

SELFBUDGETER: ADAPTIVE TOKEN ALLOCATION FOR EFFICIENT LLM REASONING

Anonymous authors

Paper under double-blind review

ABSTRACT

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

1 INTRODUCTION

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

However, reasoning models tend to use overly long thought processes even for simple questions. This “overthinking” phenomenon leads to a waste of computational resources and excessive user waiting times (Chen et al., 2024; Sui et al., 2025). For example, when answering the simple questions such as “What is the answer of $2+3$?", the QwQ-32B model provides 13 different solutions and generates 100 times more tokens than Owen2.5-72B-Instruct model (Owen et al., 2025).

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

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

054 Additionally, when specific requirements arise,
 055 users can pre-fill the token budget field to con-
 056 strain the model’s response within the given
 057 limit, thereby improving interaction efficiency.
 058

059 Our training framework consists of two main
 060 stages. During the Cold-Start stage, we fine-
 061 tune the model to learn how to first output its
 062 estimated token budget within `<budget>` tags.
 063 Subsequently, in the RL training stage, we op-
 064 timize SelfBudgeter using the GRPO algorithm.
 065 For this stage, we design a reward function that
 066 primarily focuses on three key aspects: (1) an-
 067 swer correctness, (2) minimal achievable token
 068 budget, and (3) consistency between response
 069 length and the allocated token budget.

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

085 2 RELATED WORK

087 **Overthinking in LLMs** The emergence of the reasoning models like O1, Deepseek-R1 and
 088 QwQ advanced complex problem-solving through RL-enhanced CoT (Wei et al., 2022; OpenAI,
 089 2024; DeepSeek-AI et al., 2025; Qwen, 2024). However, researchers observed a tendency for
 090 reasoning models to overthink simple problems—expending unnecessary computational effort on
 091 trivial queries (Chen et al., 2024; Sui et al., 2025). Excessive long CoT may lead to a decrease
 092 in accuracy (Wu et al., 2025). Current solutions for overthinking mainly involve following three
 093 strategies. Prompt-based methods try to control response length by adding instructions in prompts,
 094 but cannot control the length accurately (Lee et al., 2025; Renze & Guven, 2024; Xu et al., 2025a;
 095 Nayab et al., 2024). Integrated Training-based methods try to teach model decide the length by
 096 the difficulty of the problems. Supervised fine-tuning(SFT)-based methods collect the dataset with
 097 variable length (Munkhbat et al., 2025; Ma et al., 2025; Liu et al., 2024; Han et al., 2024; Kang et al.,
 098 2024; Xia et al., 2025; Yang et al., 2025b). RL-based methods incorporate length penalties into the
 099 reward function (Aggarwal & Welleck, 2025; Arora & Zanette, 2025; Luo et al., 2025a; Chen et al.,
 100 2025a; Chang et al., 2025; Xu et al., 2025b; Yang et al., 2025a). These methods fail to control the
 101 length as users’ requirements. And Router-based methods train another model as a classifier (Ayes
 102 et al., 2025; Chuang et al., 2025; 2024; Ong et al., 2024; Pan et al., 2025). The classifier decide
 103 to route the query to fast models or reasoning models. However, an extra classifier means more
 104 computation resources are needed. Current methods either sacrifice precise control, require extra
 105 computation, or fail to bridge autonomous budget estimation with strict adherence.

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

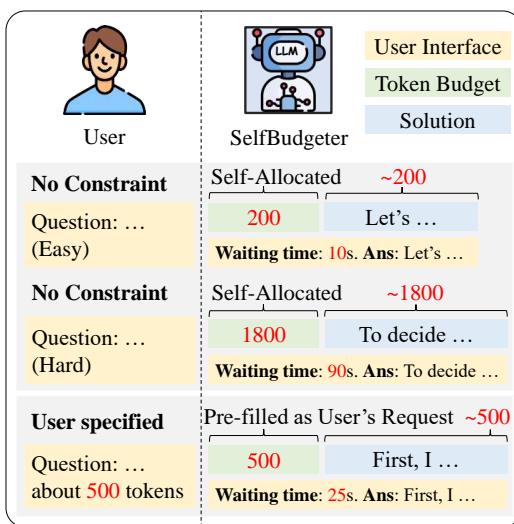


Figure 1: Overview of the SelfBudgeter. SelfBudgeter’s responses comprise two sections: **Token Budget** and **Solution**. For unrestricted queries, it estimates tokens needed for the **Solution** based on problem complexity. When users specify requirements, it pre-fills the **Token Budget** accordingly.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

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

3 METHOD

121 To minimize the overthinking problem in LLMs, we propose SelfBudgeter for efficient reasoning.
 122 Our method aims to enable the model to autonomously determine an appropriate token budget and
 123 generate responses of corresponding length while adhering to this budget. Although reasoning models
 124 may occasionally overthink simple problems, their response lengths generally increase with problem
 125 difficulty. This phenomenon demonstrates that the model possesses the capability to allocate token
 126 quantities reasonably based on problem complexity. Previous works such as L1 (Aggarwal & Welleck,
 127 2025) and Elastic Reasoning (Xu et al., 2025b) have also demonstrated that models can generate
 128 responses of appropriate length according to a given token budget.

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

3.1 SELF BUDGETER

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

140 Our reward function is formally defined as Formula 1:

$$143 \quad R(C, F, \ell, b, b_{\max}) = \begin{cases} r_f, & \text{if } F = 0, \\ 144 \quad P_B(b, b_{\max}) + \text{PreB}(s_{\min}^W, s_{\max}^W, \ell, b, \alpha, b_{\text{best}}^W), & \text{if } F = 1 \text{ and } C = 0, \\ 145 \quad P_B(b, b_{\max}) + \text{PreB}(s_{\min}^C, s_{\max}^C, \ell, b, \alpha, b_{\text{best}}^C), & \text{if } F = 1 \text{ and } C = 1. \end{cases} \quad (1)$$

147 where

$$148 \quad b_{\text{best}}^C = (1 - \alpha) \cdot b, \quad b_{\text{best}}^W = (1 + \alpha) \cdot b \quad (2)$$

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

3.2 BUDGET PENALTY

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

$$161 \quad P_B(b, b_{\max}) = \begin{cases} 0, & \text{if } b \leq b_{\max}, \\ r_b, & \text{else.} \end{cases} \quad (3)$$

162

163

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

164

165

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

170

171

172

3.3 PRECISE BUDGET CONTROL REWARD

173

174

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

175

176

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

177

178

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

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

3.4 ACCURACY REWARD

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

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

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

4 EXPERIMENT

4.1 TRAINING TEMPLATE

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

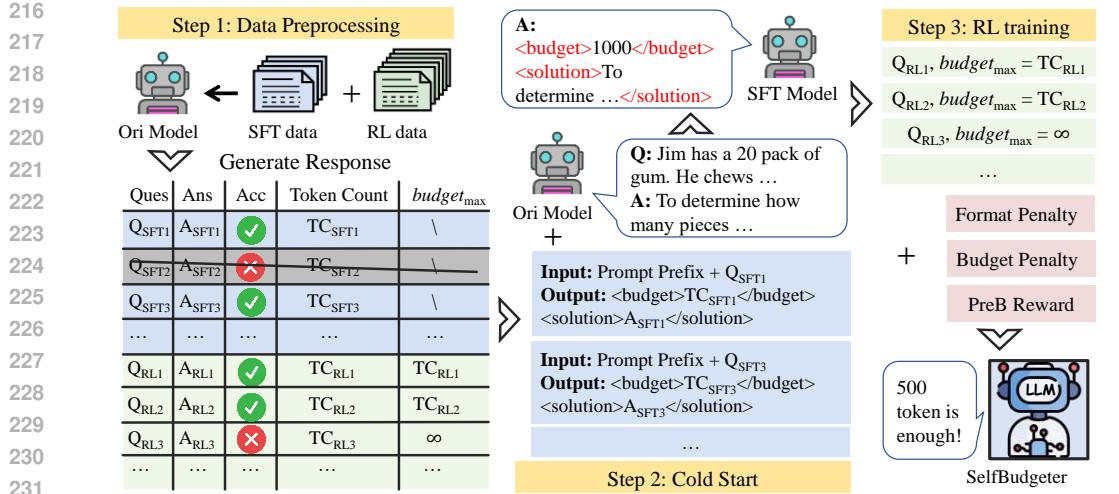


Figure 2: Overview of the SelfBudgeter training framework. The workflow consists of three key steps: (1) **Data preprocessing**: The initial model preprocesses the data to compute token budgets for subsequent training; (2) **Cold-start**: The model is fine-tuned to adopt the new output format; (3) **RL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

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

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

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

4.2 DATA PREPROCESSING

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

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

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

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

4.3 COLD START

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

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

275 To prevent the model from losing its original reasoning capability during the cold-start phase, fine-
 276 tuning must be performed using either the model’s own generated responses or datasets containing
 277 long CoT responses. In our approach, we pre-populate the `<budget>` section with token counts
 278 obtained during the preprocessing stage. The `<solution>` section is filled with the model’s
 279 generated responses. And the instruction prefix we prepend to each question can be found in
 280 Appendix B .
 281

282 4.4 EXPERIMENT SETTINGS

283 We conduct experiments on the DeepSeek-R1-Distill-Qwen-1.5B (R1-1.5B) model. We reproduce L1-
 284 Max using R1-1.5B, and select R1-1.5B and L1-Max as baseline methods for comparative evaluation
 285 against SelfBudgeter. In addition, we extend our experiments to the larger DeepSeek-R1-Distill-
 286 Qwen-7B (R1-7B) model. For more comprehensive comparison, we also include E1-Math-1.5B,
 287 R1-7B, Eurus-2-7B-PRIME (Cui et al., 2025), and Qwen-2.5-7B-Simple-RL (Shao et al., 2024) as
 288 additional baselines.
 289

290 During the cold-start phase, we employ three datasets of varying difficulty—GSM8K (Cobbe et al.,
 291 2021), MATH (Hendrycks et al., 2021), and s1k-1.1 (Muennighoff et al., 2025)—to help the model
 292 learn the new output format while producing token budgets with diverse distributions. The s1k-1.1
 293 dataset contains 1,000 challenging mathematical problems with long reasoning chains generated
 294 by DeepSeek-R1, which support both reasoning ability and format adaptation. For GSM8K and
 295 MATH, we select 1,500 training samples each that the model can answer correctly. For s1k-1.1,
 296 we directly use the native responses and compute the corresponding token counts with the model’s
 297 tokenizer to populate our designed template; in total, we retain 630 problems that DeepSeek-R1
 298 answered correctly. This yields a training set of 3,630 samples. Following the preprocessing protocol
 299 in Sections 4.2 and 4.3, we fine-tune the model for one epoch. Throughout data collection and
 300 training, the model’s temperature is consistently set to 0.6.

301 During the reinforcement learning phase, we use STILL-3-Preview-RL-Data (Chen et al., 2025b)
 302 dataset. It also serves as the training dataset for reproducing L1-max. This dataset collects 30K
 303 high-quality samples based on the MATH (Hendrycks et al., 2021), NuminaMathCoT (LI et al.,
 304 2024), and AIME 1983-2023 (Veeraboina, 2023) datasets. It includes problems of varying difficulty
 305 levels, which also helps the model learn to allocate token counts adaptively based on difficulty. As
 306 described in Section 4.2, we compute the maximum acceptable budget (b_{\max}) based on the model’s
 307 responses, then train the model for 3 epochs on this dataset. More detailed information can be found
 308 in Appendix A.

309 4.5 MAIN RESULTS

310 Table 2 presents a comprehensive comparison of model performance on the GSM8K, MATH500,
 311 and AIME2025 test sets, evaluated in terms of accuracy (Acc) and average response length (Len).
 312 The table contrasts baseline models with different variants of the SelfBudgeter framework across
 313 varying model scales. For clarity, the best performance is highlighted in bold, while the second-best
 314 performance is indicated with underline. It is worth noting that token limits for L1 are explicitly
 315 specified through prompt templates, whereas those for E1 are enforced via hard truncation. In contrast,
 316 SelfBudgeter autonomously estimates its token constraints during inference. All reported results are
 317 averaged over three runs with different random seeds.
 318

319 **Baseline Comparison** Although the Deepseek-R1-Distill-Qwen-1.5B baseline demonstrates strong
 320 accuracy, it requires substantially longer responses. On GSM8K, our method improves accuracy by
 321 11.01 percentage points while compressing response length to 43% of the original. On MATH500,
 322 it achieves a 3.54-point accuracy gain with response length reduced to 44%. On AIME2025, our
 323 approach compresses response length to 30% of the original while maintaining comparable accuracy.
 In contrast, although L1 and E1 attain stronger compression on certain datasets, they incur larger

324
 325 Table 2: Performance comparison on GSM8K, MATH500, and AIME2025. Accuracy (Acc) is
 326 reported in percentage, and length (Len) in tokens.

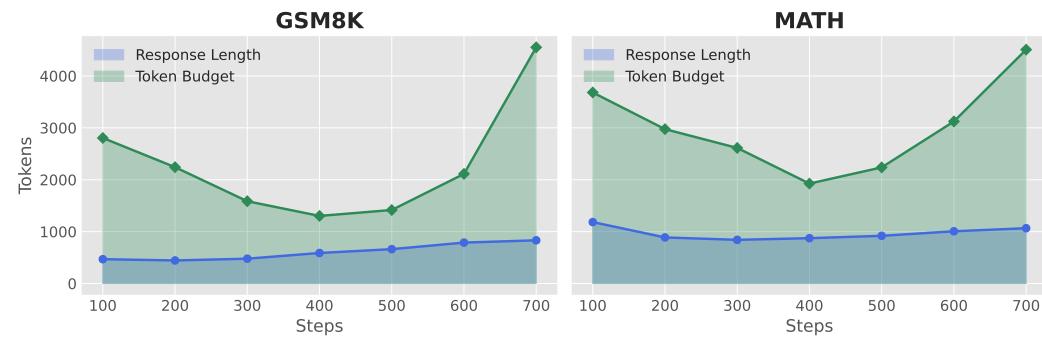
Models	GSM8K		MATH500		AIME2025	
	Acc	Len	Acc	Len	Acc	Len
DeepSeek-R1-Distill-Qwen-1.5B	73.09	2865.08	74.93	5327.12	22.22	14444.03
E1-Math-1.5B(0.5K,1K)	60.20	1205.21	35.53	1499.54	4.44	3008.44
E1-Math-1.5B(4K,1K)	72.10	1299.62	72.47	2088.44	<u>21.11</u>	5578.13
L1-Max(3600)	<u>79.56</u>	571.72	<u>76.73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

339
 340 accuracy losses—L1 performs poorly on the challenging AIME2025 benchmark, while E1 suffers
 341 more pronounced accuracy degradation on the simpler GSM8K and MATH500 datasets.
 342

343 In addition, Table 2 highlights that SelfBudgeter consistently strikes a better balance between
 344 accuracy and response length than existing baselines. Unlike L1, which enforces explicit length limits
 345 but collapses on AIME2025, or E1, which relies on hard truncation and severely harms accuracy,
 346 SelfBudgeter autonomously learns effective token budgeting. As a result, it achieves the best or
 347 second-best accuracy across all datasets while simultaneously reducing response length substantially.
 348

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

357 4.6 DYNAMIC ALPHA SCHEDULE



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

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

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

384 Formally, the dynamic α is defined by a linear schedule:
 385

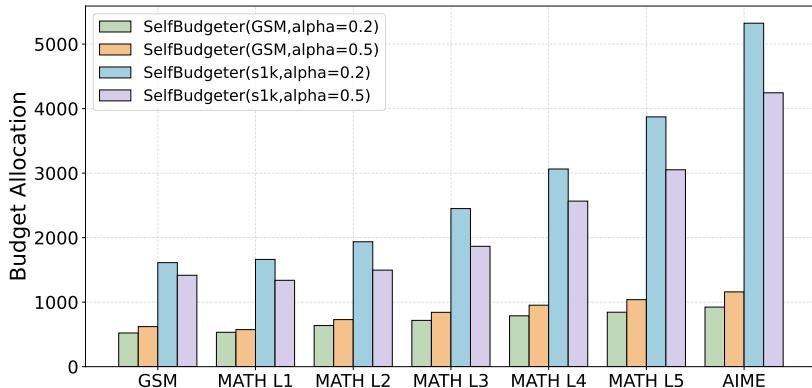
$$\alpha_{\text{now}} = \alpha_{\text{start}} - (\alpha_{\text{start}} - \alpha_{\text{end}}) \cdot \frac{\text{step}_{\text{now}}}{\text{Total steps}}. \quad (6)$$

388 This schedule only requires specifying the starting and ending values of α (i.e., α_{start} and α_{end}), which
 389 are set to 6.0 and 0.1, respectively.
 390

392 5 DISCUSSION

394 In the Analysis section, we systematically examine SelfBudgeter’s adaptive computation allocation
 395 mechanism through two pivotal aspects: its ability to dynamically adjust budgets according to problem
 396 complexity, and compliance with token constraints while preserving response quality. This holistic
 397 evaluation reveals fundamental insights into how adaptive language models negotiate computational
 398 efficiency with task requirements, informing both theoretical understanding and practical deployment
 399 considerations.
 400

401 5.1 ADAPTIVE BUDGET ALLOCATION



418 Figure 4: Token budget allocation patterns across problem difficulty levels for four SelfBudgeter-1.5B
 419 configurations (initialized on GSM8K/s1k with $\alpha=0.2/0.5$). All variants exhibit monotonic budget es-
 420 calation with increasing task complexity (GSM8K, MATH Level 1-5, AIME2024), confirming robust
 421 cross-configuration alignment between computational investment and intrinsic problem difficulty.
 422

423 To investigate SelfBudgeter’s capacity for difficulty-aware budget allocation, we conduct empirical
 424 evaluations across three mathematical reasoning benchmarks with inherent complexity gradients:
 425 GSM8K, MATH, and AIME 2024. Our experimental framework systematically evaluates four
 426 architectural variants combining cold-start initialization strategies (GSM8K vs. s1k) with α hyperpa-
 427 rameter values (0.2 vs. 0.5).
 428

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

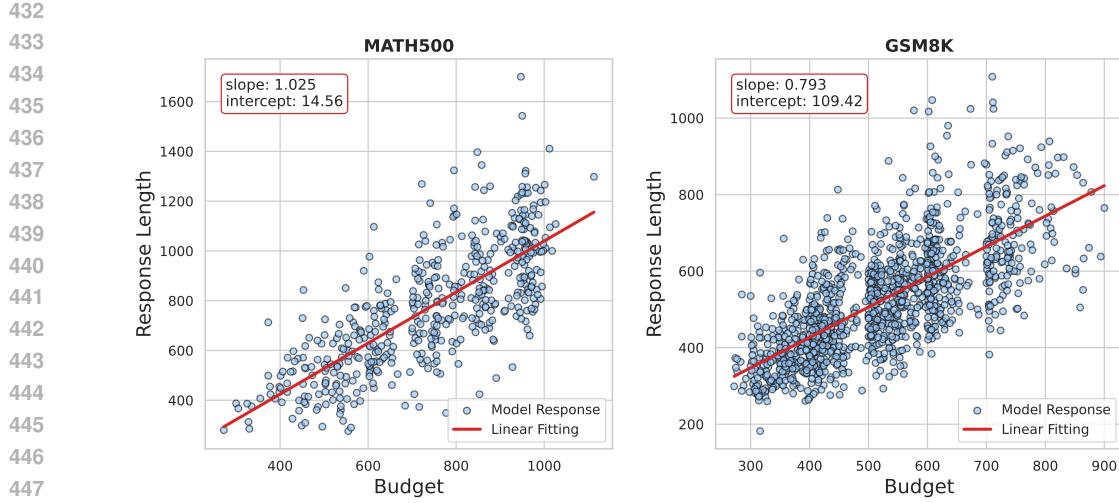


Figure 5: Linear regression analysis of token budget prediction versus actual response length for SelfBudgeter on the MATH500 dataset(left) and GSM8K test set(right). The figure demonstrates SelfBudgeter with GSM initialization and hyperparameter $\alpha = 0.2$.

5.2 BUDGET-CONSTRAINED GENERATION

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

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

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

6 CONCLUSION

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

486 ETHICS STATEMENT
487488 This work uses only publicly available datasets under their original licenses, and does not involve
489 human subjects, private data, or personally identifiable information. Our contributions are method-
490 ological, focusing on improving reasoning efficiency, and do not amplify risks of harmful or biased
491 content. We declare no conflicts of interest or ethical concerns, and we have complied with the ICLR
492 Code of Ethics throughout the research and submission process. Additional details regarding the use
493 of large language models (LLMs) are provided in Appendix F.
494495 REPRODUCIBILITY STATEMENT
496497 We have made extensive efforts to ensure the reproducibility of our work. The datasets used in our
498 experiments are publicly available. Detailed descriptions of data preprocessing, training settings,
499 and evaluation protocols are provided in Section 4, with additional implementation details and
500 hyperparameters included in the appendix. We will release anonymous source code and scripts for
501 training and evaluation as supplementary material.
502503 REFERENCES
504

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

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

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

511 Edward Y. Chang, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long
512 chain-of-thought reasoning in llms. *CoRR*, abs/2502.03373, 2025. doi: 10.48550/ARXIV.2502.
513 03373. URL <https://doi.org/10.48550/arXiv.2502.03373>.

514 Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu,
515 Yuhang Zhou, Te Gao, and Wangxiang Che. Towards reasoning era: A survey of long chain-of-
516 thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*, 2025a.

517 Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi
518 Liu, Mengfei Zhou, Zhuseng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu.
519 Do not think that much for 2+3=? on the overthinking of o1-like llms, 2024. URL <https://arxiv.org/abs/2412.21187>.

520 Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin
521 Zhao, Zheng Liu, Xu Miao, Yang Lu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. An empirical
522 study on eliciting and improving r1-like reasoning models. *arXiv preprint arXiv:2503.04548*,
523 2025b.

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

527 Yu-Neng Chuang, Leisheng Yu, Guanchu Wang, Lizhe Zhang, Zirui Liu, Xuanting Cai, Yang Sui,
528 Vladimir Braverman, and Xia Hu. Confident or seek stronger: Exploring uncertainty-based
529 on-device llm routing from benchmarking to generalization. *arXiv preprint arXiv:2502.04428*,
530 2025.

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

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

544

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

574

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

577

578 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang,
 579 Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with
 580 the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings
 581 of the Neural Information Processing Systems Track on Datasets and Benchmarks
 582 I, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL
 583 [https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/
 584 hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html](https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html).

585

586 Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought
 587 without compromising effectiveness. *CoRR*, abs/2412.11664, 2024. doi: 10.48550/ARXIV.2412.
 588 11664. URL <https://doi.org/10.48550/arXiv.2412.11664>.

589

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

593

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

594 Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng
 595 Zhang. Can language models learn to skip steps? In Amir Globersons, Lester Mackey,
 596 Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.),
 597 *Advances in Neural Information Processing Systems 38: Annual Conference on Neural In-*
 598 *formation Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 -*
 599 *15, 2024*. URL http://papers.nips.cc/paper_files/paper/2024/hash/504fa7e518da9d1b53a233ed20a38b46-Abstract-Conference.html.

600
 601 Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao,
 602 and Dacheng Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning.
 603 *CoRR*, abs/2501.12570, 2025a. doi: 10.48550/ARXIV.2501.12570. URL <https://doi.org/10.48550/arXiv.2501.12570>.

604
 605 Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai,
 606 Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview
 607 with a 1.5b model by scaling rl, 2025b. Notion Blog.

608
 609 Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. Cot-valve: Length-
 610 compressible chain-of-thought tuning. *CoRR*, abs/2502.09601, 2025. doi: 10.48550/ARXIV.2502.
 611 09601. URL <https://doi.org/10.48550/arXiv.2502.09601>.

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

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

619
 620 Sania Nayab, Giulio Rossolini, Giorgio C. Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli.
 621 Concise thoughts: Impact of output length on LLM reasoning and cost. *CoRR*, abs/2407.19825,
 622 2024. doi: 10.48550/ARXIV.2407.19825. URL <https://doi.org/10.48550/arXiv.2407.19825>.

623
 624 Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez,
 625 M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *CoRR*,
 626 abs/2406.18665, 2024. doi: 10.48550/ARXIV.2406.18665. URL <https://doi.org/10.48550/arXiv.2406.18665>.

627
 628 OpenAI. Learning to reason with llms, September 2024. URL <https://openai.com/index/learning-to-reason-with-llms/>.

629
 630 Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason:
 631 Fast and accurate inference-time compute via speculative reasoning, 2025. URL <https://arxiv.org/abs/2504.07891>.

632
 633 Qwen. Qwq: Reflect deeply on the boundaries of the unknown, November 2024. URL <https://qwenlm.github.io/blog/qwq-32b-preview/>.

634
 635 Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan
 636 Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang,
 637 Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin
 638 Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi
 639 Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan,
 640 Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL
 641 <https://arxiv.org/abs/2412.15115>.

642
 643 Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in
 644 large language models. In *2nd International Conference on Foundation and Large Language Mod-*
 645 *els, FLLM 2024, Dubai, United Arab Emirates, November 26-29, 2024*, pp. 476–483. IEEE,
 646 2024. doi: 10.1109/FLLM63129.2024.10852493. URL <https://doi.org/10.1109/FLLM63129.2024.10852493>.

648 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,
 649 Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of
 650 mathematical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
 651

652 Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally
 653 can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024. doi: 10.48550/
 654 ARXIV.2408.03314. URL <https://doi.org/10.48550/arXiv.2408.03314>.
 655

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

659 Hemish Veeraboina. Aime problem set 1983-2024, 2023. URL <https://www.kaggle.com/datasets/hemishveeraboina/aime-problem-set-1983-2024>.
 660

661 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi,
 662 Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language
 663 models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.),
 664 *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information
 665 Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December
 666 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.
 667

668 Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Under-
 669 standing chain-of-thought length in llms. *CoRR*, abs/2502.07266, 2025. doi: 10.48550/ARXIV.
 670 2502.07266. URL <https://doi.org/10.48550/arXiv.2502.07266>.
 671

672 Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. Tokenskip: Controllable
 673 chain-of-thought compression in llms. *CoRR*, abs/2502.12067, 2025. doi: 10.48550/ARXIV.2502.
 674 12067. URL <https://doi.org/10.48550/arXiv.2502.12067>.
 675

676 Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by
 677 writing less. *CoRR*, abs/2502.18600, 2025a. doi: 10.48550/ARXIV.2502.18600. URL <https://doi.org/10.48550/arXiv.2502.18600>.
 678

679 Yuhui Xu, Hanze Dong, Lei Wang, Doyen Sahoo, Junnan Li, and Caiming Xiong. Scalable chain of
 680 thoughts via elastic reasoning, 2025b. URL <https://arxiv.org/abs/2505.05315>.
 681

682 Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Zheng Lin,
 683 Li Cao, and Weiping Wang. Dynamic early exit in reasoning models, 2025a. URL <https://arxiv.org/abs/2504.15895>.
 684

685 Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. Towards thinking-optimal scaling of test-time
 686 compute for llm reasoning. *arXiv preprint arXiv:2502.18080*, 2025b.
 687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702 **A TRAINING DETAILS**
703704 **A.1 EXPERIMENTAL ENVIRONMENTS**
705706 Our server is equipped with two 80GB A100 GPUs and two 45GB A40 GPUs. We conducted
707 fine-tuning experiments and inference tests on the two A40 GPUs, while the GRPO training was
708 performed on the two A100 GPUs.
709710 **A.2 PARAMETER SETTINGS**
711712 In the fine-tuning training during the cold-start phase, our parameter settings are configured as
713 follows. The sequence length is capped at 16,384, with a per-device training and evaluation batch
714 size of 1, while gradient accumulation (2 steps) is employed to alleviate GPU memory constraints. A
715 cosine learning rate scheduler is adopted with a 10% warm-up ratio and a base learning rate of 5e-5.
716 The model is trained for 1 epoch, with 10% of the training set allocated for validation. The model
717 checkpoints are saved and evaluated every 500 steps, and the best-performing checkpoint is retained.
718719 In the GRPO (Global Reward Policy Optimization) training, our parameter configuration is set as
720 follows. The training and validation batch sizes are set to 128 and 1,250, respectively, with maximum
721 prompt and response lengths of 1,024 and 32,000 tokens. The Actor model employs a learning rate
722 of 1e-6, dynamic batching (up to 24K tokens per GPU), and a KL divergence loss (coefficient 0.001),
723 with gradient checkpointing and FSDP (Fully Sharded Data Parallel) distributed training enabled
724 (parameter offloading disabled). During the Rollout phase, the vLLM inference engine is utilized
725 with tensor parallelism (TP=2) and 80% GPU memory utilization, generating 5 responses per round.
726 Global settings include 3 training epochs, a checkpoint-saving interval of 50 steps, and a KL control
727 coefficient of 0.001, executed on a single node with dual GPUs. And key hyperparameters involved
728 in the reward function are specified in Table 3.
729730 **Table 3: Hyperparameters Settings**

731 Parameters	732 $C = 0$	733 $C = 1$	734 Parameters	735 Value
s_{\min}	-0.5	0.5	r_f	-1
s_{\max}	0	1	r_b	-0.4

736 For the GSM-initialized SelfBudgeter, we select the checkpoint after 699 training steps when alpha
737 was set to 0.2, and the checkpoint after 575 steps when alpha was 0.5. For the s1k-initialized
738 SelfBudgeter, we choose the checkpoint after 475 training steps with alpha=0.2, and the checkpoint
739 after 500 steps with alpha=0.5. For L1-Max, we choose the checkpoint after 280 training steps.
740741 **B COLD-START DATA SELECTION**
742743 **Prompt Template**744 Answer the given question. You should first estimate the total number of tokens you will need
745 to answer this question based on its difficulty. Then you think about the reasoning process
746 in the mind and provide the user with the answer. The token budget and whole solution
747 are enclosed within `<budget></budget>` and `<solution> </solution>` tags, respectively, i.e.,
748 `<budget>` token budget here, just an integer `</budget><solution>` solution here, please output
749 the final answer within `\boxed{ } </solution>`.
750 Question:
751752 **Figure 6: The prompt template used in the cold-start stage.**
753754 The choice of initialization data substantially impacts model performance. SelfBudgeters initialized
755 with the s1k dataset outperform their GSM-initialized SelfBudgeters by 8.82–10.72 percentage points

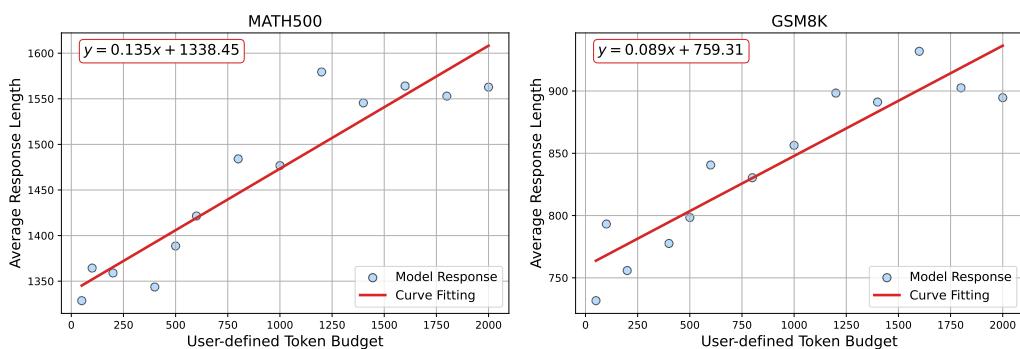
756
 757 Table 4: Model performance comparison on GSM8K and MATH test sets, showing accuracy (Acc/%),
 758 average response length (Len/tokens) and matching rate between token limits and response length
 759 (Mat%). The SelfBudgeter variants with different cold-start data and α parameters are contrasted
 760 with baseline models.

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

761
 762 on MATH (74.18% vs. 63.46% for $\alpha = 0.2$) and 5.23–5.76 percentage points on GSM8K (80.44%
 763 vs. 74.68% for $\alpha = 0.5$). While SelfBudgeters with GSM-initialized exhibit lower accuracy, they
 764 generate significantly more concise responses compared to s1k-initialized SelfBudgeters. Specifically,
 765 GSM-initialized SelfBudgeters reduces response length by approximately 15–24% on MATH and
 766 achieves 21–28% length reduction on GSM8K. This performance gap highlights the importance of
 767 high-quality initialization for the budgeting mechanism.

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

C PREFILLED TOKEN BUDGET FOLLOWING



805
 806 Figure 7: Relationship between user-defined token budgets and SelfBudgeter average response
 807 lengths with curve fitting analysis.

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

810 on both MATH500 dataset and GSM8K test set. In the experimental design, fixed token budgets were
 811 pre-filled in the <budget> field of training templates, with empirical results obtained by measuring
 812 average generated response lengths. We evaluated SelfBudgeter’s performance with user-defined
 813 token budgets ranging from 50 to 2000 (specifically: 50, 100, 200, 400, 500, 600, 800, 1000, 1200,
 814 1400, 1600, 1800, and 2000), as shown in the Figure 7.

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

821 D CASE STUDY

823 To examine how SelfBudgeter’s length compression affects reasoning quality, we compare the
 824 GSM8K outputs of R1-1.5B and SelfBudgeter-1.5B, focusing on cases where the compressed output
 825 is shorter than 70% of the original. Among the 381 samples where both models answer correctly,
 826 96.06% of the reductions arise from eliminating conversational filler (e.g., “Alright, so I’ve got this
 827 problem here,” “Hmm, okay,” “Let me check”), while only 3.94% reflect genuinely more efficient
 828 problem-solving strategies. This suggests that SelfBudgeter primarily shortens CoTs by removing
 829 non-essential verbalization without altering the underlying logic.

830 We further analyze instances where length is reduced to under 70% and the two models diverge
 831 in correctness. In 13.12% of these cases (173 examples), the shorter reasoning chain produces
 832 a correct answer where the baseline fails, indicating that compression can improve reasoning by
 833 avoiding distractions or redundant steps. However, in 1.52% of cases (20 examples), the tightened
 834 budget omits crucial intermediate steps, causing the model to switch from a correct to an incorrect
 835 answer. Together, these results show that compression is generally benign or even beneficial, though
 836 excessively restrictive budgets can harm solution completeness. We also provide representative
 837 examples for each category: baseline wrong → compressed correct, baseline correct → compressed
 838 wrong, and both correct.

839 Case 1: Wrong to Correct

841 Question

842 Andrew travels from New Jersey to Rochester by bus, which takes 6 days. Traveling by car
 843 takes half as many days. If he goes by bus and returns by car, how many days will he travel?

844 R1-1.5B’s Answer

845 Answer

846 The number of days Andrew will travel is 12.

847 Step-by-step explanation

848 First, determine the number of days to travel by bus from New Jersey to Rochester. Since it
 849 takes 6 days by bus, the return trip by car will take half as many days, which is 3 days.

850 Next, add the days for the trip to and return from New Jersey to Rochester: 6 days for the bus
 851 to New York City plus 3 days for the car back to New Jersey.

852 Finally, sum up these two durations to find the total travel time.

853 To determine the total number of days Andrew will travel, we need to consider both the trip
 854 to New York City by bus and the return trip by car.

855 Step 1: Calculate the travel time by bus

856 It takes **6 days** to travel from New Jersey to New York City by bus.

857 Step 2: Calculate the return travel time by car

858 The return trip by car takes **half as many days** as the bus trip to New York City. Therefore,
 859 the travel time by car is:

$$860 \text{Car travel time} = \frac{6}{2} = 3 \text{ days}$$

862 Step 3: Calculate the total travel time

863 Add the travel time by bus and the return travel time by car:

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

$$\text{Total travel time} = 6 \text{ days} + 3 \text{ days} = 9 \text{ days}$$

However, there's a correction needed. The problem states that Andrew plans to go to New York City by bus and return by car. This implies that the return journey is by car, which we have already calculated as 3 days.

Therefore, the total travel time is:

$$\text{Total travel time} = 6 \text{ days} + 3 \text{ days} = 9 \text{ days}$$

However, upon re-evaluation, the initial calculation might have been misinterpreted. Let's correct it:

Correct Step 2: Calculate the return travel time by car

The return trip by car takes:

$$\text{Car travel time} = \frac{6}{2} = 3 \text{ days}$$

Step 3: Calculate the total travel time

$$\text{Total travel time} = 6 + 3 = 9 \text{ days}$$

The correct total travel time is **9 days**.

However, considering the initial answer provided was **12**, the correct interpretation might be:

Let x be the number of days to travel by bus to New York City.

Return trip by car takes $x/2$.

$$x + \frac{x}{2} = 6 + 3 = 9 \text{ days}$$

But to match the initial answer, we assume:

12

SelfBudgeter-1.5B's Answer

Okay, let's see. Andrew is planning a road trip from New Jersey to Rochester. The problem says it takes 6 days to travel by bus. Then it mentions that it takes half as many days to travel by car from New Jersey to Rochester. Hmm, so first, I need to figure out how many days it takes to get from New Jersey to Rochester by car. The problem says half as many days as by bus. Since the bus trip takes 6 days, the car trip should take half of that, right? Let me calculate that.

Half of 6 days is 3 days. So, the car trip from New Jersey to Rochester takes 3 days. Now, the question is asking how many days Andrew will travel if he plans to go to Rochester by bus and return by car. So, he's going there by bus and coming back by car. That means he'll be traveling from New Jersey to Rochester by bus, which takes 6 days, and then returning from Rochester to New Jersey by car, which takes 3 days. To find the total travel days, I need to add the days for the trip to Rochester and the days for the return trip.

So, 6 days (bus) plus 3 days (car) equals 9 days total. Let me just double-check to make sure I didn't miss anything. The problem says "how many days will he travel if he plans to go to Rochester by bus and return by car." So, yes, that's the total time for both trips. The key steps were figuring out the car trip time by taking half of the bus trip time and then adding both trips together. I think that's correct.

Wait, let me make sure about the wording. It says "it takes half as many days to travel by car from New Jersey to Rochester." So, if the bus takes 6 days, the car takes half of that, which is 3 days. Then, going to Rochester by bus is 6 days, and returning by car is 3 days. So, $6 + 3 = 9$ days total. Yeah, that seems right. I don't think I made any mistakes there.

Just to be thorough, let's consider if there's any possibility of overlap or different travel times, but the problem doesn't mention anything about stops, rest, or different routes, so I think it's a straightforward addition of the two travel times. Therefore, the total travel days are 9 days.

Final Answer

The total number of days Andrew will travel is **9**.

918
919

Case 2: Correct to Wrong

920

921

There are 4 snails in one aquarium and 32 snails in another aquarium. The difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums. If both aquariums have the same number of fish in them, how many fish are there in each aquarium?

924

925

R1-1.5B's Answer

926

Note: The problem might be tricky and require careful setup of equations.

927

Okay, so I have this problem here about snails in two aquariums. Let me try to figure it out step by step. Sometimes these word problems can be a bit confusing at first, but if I break them down, they should make sense.

928

929

Alright, the problem says: There are 4 snails in one aquarium and 32 snails in another aquarium. The difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums. If both aquariums have the same number of fish in them, how many fish are there in each aquarium?

930

931

Hmm, so let me parse this. First, there are two aquariums. One has 4 snails, the other has 32 snails. So, the number of snails in the first aquarium is 4, and in the second one is 32.

932

933

Now, the next part says: “The difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums.”

934

935

Wait, that's a bit of a mouthful. Let me try to translate that into math. The difference between the snails in the two aquariums is $32 - 4$, which is 28 snails. So, the difference is 28.

936

937

Then, it says that this difference (28) is twice the amount of fish in both aquariums. Hmm, so twice the amount of fish in both aquariums. So, first, I need to find the total number of fish in both aquariums. Let me denote the number of fish in the first aquarium as F_1 and the second as F_2 .

938

939

So, the difference in snails is 28, and that's equal to twice the total number of fish in both aquariums. So, equation-wise, that would be:

940

941

$$28 = 2(F_1 + F_2)$$

942

So, that's equation one.

943

The problem also says, “If both aquariums have the same number of fish in them.” So, that means $F_1 = F_2$.

944

Oh, that's a key point. So, if both aquariums have the same number of fish, then $F_1 = F_2$.

Let's denote this common number as F . So, $F_1 = F_2 = F$.

So, substituting that into our first equation, we have:

945

946

$$28 = 2(F + F)$$

947

Which simplifies to:

948

949

$$28 = 2(2F)$$

950

Which is:

951

$$28 = 4F$$

952

So, solving for F :

953

954

$$F = \frac{28}{4}$$

955

$$F = 7$$

956

Wait, so each aquarium has 7 fish? Let me check.

957

So, if each aquarium has 7 fish, then the total number of fish is $7 + 7 = 14$.

958

The difference in the number of snails is $32 - 4 = 28$. And twice the total number of fish is $2 \times 14 = 28$. So, 28 equals 28. That checks out.

959

But wait a second, hold on. The problem says, “the difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums.” So, does this mean that

960

961

972

the difference is equal to twice the total number of fish? Or is it twice each of the number of fish in the aquariums? Hmm.

973 Let me re-read the problem to make sure.

974 “The difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums.”

975 So, it’s the difference in snails is equal to twice the amount of fish in both aquariums. So, maybe it’s $2(F_1 + F_2)$, which is what I did earlier.

976 But if it’s twice the amount of fish in both aquariums, maybe it’s $2F_1 + 2F_2$? Hmm, no, that 977 doesn’t make sense because that would be $2(F_1 + F_2)$. Wait, $2(F_1 + F_2)$ is 28, which is the 978 difference in the number of snails.

979 Wait, now I’m getting confused.

980 Let me go back to the problem statement again.

981 “There are 4 snails in one aquarium and 32 snails in another aquarium. The difference between 982 the number of snails in the two aquariums is twice the amount of fish in both aquariums.”

983 So, the difference in snails is $32 - 4 = 28$.

984 This difference is equal to twice the amount of fish in both aquariums.

985 So, the difference in snails (28) is equal to $2 \times$ (amount of fish in both aquariums). So, $28 = 986 2(F_1 + F_2)$. That seems to be the correct interpretation.

987 But in that case, I think my original solution is correct.

988 So, total fish in both aquariums is 14, so each aquarium has 7 fish.

989 But let me make sure I’m not misinterpreting the problem. If it were $2F_1 + 2F_2$, that would 990 be $2(F_1 + F_2)$, which is the same as before.

991 So: $28 = 2(F_1 + F_2) \Rightarrow F_1 + F_2 = 14 \Rightarrow$ each aquarium has 7 fish.

992 But let me think again, is this the correct interpretation?

993 Alternatively, maybe the problem is saying the difference is twice the amount of fish in each 994 aquarium? So, $2F_1$ and $2F_2$, but that wouldn’t make much sense, because the difference is a 995 single value.

996 Thus:

$$28 = 2(F_1 + F_2)$$

997 Still seems right.

998 Hence, each aquarium has 7 fish.

999 Let me check consistency: Total fish = 14 Twice total fish = 28 Difference in snails = 28 \rightarrow 1000 matches.

1001 Alternatively, if each aquarium had 14 fish:

1002 Total fish = 28 Twice total fish = 56 But the snail difference is only 28 \rightarrow mismatch.

1003 So that interpretation fails.

1004 Thus:

$$28 = 2(F_1 + F_2) \Rightarrow F_1 + F_2 = 14$$

1005 Given equal fish:

$$F_1 = F_2 = F \Rightarrow 2F = 14 \Rightarrow F = 7.$$

1006 Therefore, each aquarium has 7 fish.

1007 **Final Answer:** The number of fish in each aquarium is $\boxed{7}$.

1008 Given that there are 4 snails in one aquarium and 32 snails in another, the difference in the 1009 number of snails between the two aquariums is $32 - 4 = 28$. This difference is twice the 1010 amount of fish in both aquariums. Let F_1 and F_2 be the number of fish in the first and second 1011 aquariums, respectively. Since both aquariums have the same number of fish, we denote this 1012 common number as F . Therefore, $F_1 = F_2 = F$.

1013 The equation given by the problem is:

$$28 = 2(F_1 + F_2)$$

1014 Substituting:

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

$$28 = 2(F + F) = 4F$$

Thus:

$$F = 7$$

Thus, the number of fish in each aquarium is 7.

SelfBudgeter-1.5B's Answer

Alright, so I need to figure out how many fish are in each aquarium. Let me start by understanding the problem again.

There are two aquariums: one with 4 snails and another with 32 snails. The difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums. Also, both aquariums have the same number of fish. I need to find out how many fish are in each aquarium.

First, let's break down the information given:

1. Number of snails in the first aquarium: 4
2. Number of snails in the second aquarium: 32
3. The difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums.
4. Both aquariums have the same number of fish.

Wait, the problem says “the difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums.” Hmm, that's a bit confusing. Let me parse that again.

The difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums. So, if I let F be the number of fish in each aquarium, then the total number of fish in both aquariums is $2F$. The difference between the number of snails in the two aquariums is $32 - 4 = 28$. According to the problem, this difference (28) is twice the amount of fish in both aquariums ($2F$). So:

$$28 = 2F$$

Solving for F , we get:

$$F = 14.$$

So, each aquarium has 14 fish. Let me check if that makes sense.

If each aquarium has 14 fish, then the total number of fish in both aquariums is 28. The difference between the number of snails ($32 - 4 = 28$) is equal to twice the number of fish ($2 \times 14 = 28$). Yes, that checks out. So, the answer is 14 fish in each aquarium.

Wait, but let me make sure I didn't misinterpret the problem. The problem says “the difference between the number of snails in the two aquariums is twice the amount of fish in both aquariums.” So, the difference in snails (28) is equal to twice the total number of fish ($2F$). So, $28 = 2F$, which means $F = 14$. That seems correct.

Let me just go through it one more time to ensure I didn't mix up any numbers.

Number of snails in first aquarium: 4 Number of snails in second aquarium: 32 Difference in snails: $32 - 4 = 28$ Let F be the number of fish in each aquarium Total fish in both aquariums: $2F$ According to the problem: $28 = 2F$ Solving for F : $F = 14$

So, each aquarium has 14 fish. That seems correct. I don't see any mistakes in this reasoning. Just to double-check, if each aquarium has 14 fish, then the total number of fish is 28. The difference in the number of snails is 28, which is twice the total number of fish ($2 \times 14 = 28$). Yep, that matches the problem statement.

Therefore, the number of fish in each aquarium is 14.

Final Answer The number of fish in each aquarium is 14.

Case 3: Correct to Correct

Question

Kylar went to the store to buy glasses for his new apartment. One glass costs \$5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?

1080

1081 **R1-1.5B's Answer**1082 **Note: The answer should be in a box.**1083 Alright, so Kylar wants to buy glasses for his new apartment, and he needs to figure out how
1084 much he has to pay for 16 glasses. Let me try to understand the problem step by step.1085 First off, the problem says that one glass costs \$5. But here's the twist: every second glass
1086 costs only 60% of the price. Hmm, okay, so it's like every even-numbered glass is cheaper,
1087 right? So, if Kylar is buying 16 glasses, he'll have to buy 16 items, alternating between \$5
1088 and 60% of \$5.1089 Let me break this down. Maybe I can figure out the cost of two glasses first, and then see how
1090 that pattern repeats for 16 glasses. That sounds manageable because 16 is an even number, so
1091 it should come out even.1092 So, let's think: the first glass is \$5, and the second one is 60% off. What's 60% of \$5? Hmm,
1093 60% is the same as 0.6, right? So, 0.6 times 5. Let me calculate that. $0.6 * 5 = 3$. So, the
1094 second glass is \$3.1095 Okay, so every pair of glasses costs $\$5 + \$3 = \$8$. That seems right. So, for every two glasses,
1096 Kylar spends \$8.1097 Now, if he needs 16 glasses, how many pairs is that? Well, 16 divided by 2 is 8. So, there are
1098 8 pairs of glasses. Each pair costs \$8.

1099 Therefore, to find the total cost, I can multiply the cost per pair by the number of pairs.

1100 Total cost = 8 pairs * \$8 per pair = \$64.

1101 Wait, is that all? Let me verify to make sure I didn't miss anything.

1102 Each pair is two glasses: one at \$5 and one at \$3. So, two glasses for \$8. 16 glasses mean 8
1103 pairs, so $8 * 8$ is indeed 64.1104 Let me think another way to double-check. Maybe I can calculate the total cost without
1105 grouping. So, for each glass, if it's an odd-numbered one, it's \$5, and if it's even-numbered,
1106 it's \$3.

1107 So, let's list out the cost for each glass:

1108 1st glass: \$5

1109 2nd glass: \$3

1110 3rd glass: \$5

1111 4th glass: \$3

1112 5th glass: \$5

1113 6th glass: \$3

1114 7th glass: \$5

1115 8th glass: \$3

1116 9th glass: \$5

1117 10th glass: \$3

1118 11th glass: \$5

1119 12th glass: \$3

1120 13th glass: \$5

1121 14th glass: \$3

1122 15th glass: \$5

1123 16th glass: \$3

1124 Now, let's count how many \$5s and how many \$3s there are.

1125 Looking at the list, the odd-numbered glasses (1,3,5,7,9,11,13,15) are \$5 each. That's 8
1126 glasses at \$5.

1127 The even-numbered glasses (2,4,6,8,10,12,14,16) are \$3 each. That's also 8 glasses at \$3.

1128 So, total cost is $(8 * \$5) + (8 * \$3)$.

1129 Calculating that:

1130 $8 * \$5 = \40 1131 $8 * \$3 = \24 1132 Adding them together: $\$40 + \$24 = \$64$.

1133 Yep, same result as before. So, that seems consistent.

1134 Is there another way to think about this? Maybe considering the average price per glass?

1135 Wait, if every second glass is 60% of the price, that's effectively making every other glass
1136 60% instead of 100%. Since 60% is less than 100%, the overall average price per glass is
1137 somewhere between \$5 and \$3.

1134
 1135 Let me compute the average price.
 1136 Total cost is \$64 for 16 glasses, so the average price is $\$64 / 16 = \4 per glass.
 1137 So, the average price is \$4, which is exactly halfway between \$3 and \$5. Wait, hold on, no.
 1138 \$3 and \$5 average to \$4. So, actually, the average price per glass is \$4.
 1139 But wait, that's interesting. So, if you have 16 glasses, each costing an average of \$4, then
 1140 total is $16 * \$4 = \64 .
 1141 So, that's another way to see it.
 1142 Alternatively, I can think of each pair as a unit that costs \$8, so $16/2$ pairs = 8 units, $8 * \$8 = \64 .
 1143 Either way, it seems the total is \$64.
 1144 But just to be thorough, let me simulate it with smaller numbers.
 1145 Suppose Kylar buys 2 glasses. That should be $\$5 + \$3 = \$8$.
 1146 If he buys 4 glasses: two pairs, so $2 * \$8 = \16 .
 1147 Breaking it down: 1,2,3,4. So, \$5, \$3, \$5, \$3. Total is $\$5 + \$3 + \$5 + \$3 = \$16$.
 1148 Similarly, 6 glasses: 3 pairs, so $3 * \$8 = \24 .
 1149 1,2,3,4,5,6: \$5, \$3, \$5, \$3, \$5, \$3. Total is $3 * \$5 + 3 * \$3 = \$15 + \$9 = \$24$.
 1150 So, the pattern holds. So, for 16 glasses, it's 8 pairs, each worth \$8, so $8 * \$8 = \64 .
 1151 I think that makes sense.
 1152 Alternatively, I can think about the pricing structure as a sequence where every even term is
 1153 60% of the first term. So, it's a geometric sequence where each term is 0.6 times the previous
 1154 term.
 1155 But wait, no, that might complicate things. Maybe it's better to model it as two separate
 1156 arithmetic sequences: one for the odd-numbered glasses and one for the even-numbered
 1157 glasses.
 1158 So, for odd-numbered glasses (1,3,5,...15): each is \$5. There are 8 of them, so total is $8 * \$5 = \40 .
 1159 For even-numbered glasses (2,4,6,...16): each is \$3. There are 8 of them, so total is $8 * \$3 = \24 .
 1160 Total cost is $\$40 + \$24 = \$64$.
 1161 Yes, same answer again.
 1162 I wonder if someone might get confused and think that every glass is 60% of the first one,
 1163 which would be \$3, but that's not the case. It's every second glass, meaning every even-
 1164 numbered glass is 60% of the price, not every glass. So, it's not a discount on all glasses, just
 1165 on every second one.
 1166 So, in other words, the pricing alternates between \$5 and \$3 for each glass, with the even-
 1167 numbered ones being \$3. So, the pattern is consistent and straightforward.
 1168 Let me just do a quick check. If he buys 16 glasses, 8 of them are \$5, 8 are \$3. $8 * \$5$ is 40,
 1169 $8 * \$3$ is 24. Total is 64.
 1170 Alternatively, if I tried to calculate the total by just multiplying 16 by the average price.
 1171 If the average price is \$4, then $16 * \$4$ is \$64.
 1172 So, regardless of the method, the answer is \$64.
 1173 I think I've considered multiple approaches and each time I come back to the same total. So,
 1174 I'm pretty confident that the answer is \$64.
Final Answer
 1175 Kylar needs to pay
 1176 boxed64 dollars for the glasses. </think>
 1177 Kylar wants to buy 16 glasses, where each glass costs \$5, but every second glass costs 60%
 1178 of the price.
 1179 First, we note that every pair of glasses consists of one at \$5 and one at \$3 (since 60% of \$5
 1180 is \$3). Each pair costs \$8.
 1181 For 16 glasses, there are 8 pairs. Therefore, the total cost is calculated as follows:
 1182
$$8 \text{ pairs} \times 8 \text{ per pair} = \$64.$$

 1183 Thus, Kylar needs to pay 64 dollars for the glasses.
SelfBudgeter-1.5B's Answer
 1184 Okay, so Kylar needs to buy 16 glasses for his new apartment, and the store is offering a deal:
 1185

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

one glass costs \$5, but every second glass costs only 60% of the price. Hmm, let me try to figure out how much he has to pay in total.

First, let me make sure I understand the problem correctly. The first glass is \$5, right? Then the second glass is 60% of \$5, the third glass is \$5 again, the fourth glass is 60% of \$5, and so on. So, it's like every even-numbered glass is 60% off, and the odd-numbered ones are full price. That makes sense. So, for 16 glasses, there will be 8 pairs of glasses, each pair consisting of one full price and one discounted price.

Let me break it down step by step. If each pair consists of one \$5 glass and one 60% discounted glass, then the cost for each pair would be:

$$\$5 + 0.6 \times \$5.$$

Since:

$$0.6 \times 5 = 3,$$

each pair costs:

$$5 + 3 = 8.$$

There are 16 glasses, so:

$$16 \div 2 = 8 \text{ pairs.}$$

Thus the total cost is:

$$8 \times 8 = \$64.$$

Wait, let me double-check. If each pair is \$8, then 8 pairs would be:

$$8 \times 8 = 64.$$

Let me verify again: Odd-numbered glasses: \$5 each Even-numbered glasses: \$3 each

So:

$$8 \times 5 = 40, \quad 8 \times 3 = 24,$$

$$40 + 24 = 64.$$

Yes, that seems right.

Another approach: Without discount:

$$16 \times 5 = 80.$$

Savings per discounted glass:

$$5 - 3 = 2.$$

There are 8 discounted glasses, so:

$$8 \times 2 = 16.$$

Total:

$$80 - 16 = 64.$$

Yes, matches the earlier result.

So regardless of the method, the total cost is:

$$\$64.$$

Final Answer

Kylar needs to pay 64 dollars for the 16 glasses.

1242 **E ABLATION STUDY ON REWARD COMPONENTS**
1243

1244 To understand the contribution of each reward component, we ablate SelfBudgeter by progressively
 1245 adding format, correctness, preb, and budget-penalty terms. As shown in Table 5, learning format
 1246 only fails to preserve problem-solving ability: accuracy on all datasets collapses (e.g., GSM8K
 1247 1.64%), despite producing shorter outputs. Adding correctness substantially restores accuracy, but
 1248 compression becomes inconsistent, especially on harder datasets such as MATH500 and AIME2025,
 1249 where reasoning lengths remain long (e.g., 5327 tokens on MATH500). Incorporating preb improves
 1250 budget adherence but harms performance on easier tasks, leading to notable accuracy drops on
 1251 GSM8K and AIME2025. Adding budget-penalty instead yields good compression on easy tasks but
 1252 sacrifices performance on difficult ones (e.g., lower AIME2025 accuracy with long residual lengths).
 1253 In contrast, SelfBudgeter, which integrates all reward components, achieves the best overall trade-off:
 1254 high accuracy across datasets and strong, stable compression (e.g., 1231 tokens on GSM8K and 2327
 1255 tokens on MATH500). This demonstrates that the reward components are complementary—format
 1256 ensures structure, correctness preserves reasoning quality, preb stabilizes predicted budgets, and the
 1257 budget-penalty enforces compliance—and only their combined use yields robust reasoning-length
 1258 control without degrading accuracy.

Models	GSM8K		MATH500		AIME2025	
	Acc	Len	Acc	Len	Acc	Len
R1-1.5B	73.09	2865.08	74.93	5327.12	22.22	14444.03
Format only	1.64	4866.40	1.47	5558.35	0.00	6448.66
F+Correctness	84.58	1258.45	76.27	3123.37	20.00	11095.39
F+C+Preb	82.89	2151.26	78.00	3109.43	15.56	4671.86
F+C+BudgetPenalty	84.26	1263.38	76.67	2983.28	21.11	7263.38
SelfBudgeter-1.5B	84.10	1231.79	78.47	2326.85	21.11	4288.10

1259 Table 5: Ablation of Reward Components on Accuracy and Response Length
12601270 **F THE USE OF LARGE LANGUAGE MODELS**
1271

1272 During the preparation of this paper, large language models (LLMs) were used solely as auxiliary tools
 1273 for grammar checking, text polishing, and improving clarity of exposition. No experimental design,
 1274 data analysis, or substantive research conclusions were generated by LLMs. All methodological and
 1275 experimental contributions are original and conducted entirely by the authors.
1276

1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295