most the average of the individual accuracy differences. Averaging the bound above therefore gives

$$|A(\pi_B^\star) - A(\hat{\pi}_B)| \le \mathbb{E}_{x\sim\mathcal{D}}[\mathbf{1}\{\hat{\pi}_B(x) \ne \pi_B^\star(x)\}] = \varepsilon_B.$$

In particular, $A(\hat{\pi}_B) \ge A(\pi_B^\star) - \varepsilon_B$, proving (8). For example, a cap-prediction error rate of 5% gives an accuracy loss of at most 5 percentage points relative to the oracle.

Step 3: add the solve-stage error. Now average over the same N questions and accuracy table used in Theorem 1. Then $A(\pi_B^\star) = \bar{A}$, and ε_B is the fraction of these questions whose predicted caps differ from their oracle labels. The solve stage has accuracy loss at most δ_B, and Step 2 adds at most ε_B:

$$\begin{aligned}\hat{V}(B) - A(\hat{\pi}_B) &= \left[\hat{V}(B) - \bar{A}\right] + \left[\bar{A} - A(\hat{\pi}_B)\right] \\ &\le \delta_B + \varepsilon_B.\end{aligned}$$

Rearranging proves (9). □

Why the coefficient cannot be smaller. Consider two caps with costs 1 and 2 and accuracies 0 and 1 for every question. Let $B = 2$ and $\lambda_B = 0$. The oracle always chooses the second cap, so its accuracy is 1 and $\delta_B = 0$. If the classifier chooses the first cap with probability ε_B, its accuracy is $1 - \varepsilon_B$ and its average cost is $2 - \varepsilon_B \le B$. Its accuracy loss is exactly ε_B, so the coefficient 1 cannot be reduced without further assumptions.

Where the bounds apply. For new questions, (8) uses the cap-prediction error on their distribution; training error alone does not give this quantity. The combined bound (9) uses the same questions and accuracy table as the solve-stage bound. When that table contains estimated accuracies, the guarantee is relative to those estimates. Both results concern the caps requested by the classifier, before the deployment rule adjusts any cap. We prove that rule's budget guarantee next.

A.5 Hard budget execution

The rule reserves enough tokens to give every remaining question the smallest cap. Consider M questions and a finite set $\mathcal{B}$ of positive integer token caps, with smallest cap b_1. Each requested cap $\hat{b}_i$ belongs to $\mathcal{B}$. Let $B \ge b_1$, set the total budget to $H = \lfloor MB \rfloor$, and start with $R_1 = H$ tokens.

Before question i, reserve $(M - i)b_1$ tokens for the later questions. Choose the largest cap $b_i \in \mathcal{B}$ that is no greater than both $\hat{b}_i$ and $R_i - (M - i)b_1$. After generating the answer, subtract its actual completion-token cost c_i, where $0 \le c_i \le b_i$, and set $R_{i+1} = R_i - c_i$. The cap is chosen before the current answer and its cost are observed.

Feasibility. Step 1: the initial budget is sufficient. Since $B \ge b_1$ and Mb_1 is an integer,

$$R_1 = \lfloor MB \rfloor \ge Mb_1.$$

There are enough tokens to assign the smallest cap to all M questions.

Step 2: one answer preserves this property. Suppose that before question i, $R_i \ge (M - i + 1)b_1$. After reserving tokens for the later questions, at least b_1 tokens are available for the current question. Since $\hat{b}_i \ge b_1$, the smallest cap is a valid choice, so the rule can always select a cap. The actual cost satisfies

$$c_i \le b_i \le R_i - (M - i)b_1.$$

Subtracting it gives $R_{i+1} = R_i - c_i \ge (M - i)b_1$. Enough tokens remain to assign the smallest cap to every later question.

Step 3: the total cost stays within budget. Repeating Step 2 for all questions gives $R_{M+1} \ge 0$. Hence,

$$\sum_{i=1}^{M} c_i = H - R_{M+1} \le H.$$

□

This guarantee requires the generation system to respect each assigned cap and the recorded cost to count the completion tokens covered by that cap. The rule charges actual token use. If an answer ends early, its unused allowance remains available for later questions. The sum of assigned caps can therefore exceed H, while actual spending stays within H. The requested caps are fixed in advance; the executed caps also depend on earlier token use. Fixed-b satisfies the same total limit because its integer cap is at most B.

B Experimental Settings

B.1 Data and Training

All three models use the same training and test questions: $160/474$ on OlympiadBench and $200/600$ on AIME-extended. The candidate caps are 256, 512, 1024, and 2048, with 4096 also available on AIME. Training questions have 16 stored answers per cap, divided into two groups of eight. Test questions normally have four answers per cap, divided into two groups of two; two partially collected mini/nano groups use their one available answer. Cost is the number of completion tokens actually generated. Qwen3 uses temperature 0.7; the earlier mini/nano records do not contain complete sampling settings.

The classifier uses 48 features computed from the question before answer generation: 16 text features and 32 embedding coordinates. The table below lists the complete feature set.

Group	Dim.	Features
Length and vocabulary	6	Character count; word count; sentence-count estimate from punctuation; question-mark count; mean word length; proportion of distinct words.
Numbers	3	Number count; mean number magnitude; largest number magnitude. Both magnitudes use a log scale.
Presence indicators	7	Percentages; fractions; time words; currency symbols or money words; rate or speed words; sequence words such as "then" or "next"; decimal numbers.
Question embedding	32	Principal-component coordinates of text-embedding-v4 question embeddings.

Words are separated by whitespace, with punctuation retained; distinct words are counted after lowercasing. The sentence estimate counts periods, exclamation marks, and question marks, with a minimum of one. Number magnitude is $\log_{10}(1 + |v|)$ for a number v; its mean and maximum are zero when no numbers occur. Each presence indicator is 1 when a matching symbol or word occurs and 0 otherwise. Currency symbols include dollar signs in mathematical notation. We fit the embedding dimensionality reduction on training questions and keep it fixed for testing. Generated answers are not classifier inputs.

The default classifier is histogram gradient boosting, with 200 iterations, maximum depth 4, learning rate 0.1, minimum leaf size 20, and no early stopping. Training examples have equal weight. If all cap labels are identical, the predictor always returns that cap.

Table 1 and Figure 3 use an additional cost-calibration step. We train candidate classifiers at feasible budgets in $0.50B, 0.55B, \ldots, 2.20B$, also including lower and upper endpoints determined from training costs. Separate questions—40 on OlympiadBench and 50 on AIME—estimate each candidate's mean token cost. We select the candidate whose cost is closest to B without exceeding it, using costs rather than answer accuracy, and keep that classifier fixed. These fits use seed 0. The cross-domain and classifier experiments instead solve and train directly at B, using only the $160/200$ training questions.

B.2 Baselines and Evaluation

Fixed-b chooses the largest sampled cap no greater than B and uses it for every question. Extra sampled caps are $290, 330, 380, 450, 520, 620$ on OlympiadBench and $294, 340, 400, 460, 560, 700$ on AIME. Random mixes two neighboring caps in estimated-cost order to meet the expected budget. Heuristic gives priority to longer questions when assigning larger caps. For Table 1 and Figure 3, Random uses the separate cost-calibration questions, and Heuristic's allocation level is also selected using their costs. In the additional experiments, both use costs estimated from training questions. STL, Random, and Heuristic share the actual-token reservation rule in Section 6.

We evaluate with previously generated answers. For each training-response group, we fit a policy and evaluate it on the opposite test-response group, then swap groups. One evaluation processes all test questions in a fixed order, draws an answer recorded at each executed cap, and charges its actual token use. Adaptive-method means average 100 evaluations per group, or 200 in total, with paired

answer draws across methods. Fixed-b averages all stored answers at its uniform cap. We score the last boxed answer, or a numeric answer following "Answer", using Math-Verify 0.9.0 with strict answer comparison. Unparseable answers and prediction-check timeouts count as incorrect; empty and incomplete responses remain in the evaluation.

The Oracle in Table 1 sees one sampled response at every candidate cap for each question, including its correctness and token cost. It selects one response per question to maximize the number correct within the total budget, including the extra Fixed-b caps. Figure 3 uses a different reference: one response group selects caps, and the other measures accuracy under the reservation rule. This reference is evaluated on different answers from those used to choose caps, so it is not an accuracy upper bound.

For the 18 primary STL–Random comparisons, we use 10,000 paired bootstrap resamples of test questions, re-executing the budget rule each time. The 2.5th and 97.5th percentiles give 95% intervals. Two-sided tests use centered bootstrap differences; Holm's correction (Holm, 1979) across the 18 tests determines significance at 0.05. Intervals remain unadjusted. The analysis holds classifiers and previously inspected answer pools fixed. Training and test question IDs are separate, although some OlympiadBench subquestions share problem descriptions. Reported costs count test-answer completion tokens only.

B.3 Cross-Domain and Classifier Experiments

Figures 4 and 5 report 51 settings across the three models. We use budgets of 280, 320, 350, 400, 460, 520, 580, and 620 on OlympiadBench, and 280, 320, 350, 400, 460, 520, 580, 650, and 720 on AIME.

For cross-domain transfer, we learn both the feature transformation and the classifier using training questions from one dataset, then apply them to test questions from the other dataset without retraining. Both directions use the common cap set $\{256, 512, 1024, 2048\}$. The same-dataset comparator is trained on separate training questions from the dataset being tested. Its training questions also provide the cost estimates for the reference baselines.

For the classifier ablation, all five classifiers receive the same training questions, cap labels, and features. The cap set has four options on OlympiadBench and additionally includes 4096 on AIME. Logistic regression, SVM, and MLP standardize features using means and scales fitted on training questions. Gradient boosting uses the settings above; the other classifiers use the following configurations.

Classifier	**Main settings**
Logistic regression	L2 regularization, $C = 1$, L-BFGS optimization, up to 2000 iterations.
RBF SVM	$C = 1$; kernel coefficient $1/(d\ \mathrm{Var}(X))$, where X is the standardized training matrix and $d = 48$.
Random forest	100 trees, bootstrap sampling, 6 randomly chosen features considered per split, and unrestricted tree depth.
MLP	One 64-unit ReLU hidden layer, Adam, learning rate 0.001, L2 penalty 0.0001, up to 800 iterations, and no early stopping.

Both studies use training seeds $0, 1, 2$ and the same 200-evaluation procedure described above. Figure points show mean accuracy across these seeds; error bars show their sample standard deviation. Summary comparisons weight the 51 displayed settings equally. No test accuracy is used to tune the classifiers. Other classifier options in these two studies use the defaults of scikit-learn 1.9.1.