BUDGET INTERNALIZATION: FIXED-BUDGET RLVR TRADES TEST-TIME SCALING FOR IN-BUDGET ACCURACY

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning models often become more accurate when they can generate longer answers at test time. This behavior is known as test-time scaling. However, RL with verifiable rewards (RLVR) usually trains them with a fixed rollout cap. We find that this cap changes model behavior, not only training cost. After training, models use few additional tokens beyond the training budget. They outperform the base models near that budget, but their accuracy stops improving and falls below the base model at larger budgets. The rollout cap thus becomes a learned constraint, that limits model's test-time scaling. We call this effect *budget internalization*. We observe it across Qwen3.5-4B, Gemma4-E2B, and Nemotron-3-Nano-4B and two reasoning training tasks: mathematical reasoning and diverse, procedurally generated problems. This effect also transfers from math training to science and coding tasks. Training-objective level ablations on Qwen3.5-4B show that most changes recover little test-time scaling. Stronger KL regularization and lower training-data truncation preserve more scaling, but stronger KL reduces in-budget accuracy. Forcing an answer on truncated training rollouts largely removes budget internalization, but also removes most of the in-budget gain. SFT on artificially truncated traces reproduces budget internalization, whereas SFT on naturally short traces does not, which shows that the effect is not specific to RL. This result links budget internalization to length-censored post-training rather than to an RL-specific objective term. Thus, a rollout cap does not only control training cost. It can trade low-budget gains for lower performance when additional inference compute is available.

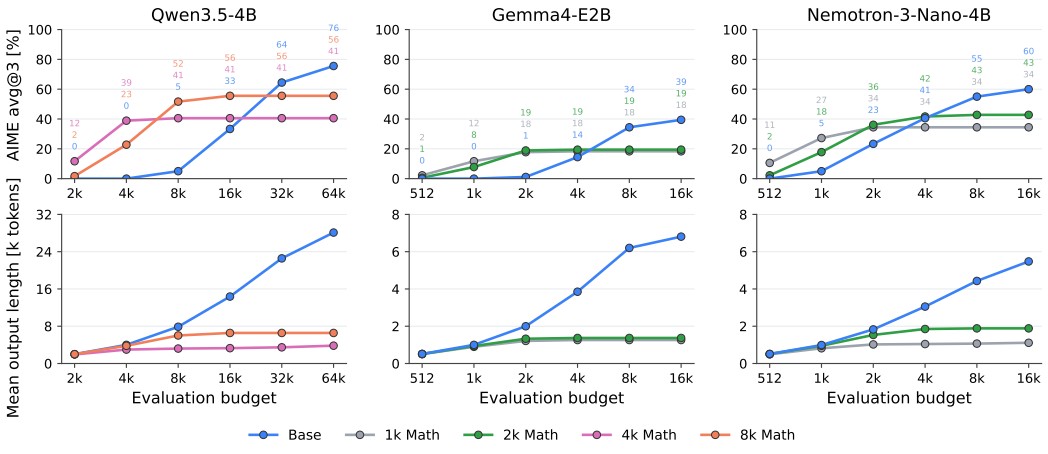


Figure 1: *Budget internalization* after fixed-budget math RLVR. Checkpoints improve AIME near the training cap, but their response length and accuracy saturate before those of their base models. Appendix Table 8 reports the exact values.

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.

1 INTRODUCTION

One promise of reasoning models is that they can solve harder problems when given more tokens at test time (Guo et al., 2025; Snell et al., 2025). This makes test-time compute a useful scaling axis: a deployment can spend more compute when a problem is difficult or when higher accuracy is required.

Unlike test-time scaling, RL with verifiable rewards (RLVR) usually does not let the model reason until it emits an end-of-sequence token. Instead, training uses a fixed rollout cap, often below the length of the model could otherwise utilize(Team et al., 2025; Luo et al., 2025; He et al., 2025). This is usually done to cut training costs, since it can scale quadratically with the sequence length (Aghajohari et al., 2026). The resulting model is then often evaluated at only one budget, at or close to the training cap. This evaluation shows whether training helps near that budget, but not whether the model can still benefit from more compute. A model can look better than its base model at the training budget and become worse when both receive a larger budget.

We show that this failure occurs after fixed-budget GRPO on two hard reasoning training tasks: competition mathematics (AIME, DeepScaler Luo et al. (2025)) and diverse procedurally generated reasoning problems (Reasoning Gym Stojanovski et al. (2025)). For Qwen3.5-4B trained with a 4k rollout cap, the model moves from a large gain over the base model at the matching evaluation budget to a large deficit when given more tokens (4K: $+38.9$ pp; 64K: -35.0 pp). Gemma4-E2B (Gemma Team, 2026) and Nemotron-3-Nano-4B (NVIDIA, 2026) show the same saturation after math training, and Reasoning Gym training reproduces it for all three models. A similar pattern also transfers from math training to GPQA-Diamond (Rein et al., 2024) and LiveCodeBench (Jain et al., 2025). We call this effect *budget internalization*: the training budget becomes part of the model's learned behavior rather than remaining only a limit on training cost.

Budget internalization changes how RLVR results should be interpreted. Accuracy measured only at the training cap can hide a reversal in model ranking. Increasing the deployment budget may then fail to recover the base model's performance, even though the base model benefits from the same additional compute. The training cap can therefore trade future test-time scaling for better performance at the budget used during training.

Our goal is to investigate budget internalization and understand why it occurs. To do this, we measure accuracy and output-length curves across model families, training caps, and reasoning benchmarks. We then run training ablations to study which training choices affect above-budget scaling. This analysis makes three contributions:

- **We show that fixed-budget training hurts test-time scaling.** Across Qwen, Gemma, and Nemotron, AIME accuracy improves near the training cap but stops improving at larger budgets. Across all six checkpoints, gains over the base model range from $+11.7$ to $+46.7$ pp at the training cap, while deficits at the largest evaluation budget shown for each model range from -17.2 to -35.0 pp.
- **The effects of fixed-budget RLVR transfer beyond the training task and can improve efficiency.** For Qwen's 8k-trained checkpoint, the high-budget differences from the base model are much smaller (GPQA, 64K: -2.2 pp; LCB-v6, 64K: -1.0 pp). At the same time, mean output length falls compared to the base Qwen3.5-4B from 11.15k to 4.56k tokens on GPQA and from 38.32k to 8.31k on LiveCodeBench. Gemma and Nemotron show the same pattern of much shorter responses with smaller accuracy differences on these tasks. Therefore, losing some test-time scaling is not always harmful and can reduce inference cost while retaining similar accuracy.
- **Our ablations reveal a persistent trade-off between in-budget gains and test-time scaling.** We examine objective variants, truncation handling, KL regularization, and SFT approaches. Scaling remains impaired under alternative length normalization and overlength penalties, but improves when truncation exposure is reduced or KL anchoring is strengthened. The scoring of truncated rollouts strongly affects budget internalization, suggesting that the responses reinforced during training matter more than any single GRPO objective

We release all our trained checkpoints (including ablations) at `https://huggingface.co/budget-internalization-iclr2027`.

term. SFT experiments support this interpretation: scaling is retained with naturally completed trajectories, but deteriorates with artificially shortened ones. Despite these partial mitigations, no tested intervention combines the full in-budget improvement with the base model's high-budget performance.

Budget internalization is therefore neither always good nor always bad. It is harmful when a task requires longer reasoning to solve harder problems, but useful when the same accuracy can be reached with fewer tokens on tasks where the model tends to overthink. Evaluating only at the training cap cannot distinguish these cases. We therefore recommend reporting accuracy and output length at several budgets, so that the effect of fixed-budget training is clear before choosing a model for deployment.

2 How Fixed-Budget Training Changes Test-Time Scaling

We study Qwen3.5-4B, Gemma4-E2B, and NVIDIA-Nemotron-3-Nano-4B. We run GRPO on the DeepScaleR math dataset with 4k and 8k rollouts cap for Qwen, and 1k and 2k caps for Gemma and Nemotron. We additionally train Qwen with 4k cap and Gemma and Nemotron with 2k caps on Nemotron-RL-ReasoningGym-v1 (NVIDIA Corporation, 2026), which contains 15,000 examples from 104 procedurally generated and verifiable Reasoning Gym environments (Stojanovski et al., 2025). These runs use the same training hyperparameters as the corresponding DeepScaleR runs. We choose these caps below the budgets at which the base models usually complete their AIME responses, so base-model generations are frequently truncated at every chosen cap; Appendix Table 12 reports the exact rates. We evaluate the selected checkpoints on AIME 2024 and 2025. We then test whether the resulting behavior transfers to out-of-domain benchmarks, GPQA-Diamond and LiveCodeBench v6. All runs use the NeMo RL framework (NeMo RL); the reward verifies the final answer and assigns zero reward to rollouts that end at the cap without one.

We run all evaluations with the Inspect AI framework (UK AI Security Institute, 2024) and report avg@k, the accuracy averaged over k sampled generations per problem, with the evaluation budget as the generated-token limit. We select checkpoints by validation accuracy at their training budget, and all runs share the same training seed. Appendix A reports the complete training and evaluation configuration.

2.1 Fixed-Budget Training Limits Test-Time Scaling

Figure 1 shows a consistent pattern across the three model families. All accuracy deltas in this section are relative to the corresponding base model at the same evaluation budget. Training gives large gains where the base model frequently reaches the evaluation cap, but these gains quickly saturate once the evaluation budget exceeds the training budget. Both Qwen checkpoints show this reversal: the 4k-trained checkpoint moves from 4K: +38.9 pp to 64K: −35.0 pp, while the 8k-trained checkpoint moves from 8K: +46.7 pp to 64K: −20.0 pp. The same pattern holds for both Gemma checkpoints: the 1k-trained checkpoint moves from 1K: +11.7 pp to 16K: −21.1 pp, while the 2k-trained checkpoint moves from 2K: +17.8 pp to 16K: −20.0 pp. For Nemotron, the 1k-trained checkpoint moves from 1K: +22.2 pp to 16K: −25.6 pp, while the 2k-trained checkpoint moves from 2K: +12.8 pp to 16K: −17.2 pp.

Table 1: Mean AIME output length at each model's largest budget in Figure 1 (Qwen: 64k; Gemma and Nemotron: 16k).

Checkpoint	Mean length
Qwen3.5-4B base	28.10k
4k-trained	3.83k
8k-trained	6.56k
Gemma4-E2B base	6.81k
1k-trained	1.26k
2k-trained	1.37k
Nemotron-3-Nano-4B base	5.48k
1k-trained	1.11k
2k-trained	1.89k

The output-length curves show that this is not only an accuracy plateau. The trained models rarely use the full additional token allowance. Table 1 reports mean output length at these endpoints. The base models average 5.48k–28.10k tokens, whereas the trained checkpoints average only 1.11k–6.56k tokens. Smaller training caps therefore shorten the model's overall response distribution rather than imposing a hard decoding limit.

2.2 Budget Internalization Across Training Tasks

We next test whether this pattern is specific to mathematical-reasoning training by replacing DeepScaleR with the diverse procedural tasks in Nemotron-RL-ReasoningGym-v1 (NVIDIA Corporation, 2026). Figure 2 shows that it is not. For Qwen, the 4k-trained checkpoint moves from 4K: +31.1 pp to 64K: −25.0 pp. Gemma's 2k-trained checkpoint moves from 2K: +21.7 pp to 16K: −15.0 pp. Nemotron shows a smaller in-budget effect: its 2k-trained checkpoint gains only 2K: +0.6 pp, but it then saturates at 49.4% while the base model reaches 60.0% at 16k (16K: −10.6 pp). The in-budget gain therefore depends on the model and training task, but the loss of test-time scaling appears after fixed-budget training on both hard reasoning datasets.

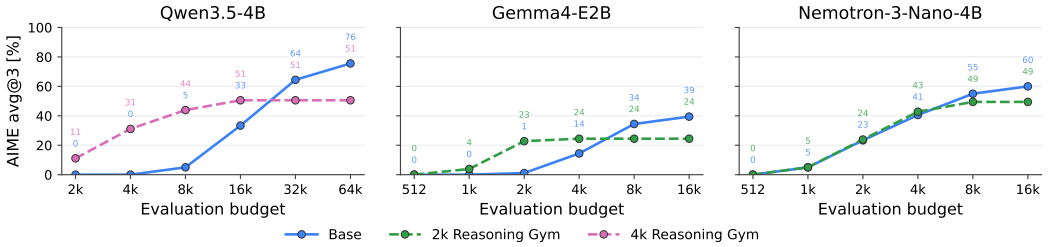


Figure 2: AIME accuracy after fixed-budget Reasoning Gym training across three model families, with dashed lines for trained checkpoints and colors matching Figure 1. Appendix Table 9 reports the exact values.

2.3 Transfer to Other Tasks and Budget Forcing

The behavior transfers from both training datasets to GPQA-Diamond, a graduate-level science benchmark, and LiveCodeBench v6, a code-generation benchmark with hidden tests (Figure 3). The trained checkpoints gain accuracy near their training budget and saturate while producing shorter responses, although the high-budget damage is smaller than on AIME. On GPQA and LiveCodeBench, both Qwen checkpoints preserve much more of the base model's high-budget accuracy than on AIME (GPQA at 64k: 4k-trained: −9.1 pp, 8k-trained: −2.2 pp; LCB-v6 at 64k: 4k-trained: −9.0 pp, 8k-trained: −1.0 pp). Gemma's 1k-trained checkpoint follows the same pattern on GPQA (16K: −1.0 pp) and LiveCodeBench (16K: −8.2 pp). Its 2k-trained checkpoint also remains close to the base model on GPQA (16K: −1.3 pp) and reduces the LiveCodeBench deficit (16K: −5.1 pp). Nemotron's 1k-trained checkpoint exceeds the base model on GPQA at high budget (64K: +1.0 pp) but not on LiveCodeBench (64K: −9.0 pp). Its 2k-trained checkpoint strengthens the GPQA improvement (64K: +4.0 pp) and narrows the LiveCodeBench deficit (64K: −4.2 pp). Reasoning Gym checkpoints follow the same pattern: Qwen remains below the base model on GPQA (64K: −6.7 pp) and LiveCodeBench (64K: −3.8 pp), as does Gemma (GPQA, 16K: −4.4 pp; LCB-v6, 16K: −4.6 pp), while Nemotron slightly exceeds the base model on GPQA (64K: +0.7 pp) but not on LiveCodeBench (64K: −4.8 pp). Because these differences are smaller, Appendix G additionally reports pass@3 curves for a more stable comparison. We hypothesize that these tasks often require less extended reasoning, so the base model sometimes overreasons rather than making productive use of additional tokens.

This behavior makes the low-budget comparison difficult to interpret. At low budgets, the base model often overreasons until it reaches the cap and fails to produce a final answer. Table 2 shows that this is common across all three benchmarks.

Table 2: Qwen3.5-4B base-model truncation rate (%) across evaluation budgets. Appendix E reports the corresponding curves and exact values for every model family and fixed-budget checkpoint.

Task	2k	4k	8k	16k	32k	64k
AIME	100.00	100.00	95.00	66.67	33.33	6.67
GPQA-Diamond	100.00	95.79	67.00	16.67	0.51	0.00
LiveCodeBench v6	99.81	97.14	81.52	70.86	51.81	45.14

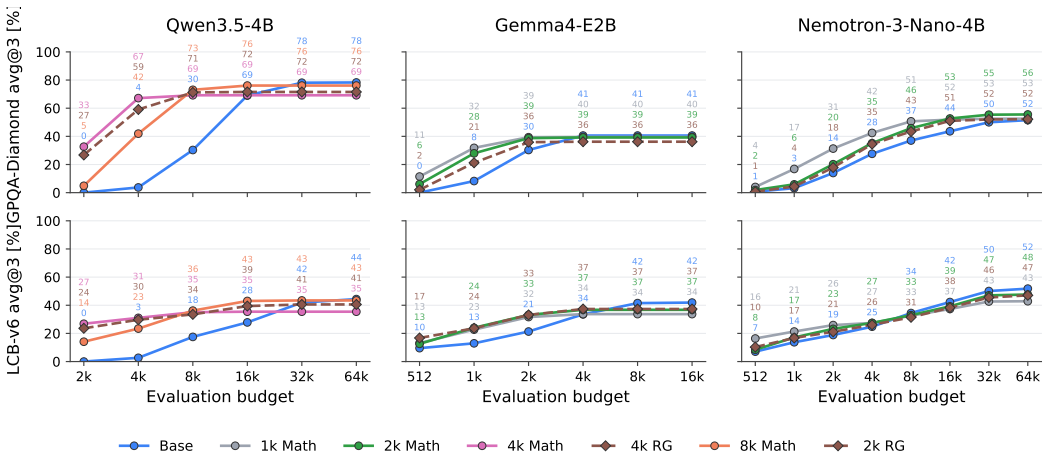


Figure 3: Accuracy on GPQA-Diamond and LiveCodeBench v6 after fixed-budget math (solid budget colors) and Reasoning Gym (brown dashed diamonds) training. Appendix Tables 10 and 11 report exact values; Appendix Figure 20 reports DeepScaleR output lengths.

We therefore first test whether the base model could answer correctly without continuing to overreason. We apply the budget-forcing procedure from Muennighoff et al. (2025). If a response reaches budget B without a parseable answer, we retain its reasoning prefix, insert the model-specific end-of-thinking delimiter and answer cue, and allow a short answer window. Naturally completed responses remain unchanged. Examples showing the prompt, model response, and budget-forcing intervention for all models and benchmarks are provided in Appendix C.1.

Answer forcing substantially improves low-budget accuracy across all three tasks (AIME, 4K: +19.4 pp; GPQA, 4K: +56.1 pp; LCB-v6, 4K: +18.9 pp; Figure 4). The same trend persists at larger budgets, where answer forcing generally preserves or improves accuracy. Appendix Figures 24 and 25 report the corresponding base-model intervention for Gemma and Nemotron.

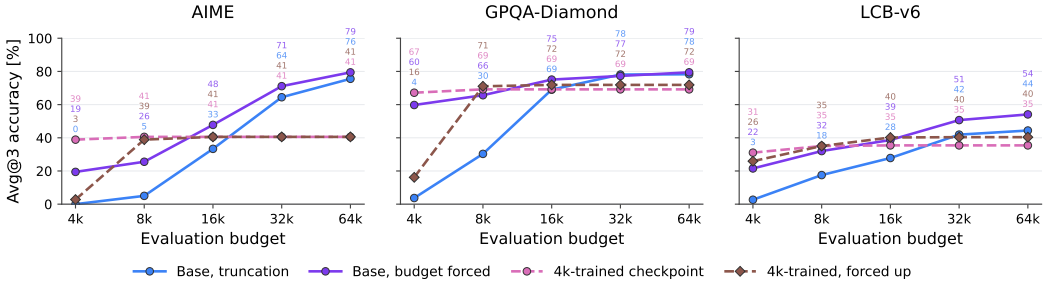


Figure 4: Qwen3.5-4B accuracy under ordinary base-model decoding, base-model answer forcing, the 4k-trained checkpoint, and forcing the trained checkpoint upward. Appendix Figure 18 and Table 7 report truncation rates and output lengths.

Next, we apply the reverse intervention to the 4k-trained checkpoint to test whether its loss of test-time scaling only reflects early stopping. We remove its end-of-thinking delimiter and final answer, append `Wait`, and ask it to continue the same trace, repeating this intervention up to four times under a 64k cap. Relative to ordinary decoding of the same checkpoint, forcing up lowers accuracy at the 4k training budget (AIME, 4K: −36.1 pp; GPQA, 4K: −51.0 pp; LCB-v6, 4K: −5.1 pp) and produces the same or only modestly higher accuracy at 64k (AIME, 64K: 0.0 pp; GPQA, 64K: +2.7 pp; LCB-v6, 64K: +5.0 pp), despite using 2.58–2.74k more tokens (Appendix C.2). Stating the budget explicitly in the prompt offers no reliable alternative, as models not trained on such cues largely ignore them (Appendix B).

Budget forcing can therefore make the base model answer earlier with fewer tokens and higher accuracy, while making the trained model spend more tokens does not restore its test-time scaling. This suggests that fixed-budget training genuinely compresses the reasoning needed to reach its answers rather than only teaching the model to stop early. Appendix C.1 gives complete examples of both procedures, while Appendix Figures 24 and 25 report the corresponding results showing the same pattern for Gemma and Nemotron.

2.4 Budget Internalization

Fixed-budget training therefore has three distinct effects: it can improve low-budget accuracy, make reasoning more economical, and reduce how much the model benefits from additional inference compute. We call the last effect *budget internalization*. A model internalizes its training budget when, above that budget, its base model continues to use more tokens and improve with additional compute while the trained model's response length and accuracy largely saturate.

3 What Affects Budget Internalization?

Having established this behavior, we now inspect the potential causes of budget internalization. We hypothesize that it arises from how the training objective treats truncated rollouts. To understand where this behavior may come from, let us inspect the GRPO objective and identify where the generation budget enters training (Shao et al., 2024):

$$\mathcal{J}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\mathbb{E}_{\{o_i\}\sim\pi_{\theta_{\text{old}}}(\cdot|q;B,\Pi)}\left[w_B(q,\{o_i\})\sum_i c_B(|o_i|)\ell_i - \beta D_{\text{KL}}(\pi_\theta\|\pi_{\text{ref}})\right]. \quad (1)$$

The budget B first enters the rollout policy by limiting how long a response can be generated, while the termination protocol Π determines what happens when this limit is reached. Here, the per-response update is $\ell_i = \sum_t m_{\epsilon^-,\epsilon^+}(r_{i,t})\hat{A}_i(R_\Pi(o_i;B))$. The budget enters the reward $R_\Pi(o_i;B)$ through the score assigned to complete and truncated responses, the normalization $c_B(|o_i|)$ through the weight given to responses of different lengths, and the prompt-selection term w_B through which sampled groups are retained for training. The token-level likelihood ratio $r_{i,t}$ is transformed by the clipping rule $m_{\epsilon^-,\epsilon^+}$, while β and the reference policy determine how strongly the update remains anchored to the base model. Finally, the dataset $\mathcal{D}$ determines how often the cap binds in the first place. In our standard GRPO run, Qwen3.5-4B is trained on $\mathcal{D}$ = DeepScaleR with $B = 8$k, R_Π assigns 1 to a correct completed response and 0 to an incorrect or truncated one, $\hat{A}_i$ standardizes this reward within a group of $G = 8$ rollouts, $c_B(|o_i|) = 1/(G|o_i|)$, $w_B = 1$ so that every group is retained, $m_{\epsilon^-,\epsilon^+}$ is PPO clipping with $\epsilon^- = \epsilon^+ = 0.2$, and $\beta = 0$. Each ablation below changes exactly one of these factors and keeps everything else, including the optimizer, update count, and evaluation protocol, identical to this run. For these ablations, we report avg@16 and compute every accuracy delta relative to the base model at the same evaluation budget.

Does high truncation exposure on hard datasets cause budget internalization? Beyond the RL algorithm, budget internalization depends heavily on the dataset's difficulty relative to the token cap. When a dataset requires long reasoning, many rollouts hit the cap and receive zero reward, creating a strong negative gradient against extended thinking. To isolate this data effect, we vary only the training dataset while holding the algorithm and 8K budget fixed. We compare GSM8K (Cobbe et al., 2021), DeepScaleR (Luo et al., 2025), and Nemotron-RL-Math-v2 (Du et al., 2025), which span a wide range of natural reasoning lengths (Table 6). We observe a trade-off between in-budget learning and test-time scaling. DeepScaleR provides the largest in-budget gain but loses it above the training budget (8K: $+41.3$ pp; 64K: -22.3 pp). Nemotron-RL-Math-v2 follows the same pattern (8K: $+34.2$ pp; 64K: -24.2 pp), while the simpler GSM8K dataset produces a smaller gain but preserves more of the base model's high-budget accuracy (8K: $+11.6$ pp; 64K: -6.8 pp). **High truncation exposure on hard datasets causes the model to internalize the budget as a hard limit**, whereas training on simpler datasets preserves the model's ability to reason beyond the training budget.

Does the way we treat truncated rollouts cause budget internalization? A rollout does not always finish before reaching its token budget, at which point the training procedure has to decide

Table 3: AIME avg@16 accuracy (%) of Qwen3.5-4B after GRPO ablations at an 8k training budget and after rejection fine-tuning. Gray ± values are 95% confidence intervals across the 60 problem-level avg@16 scores. Green and red values improve or degrade relative to the base model; bold and underlined values are best and second best in each column.

Ablation	Variant	2k	4k	8k (train)	16k	32k	64k
Base model		0.0±0.0	0.3±0.6	9.2±6.2	34.2±10.8	65.4±10.1	78.8±8.5
GRPO	DeepScaleR	1.1±1.7	22.7±9.8	**50.4**±10.4	56.5±10.0	56.5±10.0	56.5±10.0
Data	GSM8K	0.0±0.0	2.8±2.7	20.7±9.2	52.7±11.2	**70.6**±9.4	72.0±9.2
	Nemotron-RL-Math-v2	3.4±2.5	19.9±8.0	43.3±10.1	54.0±9.4	54.6±9.5	54.6±9.5
Truncation	Loss masking	1.0±1.0	16.1±7.6	38.3±10.7	56.8±9.9	59.7±9.4	59.7±9.4
	DAPO overlength penalty	**6.4**±4.2	**27.2**±9.9	44.2±10.5	49.7±10.2	50.0±10.2	50.0±10.2
	Forced answer	0.0±0.0	0.1±0.2	7.0±5.3	31.6±10.4	66.1±9.8	77.2±8.5
Normalization	Dr.GRPO	1.5±2.1	21.9±9.5	48.4±10.9	55.6±10.5	55.6±10.5	55.6±10.5
KL anchoring	$\beta = 0.001$	0.6±0.9	18.3±8.6	47.1±10.7	56.2±10.1	56.2±10.1	56.2±10.1
	$\beta = 0.01$	0.0±0.0	8.9±5.9	32.4±11.0	**57.5**±10.5	69.4±8.8	70.1±8.7
RFT	Short traces (<4K)	0.0±0.0	0.3±0.6	7.8±4.8	30.3±10.5	63.5±10.5	**83.0**±7.9
	Budget-forced traces (4K)	0.0±0.0	5.7±3.5	25.1±9.2	25.5±9.1	26.4±9.0	26.6±9.0

what to do with it. Standard GRPO assigns it zero reward, so we try three augmentations that treat such rollouts differently: loss masking removes it from the policy-gradient objective; DAPO applies a soft overlength penalty that decreases from 0 to -1 over the final 512 tokens (Yu et al., 2025); and forced-answer training closes the truncated rollout's reasoning and forces an answer prefix, giving the model up to 120 extra tokens to answer, similarly to the budget-forcing control in Section 2.3. These choices produce substantially different outcomes. Standard GRPO produces a large in-budget gain but falls below the base model at high budget (8K: $+41.3$ pp; 64K: -22.3 pp). DAPO worsens this trade-off (8K: $+35.0$ pp; 64K: -28.8 pp), while loss masking also does not help (8K: $+29.2$ pp; 64K: -19.1 pp). Forced-answer training nearly removes the high-budget gap but gives up the in-budget gain (8K: -2.2 pp; 64K: -1.6 pp). Thus, **how truncated rollouts are scored has a direct effect on budget internalization**: penalizing them makes it stronger, masking them is insufficient, and recovering an answer can nearly remove it at the cost of in-budget learning. This also suggests that the effect depends on the response-length distribution seen during training—in particular, on whether short or long responses provide the successful training signal.

Does response-level length normalization cause budget internalization? Standard GRPO averages token losses within each response using the $1/|o_i|$ factor, where o_i is the completion. This normalization introduces a length bias: it amplifies positive updates on short correct responses and dilutes negative updates on long incorrect responses (Liu et al., 2025b). Because budget internalization reduces long-horizon reasoning, this bias is a plausible algorithmic mechanism. We test this by rerunning the 8K-budget Qwen experiment with Dr.GRPO (Liu et al., 2025b), changing only the normalization from response length $|o_i|$ to the fixed generation budget B, while keeping the data, reward, optimizer, and update count fixed. Dr.GRPO (8K: $+39.3$ pp; 64K: -23.1 pp) performs similarly to standard GRPO (8K: $+41.3$ pp; 64K: -22.3 pp). **Response-level length bias is not the cause of budget internalization** in our setting.

Can anchoring the policy to the base model preserve elasticity? Budget internalization might emerge because the RL update allows the policy to drift too far from its capable initialization. Because the base model already possesses significant long-budget reasoning capabilities, anchoring the trained policy to the base model via a Kullback-Leibler (KL) penalty, which keeps the policy closer to the reference as the penalty grows (Mroueh, 2025), might prevent the model from forgetting how to extrapolate. We evaluate KL regularization ($\beta = 0.001$ and $\beta = 0.01$) while holding the dataset and 8K training budget fixed. The weak penalty performs similarly to standard GRPO ($\beta = 0.001$: 8K: $+37.9$ pp; 64K: -22.5 pp). Increasing the penalty preserves more high-budget accuracy but reduces the in-budget gain ($\beta = 0.01$: 8K: $+23.2$ pp; 64K: -8.6 pp). **KL regularization serves as a highly effective control for preserving above-budget capability** at the cost of in-budget learning.

Is budget internalization an artifact of RL? The preceding results suggest that budget internalization depends on the response-length distribution and how truncated rollouts are penalized in RL.

However, the dynamics of GRPO might also play a key role and be critical for this effect to emerge. To isolate the influence of the training data, we ablate the training method. We test supervised fine-tuning (SFT) on the model's own successful rollouts (rejection fine-tuning, RFT), keeping only those that terminate naturally within 4K tokens. We then compare this to artificially shortened traces of the same length constructed via budget forcing (Section 2.3). Specifically, we take rollouts that naturally exceed 4K tokens, truncate their reasoning at exactly 4K, and append the correct final answer, forcing an abrupt transition from unfinished reasoning to a conclusion. Both sets of traces are at most 4K tokens long and differ only in whether the reasoning ends naturally. Short-trace RFT does not produce the elasticity drop seen in GRPO and exceeds the base model at high budget despite no in-budget gain (8K: -1.4 pp; 64K: $+4.3$ pp), whereas budget-forced RFT reproduces the GRPO failure (8K: $+15.9$ pp; 64K: -52.2 pp). Budget internalization is therefore not specific to RL: supervised training induces it as well, but only when the training traces contain truncation. Naturally terminated traces reinforce concise reasoning where the model already concludes within the budget, without imposing the budget as a stopping rule, which parallels the GSM8K results, where few rollouts reach the cap. Avoiding truncation thus mitigates budget internalization, but yields no improvement at 4K or 8K, unlike standard GRPO (8K: $+41.3$ pp). Overall, **budget internalization emerges whenever the training signal explicitly enforces a budget constraint** (whether through truncation penalties on hard datasets in RL, or artificially shortened trajectories in SFT).

Taken together, the ablations point away from a single GRPO-specific objective term and toward the distribution of responses reinforced during training. Changing response-length normalization or applying an overlength penalty does not restore scaling, while reducing truncation exposure preserves substantially more high-budget accuracy. The way truncated rollouts are scored has the largest effect: penalizing them strengthens internalization, masking them is insufficient, and recovering an answer nearly removes the high-budget gap only by giving up the in-budget gain. KL regularization exposes the same trade-off more gradually, preserving more of the base model's high-budget capability as anchoring increases but reducing accuracy at the training budget. The SFT controls show that this behavior is not specific to RL, since naturally completed trajectories preserve scaling whereas artificially shortened trajectories reproduce the contraction. Across these interventions, stronger in-budget gains generally coincide with greater budget internalization, and none preserves both the full low-budget improvement and the base model's test-time scaling. Table 3 reports the exact comparison, and Appendix Figure 5 shows the complete response curves.

4 Related Work

Reasoning in LLMs. Reasoning has become a central capability of recent large language models. It was first elicited through chain-of-thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022), and theoretical work shows that some problems cannot be solved without such intermediate computation (Feng et al., 2023; Merrill & Sabharwal, 2024; Li et al., 2024). Subsequent work moved from eliciting reasoning at inference time to training it directly. Guo et al. (2025) showed that reinforcement learning with verifiable rewards (RLVR) alone can induce extended reasoning without human demonstrations, and current models typically combine supervised bootstrapping on externally generated traces with a subsequent RL stage (Yang et al., 2025; NVIDIA et al., 2026). This RL stage runs under a fixed generation budget, and it is the setting in which we study budget internalization. Some recent models additionally expose a reasoning-effort setting that selects the amount of reasoning at inference time (Qwen Team, 2026b; DeepSeek-AI et al., 2026). Such explicit budget conditioning may mitigate budget internalization by turning the training cap from an implicit prior into a clearly defined condition.

Limits of test-time reasoning. The benefits of reasoning are uneven. Gains concentrate on mathematical and symbolic tasks and are weaker or negative elsewhere (Sprague et al., 2025; Liu et al., 2025a), and even where reasoning helps, performance remains brittle under counterfactual or perturbed problems (Wu et al., 2024; Mirzadeh et al., 2025; Dziri et al., 2023). Longer reasoning is also not uniformly better: models overthink, and accuracy can plateau or decline as the budget grows (Ma et al., 2025; Gema et al., 2025; Zhao et al., 2026; Shojaee et al., 2025). Our budget-forcing experiments are consistent with this picture, as base models often reach a correct answer well before they stop reasoning. Budget internalization adds a complementary failure mode, in which training removes the model's ability to benefit from additional sequential compute.

Efficient and budget-aware reasoning. Overthinking has motivated a large body of methods that reduce unnecessary inference. Some allocate compute adaptively by deciding when to think, estimating a token budget, or conditioning on a user-specified budget (Damani et al., 2025; Zhang et al., 2025; Li et al., 2026; Wen et al., 2025; Xu et al., 2026), while others directly optimize reasoning length through length-controlled training, pruning, anytime objectives, or adaptive length penalties (Aggarwal & Welleck, 2025; Hou et al., 2026; Qi et al., 2025; Liu et al., 2026; Xiang et al., 2025; Yi et al., 2025). These approaches shorten reasoning by design. We instead show that standard fixed-budget RLVR shortens it as a side effect, and that this compression transfers beyond the training domain. The effect can be desirable when additional tokens bring little gain, as on LiveCodeBench, but harmful when they do, as on AIME. Related studies examine how the training horizon should relate to the base policy's solution length (Wachi et al., 2026; Xue et al., 2026) and how exploration can let test-time compute extrapolate beyond the training budget (Setlur et al., 2026). We complement them by isolating which components of the training signal cause budget internalization.

5 Limitations and Future Work

Our experiments show that fixed-budget post-training can change test-time scaling, but they do not establish how universal this effect is. The tested models have only four to five billion active parameters, each training condition has one seed, and most experiments use mathematical reasoning, so the same comparisons must be repeated at larger scales and on tasks with different reasoning demands. Algorithmic generality is also unresolved because our RL experiments focus on GRPO and its variants. However, the signals implicated by our results are not specific to GRPO: any RL algorithm must decide how to score truncated trajectories, whether to anchor the policy with KL regularization, and which response lengths enter through the training data. We therefore expect budget internalization to depend as much on these shared choices as on the optimizer itself, but direct comparisons with other RL algorithms are needed to test this hypothesis. Our behavioral measurements also cannot determine whether the model represents the cap explicitly or whether shorter successful trajectories simply shift its response distribution. The ablations suggest that changing a single objective term is insufficient, while changing which response lengths receive a successful training signal has a larger effect. Training across multiple rollout budgets, conditioning explicitly on available inference compute, and retaining long successful responses are promising ways to provide that signal. The central open problem is whether such training can preserve the low-budget gains of fixed-cap RLVR without removing the model's ability to benefit from more compute, which we leave to future work.

6 Conclusion

In this work, we introduce the term *budget internalization*: a learned constraint on test-time scaling induced by fixed-budget training. Across three model families and multiple benchmarks, we demonstrate that accuracy improves near the cap while response length and performance saturate beyond it, effectively damaging the model's capacity for test-time scaling. No tested method removes this trade-off, and the largest changes come from the training-data distribution and the treatment of truncated rollouts. Evaluation across reasoning budgets is therefore essential, since evaluation only near the training cap can miss a substantial loss of test-time scaling. At the same time, budget internalization is not always harmful. On AIME, it limits the model's ability to solve problems that require longer reasoning. On GPQA and LiveCodeBench, however, fixed-budget checkpoints often perform better with shorter responses, showing that internalization can improve inference efficiency. Budget forcing further shows that even base models can often produce an answer earlier without completing their natural reasoning trace, suggesting that overthinking is itself an important problem. This efficiency effect could be deliberately exploited on tasks that benefit little from extended reasoning, provided that accuracy and output-length curves are used to separate useful compression from lost capability.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing, code development, figure generation, and document formatting. All research decisions, experiments, numerical results, citations, and AI-assisted outputs were reviewed and verified by the authors. The authors take responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

We release all trained checkpoints, including ablations, at https://huggingface.co/budget-internalization-iclr2027. Appendix A reports the complete training and evaluation configuration. The remaining appendices provide the evaluation prompts, exact values underlying the main figures, and additional results needed to reproduce the reported comparisons.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=4jdIxXBNve.

Milad Aghajohari, Kamran Chitsaz, Amirhossein Kazemnejad, Sarath Chandar, Alessandro Sordoni, Aaron Courville, and Siva Reddy. The markovian thinker: Architecture-agnostic linear scaling of reasoning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 46743–46765, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/4dd0cfda282c778a21b38fd1d637f37b-Paper-Conference.pdf.

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

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of lm computation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 102783–102802, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ff414825df833edb8b1839e3d5d495e9-Paper-Conference.pdf.

DeepSeek-AI, :, Anyi Xu, B. Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, C. C. Yu, Chao Jin, Chaofan Lin, Chen Dong, Chenbing Wang, Chenfan Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyuan Zhang, Chenhao Xu, Chenqi Zhao, Chenze Shao, Chuhao Wang, Chuqi Zhang, Damai Dai, Dejian Yang, Deli Chen, Di Huang, Di Wu, Donghao Li, Erhang Li, Eric Fu, F. Zhou, Fangwei Zhou, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanglin Li, Guanting Chen, Guoai Cao, Guofan Fan, Guolai Meng, Guowei Li, Haichuan Zhang, Haiyang Ma, Haiyang Shen, Han Li, Han Yu, Han Zhang, Hangyuan Deng, Hanwei Xu, Hanxiang Xu, Hanxun Zhong, Hao Guo, Hao Jiang, Hao Li, Hao Qin, Haodong Wen, Haofen Liang, Haofeng Huang, Haohua Liu, Haoling Zhang, Haoming Luo, Haoran Yang, Haotian Xu, Haotian Yuan, Haoting Huang, Haowen Luo, Haoyang Cai, Haoyu Chen, Haozhe Ji, Hengran Zhang, Hengrui Wang, Hengxu Wu, Honghui Ding, Hongxuan Tang, Huadong Wang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J. Yang, J. H. Jin, J. H. Zhang, J. X. Zou, Jia Yu, Jiahui Zhou, Jiajun Chen, Jialiang Huang, Jialin Zhao, Jiamin Tang, Jian Zhou, Jianan Tong, Jianwen Li, Jiaqi Zhu, Jiarui Wang, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiaying Ding, Jibai Lu, Jiewen Hu, Jin Yan, Jincheng Zhai, Jingchang Chen, Jingcheng Hu, Jingli Zhou, Jingsheng Xu, Jingting Xiang, Jingyan Yun, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jinpeng Wang, Jinyi Chen, Jinyi Hu, Jiping Yu, Jueliang Guo, Junbo Pei, Junbo Sun, Junguang Jiang, Junjie Qiu, Junkang Zhou, Junqi Liu, Junren Li, Junxian Li, Junxiao Song, Junyi Guo, Kai Dong, Kaifeng Chen, Kaige Gao, Kang Guan, Kangdong Yuan, Ke Hong, Ke Xu, Kefan Zhao, Kexin Ji, Kexin Zhang, Kexing Zhou, Kuai Yu, Lan Zhang, Lean Wang, Lecong Zhang, Lei Wang, Letian Gao, Liang Zhao, Liansheng Xu, Lihua Guo, Lingxiao Luo, Lingyue Fu, Litao Deng, Litong Wang, Liyue Zhang, Longhao Chen, Lu Chen, Luotian Huang, Luyao

Ma, Luyao Wang, M. S. Di, Max Mei, Menghao Ye, Miao Cui, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingjing Zhang, Mingqi Wei, Mingshu Chen, Mingxing Liu, Mingxu Zhou, Mingyu Xu, Mingyu Yang, Mingze Wang, Muyang Chen, Ni Shentu, Ning Wang, Niufang Ning, Panpan Huang, Peixin Cong, Peiyi Wang, Peiyuan Xin, Pengfei Ren, Pengfei Yan, Pengle Zhang, Qi Kang, Qi Tang, Qiancheng Wang, Qiang Li, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qizhou Guo, Rongxian Xu, Rui Ding, Rui Hu, Rui Tian, Rui Yu, Ruidong Zhu, Ruifan Xu, Ruihan Yang, Ruihang Xia, Ruijie Lu, Ruilin Geng, Ruipeng Hong, Ruiqi Ge, Ruisong Zhang, Ruize Sun, Ruizhe Pan, Runji Wang, Runqian Chen, Runxin Xu, Ruohong Tian, Ruomeng Shen, Ruoyu Zhang, Ryan X., S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoyuan Chen, Shengding Hu, Shengkai Lin, Shengwen Ran, Shengyu Liu, Shengyuan Jia, Shi Bai, Shi Feng, Shicheng Xu, Shichun Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shiyuan Feng, Shufan Gong, Shuhan Lin, Shuiping Yu, Shunfeng Zhou, Shuo Yang, Shuomeng Wang, Shuting Guo, Shuting Pan, Shuying Yu, Sinuo Cao, Siyi Lin, Sizhe Chen, Songyang Chen, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tongrui Xiong, Wangding Zeng, Wei Liu, Wei Zhang, Weibin Xu, Weihao Zeng, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Shao, Wenkai Yang, Wenli Zhang, Wenlu Wang, Wenlve Huang, Wenqian Yan, Wentao Zhang, Xi Gao, Xiang He, Xiang Li, Xiangli Li, Xiangwen Wang, Xiangying Zhang, Xiankui Wei, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojian Qu, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaoyao Zou, Xiaoyuan Li, Xicheng Guo, Xieting Chu, Xin Cheng, Xin Liu, Xin Xie, Xinbo Xu, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xintong Yao, Xinyang Chen, Xinyong Jiang, Xinyu Yang, Xinyu Yang, Xu Chen, Xuanyu Wang, Xubei Zhong, Xuecheng Su, Xuejie Liu, Xuheng Lin, Xujie Fan, Xuncheng Zhao, Xuwei Fu, Y. C. Yan, Y. H. Jiang, Y. T. Wu, Y. W. M., Y. Z. Wang, Yafei Gao, Yang Yang, Yang Zhang, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Meng, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yaoyang Ye, Yehang Yin, Yexinrui Wu, Yi Qian, Yi Tao, Yi Yu, Yichao Zhang, Yichen Jiang, Yicheng Wang, Yifan Ding, Yifan Shi, Yifeng Peng, Yifeng Zhai, Yijia Wu, Yiliang Xiong, Yilun Wang, Ying He, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yiping Wang, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyao Yang, Yiyuan Liu, Yizai Cai, Yizhen Wei, Yizhi Wang, Yonglun Yang, Yongqi Zhuo, Yongqiang Guo, Yongtong Wu, Yu Wu, Yu Zhang, Yuan Bian, Yuan Cheng, Yuan Ou, Yuan Sun, Yuanfan Xu, Yuanhang Sun, Yuanhao Li, Yuchen Liu, Yuchen Yao, Yudong Han, Yuduan Wang, Yuhan Wu, Yuhao Meng, Yuheng Zou, YuKun Li, Yunchuan Wang, Yunfan Xiao, Yunfan Xiong, Yupeng Chen, Yuqian Cao, Yuqian Wang, Yuqing Chen, Yushun Zhang, Yutong Lin, Yuwei Xiao, Yuxian Gu, Yuxiang Chen, Yuxiang Huang, Yuxiang Luo, Yuxiang You, Yuxin Chen, Yuxin Xiang, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhe Guo, Yuzhen Huang, Yuzhuo Bai, Z. Y. Z., Zanlin Ni, Zehao Wang, Zehua Zhao, Zehui Ren, Zejun Zhao, Zhangli Sha, Zhanying Wang, Zhaochen Zhang, Zhaoshuai Du, Zhe Fu, Zhean Xu, Zhenda Xie, Zheng Liu, Zhengyan Zhang, Zhenhua Dong, Zhewen Hao, Zhibang Wang, Zhibin Gou, Zhicheng Ma, Zhihao Li, Zhihong Shao, Zhihuan Huang, Zhijie Li, Zhirui Lu, Zhixian Huang, Zhixuan Chen, Zhixuan Chen, Zhixuan Pan, Zhiyu Wu, Zhizhou Ren, Zhu He, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zili Zhang, Zilin Li, Zilong Hou, Zilong Lyu, Ziqiao Wang, Ziwei Xie, Ziya Zhang, Ziyi Gao, Zizheng Pan, Zonglin Li, Zongqing Yao, Zui Chen, Zuofan Wu, Chenchen Ling, Chengyu Hou, Chong Chen, D. Li, Di Qi, Dongjie Ji, Fang Wei, Fanyi Xia, Fei Xie, Feiyi Tan, Hailong Guo, Haiyan Zhai, Hui Zhou, Huihui Tan, Huijie Li, Jia Luo, Jia Song, Jialu Cai, Jian Liang, Jiangting Zhou, Jiaqi Gao, Jiayi Shao, Jie Chen, Jieyu Yang, Jin Chen, Jingde Zhang, Jingzi Zhou, Jinqian Wang, Jinyang Liu, JinZhao Sun, Junhua Ling, Junmin Zheng, Kaicheng Yang, Ke Xu, Le Su, Leyi Xia, Liangfeng Ding, Lin Zhuo, Linwang Ma, Linyan Zhu, Liyu Cai, Luqi Yao, M. K. Zhang, Meng Li, Miao Lin, Miaojun Wang, Min Zhang, Mingming Li, Mingming Wang, Mingze Yin, Minmin Han, Nan Cao, Ning Wang, Ningxin Ma, Panpan Wang, Peihan Lin, Peng Sun, Peng Zhang, Qian Ying, Qiang Xiang, Qiao Wang, Qingmiao Mao, Qiwei Jiang, Rongli Jin, Ruyi Chen, Sha Tao, Shangmian Sun, Shaoqing Wu, Shichao Zou, Si Lei, Tianyang Zhang, Tianyu Sun, Tingting Yin, W. L. Xiao, Wei An, Wei Li, Wei Wang, Weiwei Lin, Wenqing Hou, X. Lin, Xiangfei Meng, Xianzhu Huang, Xiao Peng, Xiaoqian Li, Xiaoting Zhang, Xiaowen Sun, Xiaoxiang Wang, Xiaoyu Ye, Xinrou Zhang, Xinyu Zhang, Xue Cao, Xueyin Chen, Yanan Zhou, Yanhong Xu, Yao Xia, Yao Xu, Yi Shao, Yihong Zhang, Yiling Ma, Ying Tang, Yining Lou, Yiru Chen, Yishi Piao, Yixuan Chen, Yong Xiong, Yuchen Xuan, Yuehan Yang, Yuer Xu, Yukun Zha, Yunxian Ma, Yuping Lin, Yuting Yan, Yutong Xie, Yuwen Sheng, Yuxuan Zhu, Zekai Zhang, Zhe Ju, Zhenzhen Lin, Zheren Gao, Zheyang Sun, Zhigang Yan, Zhongyu Wu, Zi Wang, Zihua Qu, Ziling Yan,

and Ziyi Wan. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026. URL https://arxiv.org/abs/2609.19969.

Wei Du, Shubham Toshniwal, Branislav Kisacanin, Sadegh Mahdavi, Ivan Moshkov, George Armstrong, Stephen Ge, Edgar Minasyan, Feng Chen, and Igor Gitman. Nemotron-math: Efficient long-context distillation of mathematical reasoning from multi-mode supervision, 2025. URL https://arxiv.org/abs/2512.15489.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 70293–70332. Curran Associates, Inc., 2023. doi: 10.52202/075280-3081. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/deb3c28192f979302c157cb653c15e90-Paper-Conference.pdf.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 70757–70798. Curran Associates, Inc., 2023. doi: 10.52202/075280-3100. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/dfc310e81992d2e4cedc09ac47eff13e-Paper-Conference.pdf.

Aryo Pradipta Gema, Alexander Hägele, Runjin Chen, Andy Arditi, Jacob Goldman-Wetzler, Kit Fraser-Taliente, Henry Sleight, Linda Petrini, Julian Michael, Beatrice Alex, Pasquale Minervini, Yanda Chen, Joe Benton, and Ethan Perez. Inverse scaling in test-time compute. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=NXgyHW1c7M. Featured Certification, J2C Certification.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*,

645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner 1 technical report, 2025. URL https://arxiv.org/abs/2505.22312.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. ThinkPrune: Pruning long chain-of-thought of LLMs via reinforcement learning. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=V51gPu1uQD.

Naman Jain, Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, 2022.

Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, Kai Jia, and Zhifang Sui. SelfBudgeter: Adaptive token allocation for efficient LLM reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21135–21156, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1063. URL https://aclanthology.org/2026.findings-acl.1063/.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 11911–11943, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/3309b4112c9f04a993f2bbdd0274bba1-Paper-Conference.pdf.

Ryan Liu, Jiayi Geng, Addison J. Wu, Ilia Sucholutsky, Tania Lombrozo, and Thomas L. Griffiths. Mind your step (by step): Chain-of-thought can reduce performance on tasks where thinking makes humans worse. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 38489–38517. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/liu25t.html.

Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng LIU, Yuntian Deng, Yizhe Zhang, and Junxian He. Learn to reason efficiently with adaptive length-based reward shaping. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 43279–43308, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/47795c4ae2f7d07ea2fb0d11fa2c3c90-Paper-Conference.pdf.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *Conference on Language Modeling (COLM)*, 2025b.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2, 2025. Notion Blog.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning models can be effective without thinking, 2025. URL https://arxiv.org/abs/2504.09858.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/1f59721c106ea80f613299039112f651-Paper-Conference.pdf.

Iman Mirzadeh, Keivan Alizadeh-Vahid, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 94743–94765, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ec2e7a896f8250986b3907f57621ce94-Paper-Conference.pdf.

Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification, 2025. URL https://arxiv.org/abs/2503.06639.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

NeMo RL. Nemo rl: A scalable and efficient post-training library. https://github.com/NVIDIA-NeMo/RL, 2025. GitHub repository.

NVIDIA. NVIDIA-Nemotron-3-Nano-4B-BF16. Hugging Face model card, March 2026. URL https://huggingface.co/nvidia/NVIDIA-Nemotron-3-Nano-4B-BF16.

NVIDIA, :, Aaron Blakeman, Aaron Thomas, Aastha Jhunjhunwala, Abhibha Gupta, Abhinav Khattar, Adam Rajfer, Adi Renduchintala, Adil Asif, Aditya Vavre, Adriana Flores Miranda, Ahmad Bilal, Aileen Zaman, Ajay Hotchandani, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, Alex Gronskiy, Alex Kondratenko, Alex Steiner, Alex Ye, Alexander Bukharin, Alexandre Milesi, Ali Taghibakhshi, Alice Gatti, Alisa Liu, Alok Kumar, Amar Phanishayee, Ameya Sunil Mahabaleshwarkar, Amir Klein, Amit Zuker, Amnon Geifman, Anahita Bhiwandiwalla, Ananth Subramaniam, Andrea Santilli, Andrew Fulks, Andrew McHarg, Andrew Tao, Andrii Skliar, Anjulie Agrusa, Ankur Srivastava, Ankur Verma, Anna Shors, Anna Warno, Antoni-Joan Solergibert I Llaquet, Arham Mehta, Arkadiusz Nowaczynski, Arti Jain, Ashwath Aithal, Ashwin Poojary, Asif Ahamed, Asit Mishra, Asma Kuriparambil Thekkumpate, Atefeh Sohrabizadeh, Avinash Kaur, Avinash Vem, Ayush Dattagupta, Barath Subramaniam Anandan, Bardiya Sadeghi, Ben Lanir, Benedikt Schifferer, Besmira Nushi, Bilal Kartal, Bill Thiede, Bita Darvish Rouhani, Bo Deng, Bob Schatz, Boris Ginsburg, Boxin Wang, Brad Nemire, Brandon Norick, Brian Dang, Brian Westphal, Brian Yu, Brucek Khailany, Bryan Catanzaro, Carlo del Mundo, Caryln Aarish, Chankyu Lee, Chantal Hwang, Charbel Sakr, Charles Wang, Charlie Truong, Chen Cui, Cheng Cheng, Cheng-Ping Hsieh, Chenghao Zhang, Chenhui Deng, Chintan Patel, Chris Alexiuk, Christian Cosgrove, Christian Munley, Christine Harvey, Christopher Parisien, Chunyang Shen, Coco Li, Collin Neale, Cynthia Gao, Cyril Meurillon, Dan Gil, Dan Su, Dan Zhao, Dane Corneil, Daniel Afrimi, Daniel Egert, Daniel Korzekwa, Daniel Lo, Daniel Machlab, Daniel Serebrenik, Daniil Sorokin, Daria Gitman, Daria Levy, Darko Stosic, David Mosallanezhad, David Yu, Davit Karamyan, Deena Donia, Deep Debroy, Deepak Narayanan, Devin O'Kelly, Dheeraj Peri, Dhruv Nathawani, Di, Wu, Dima Rekesh, Divyanshu Kakwani, Donald Plummer, Dong Anh, Dongfeng Yu, Dongfu Jiang, Donnie Kim, Dorrin Poorkay, Duncan Riach, Dusan Stosic, Dustin VanStee, Eavan Meng, Edgar Minasyan, Edward Lin, Eileen Margaret Peters Long, Elad Sarafin, Elad Segal, Elena Lantz, Ellie Evans, Elliott Ning, Eric Chung, Eric Harper, Eric Pham-Hung, Eric Tramel, Eric Yang, Erick Galinkin, Erik Pounds, Erika Goncalves

Goncalves, Evan Briones, Evan Wu, Evelina Bakhturina, Evgeny Tsykunov, Ewa Dobrowolska, Faisal Ladhak, Farzan Memarian, Fay Wang, Fei Jia, Felipe Soares, Felipe Vieira Frujeri, Feng Chen, Fengguang Lin, Ferenc Galko, Frank Sun, Frankie Siino, Frida Hou, Gal Hubara Agam, Gal Kaplun, Gantavya Bhatt, Gargi Prasad, Garvit Kulshreshtha, George Armstrong, Gerald Shen, Giulio Borghesi, Gordana Neskovic, Gorkem Batmaz, Grace Lam, Greg Mason, Greg Pauloski, Grigor Nalbandyan, Grzegorz Chlebus, Grzegorz Karch, Guan-Ting Liu, Guoming Zhang, Guyue Huang, Haggai Maron, Haifeng Qian, Haim Elisha, Haoxing Ren, Haran Kumar Shiv Kumar, Haribhau Hud, Harris Nover, Harrison Saturley Hall, Hayate Iso, Helen Ngo, Herbert Hum, Herman Sahota, Hexin Wang, Himanshu Soni, Hovhannes Tamoyan, Hua Li, Huanhuan Chen, Hui Li, Hui Wang, Huy Nguyen, Ian Chiles, Ido Galil, Ido Shahaf, Igor Gitman, Igor Shovkun, Ilya Loshchilov, Ingo Guehring, Itamar Schen, Itay Levy, Itay Neeman, Ivan Moshkov, Izik Golan, Izzy Putterman, Jaemin Choi, Jakub Slowikowski, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jatin Mitra, Jeffrey Glick, Jenny Chen, Jesse Oliver, Jiacheng Xu, Jiafan Zhu, Jialin Song, Jian Zhang, Jiantao Jiao, Jiaqi Zeng, Jie Lou, Jim King, Jimmy Zhang, Jingquan Wang, Jinhang Choi, Jinju Chu, Joey Conway, Joey Guman, Johan Jatko, Johannes Rausch, John Kamalu, John Roberts, Johnny Greco, Johnny Mensel, Jonah Alben, Jonas Yang, Jonathan Cohen, Jonathan Raiman, Joseph Jennings, Joshua Mabry, Joshua Pierce, Joyjit Daw, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kajal Jain, Kan Zhu, Kari Briski, Katherine Cheung, Katherine Luna, Keith Willowhawk, Keith Wyss, Keshav Santhanam, Kevin Shih, Kezhi Kong, Khanh Nguyen, Khushi Bhardwaj, Kirthi Shankar Sivamani, Konstantinos Krommydas, Krishna C. Puvvada, Krzysztof Pawelec, Kumar Anik, Kyle Keprios, Kylie Day, Lawrence McAfee, Leo Du, Leon Derczynski, Li Ding, Linda Liu, Lingjie Wu, Lior Kadoch, Lizzie Wei, Luis Vega, Luke Robison, Lun Su, Maarten Van Segbroeck, Maciej Jakub Mikulski, Maer Rodrigues de Melo, Magda Sypula, Mahan Fathi, Makesh Narsimhan Sreedhar, Makesh Tarun Chandran, Manoj Kilaru, Maor Ashkenazi, Marc Cuevas, Marc Romeijn, Marcin Chochowski, Mark Cai, Mark Mozolewski, Markus Kliegl, Marta Stepniewska-Dziubinska, Martyna Patelka, Mattei Machczynski, Matvei Novikov, Mauricio Ferrato, Maximilian Golub, Mehrzad Samadi, Melissa Corpuz, Mengru Wang, Mengxi Wu, Meredith Price, Meriem Boubdir, Micah Schaffer, Michael Andersch, Michael Boone, Michael Gschwind, Michael Lightstone, Michael Loh, Michal Bien, Michal Zawalski, Michelle Gill, Miguel Martinez, Mikail Khona, Mike Chrzanowski, Mike Houston, Mingyuan Ma, Minseok Lee, Mohamed Fawzy, Mohammad Dabbah, Mohammad Shoeybi, Mostofa Patwary, Nabin Mulepati, Najeeb Nabwani, Namit Dhameja, Narimane Hennouni, Natalie Hereth, Nathaniel Pinckney, Nave Algarici, Nave Assaf, Netanel Haber, Nicholas Knight, Nick Reamaroon, Nickson Quak, Nidhi Bhatia, Nikhil Desai, Nikolai Ludwig, Nima Tajbakhsh, Ning Xu, Nir Ailon, Nirmal Juluru, Nitin Nitin, Ofri Masad, Oleg Rybakov, Oleksii Hrinchuk, Oleksii Kuchaiev, Olivia Viessmann, Olivier Delalleau, Oluwatobi Olabiyi, Omer Ullman Argov, Omri Puny, Oren Tropp, Pablo Ribalta, Pallab Bhattacharya, Panos Lampropoulos, Parth Mannan, Pasha Shamis, Patrick Legresley, Paul Gibbons, Pavlo Molchanov, Pawel Morkisz, Peter Dykas, Peter Jin, Pierre-Yves Aquilanti, Pinky Xu, Piotr Januszewski, Piotr Laskiewicz, Pooya Jannaty, Prakash Gurumurthy, Pranav Prashant Thombre, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Puhui Meng, Qiyu Wan, Rabeeh Karimi Mahabadi, Rachel Oberman, Rachit Garg, Radha Sri-Tharan, Rahul Kandu, Rakshit Sanadhya, Ran El-Yaniv, Ran Zilberstein, Rasoul Shafipour, Ray Macalisang, Rayen Tian, Reka Kovacs, Renjie Pi, Rick Izzo, Rima Shahbazyan, Rishabh Garg, Rishi Puri, Rita Fernandes Neves, Ritchie Zhao, Ritika Borkar, Ritu Gala, Riyad Islam, Robert Clark, Robert Hesse, Robert Kirby, Roger Waleffe, Rohit Watve, Roi Koren, Ron Banner, Ruoxi Zhang, Russell J. Hewett, Ryan Prenger, Ryan Stewart, Ryota Egashira, Sadegh Mahdavi, Saee Paliwal, Sagar Singh, Sahil Modi, Salika Dave, Samantha Shinagawa, Samuel Kriman, Sandip Bhaskar, Sangkug Lym, Sanjay Kariyappa, Sanjeev Satheesh, Saran Vikas Murari, Satish Pasumarthi, Saurabh Mishra, Saurav Muralidharan, Scott Hara, Sean Narentharen, Selvaraj Anandaraj, Seonjin Na, Seonmeyong Bak, Seonmyeong Bak, Sepehr Sameni, Seph Mard, Serge Panev, Seth Henneman, Seth Poulos, Shahar Mor, Shantanu Acharya, Shaona Ghosh, Sharath Turuvekere Sreenivas, Sharon Mendelson, Shaun Kotek, Shawn Wang, Shay Aharon, Shaya Gharghabi, Sheng-Chieh Lin, Shi Chen, Shiqing Fan, Shirish Baskaran, Shreya Gopa, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shuoyang Ding, Shwetha Krishnamurthy, Siddharth Singh, Simeng Sun, Sirshak Das, Sivakumar Arayandi Thottakara, Smita Ithape, Somshubra Majumdar, Soumye Singhal, Sri Harsha Singudasu, Sridhar Bhuvanapalli, Srimukh Veccham, Stas Sergienko, Stefania Alborghetti, Stephen Ge, Su Rong, Sugam Dipak Devare, Sukrit Rao, Sumeet Kumar Barua, Sungsoo Ha, Sunny Gai, Suriya Gunasekar, Suseella Panguluri, Suyog Gupta, Sviataslau Hinzburh, Sweta Priyadarshi, Syeda Nahida Akter, Talor Abramovich, Tan Bui,

Tanay Varshney, Tatevik Ter-Hovhannisyan, Teodor-Dumitru Ene, Terry Kong, Thanh Do, Tianhe Zhang, Tiffany Moore, Tijmen Blankevoort, Tim Moon, Tiyasa Mitra, Tom Balough, Tomasz Grzegorzek, Tomasz Hliwiak, Tomer Asida, Tomer Bar Natan, Tomer Keren, Tomer Ronen, Tony Salim, Tony Wang, Traian Rebedea, Tugrul Konuk, Twinkle Vashishth, Udi Karpas, Ushnish De, Vahid Noorozi, Venkat Srinivasan, Venmugil Elango, Vibhor Agrawal, Victor Cui, Vijay Korthikanti, Vikas Mehta, Vinay Rao, Virginia Wu, Vitaly Kurin, Vitaly Lavrukhin, Vladimir Anisimov, Vu Pham, Wanli Jiang, Wasi Uddin Ahmad, Wataru Ishihara, Wei Du, Wei Ping, Weiheng Chai, Wenliang Dai, Wesley Helmholz, Will Jennings, Will Zhu, Wojciech Prazuch, Xiaowei Ren, Xiwen Yu, Yan Breek, Yang Chen, Yang Yu, Yangyi Chen, Yaniv Galron, Yashaswi Karnati, Yejin Choi, Yev Meyer, Yi-Fu Wu, Yian Zhang, Ying Lin, Yonatan Geifman, Yonggan Fu, Youngeun Kwon, Yu Yao, Yugi Guvvla, Yuki Huang, Yunsheng Liu, Zach Moshe, Zachary Newell, Zhilin Wang, Zhiyu Li, Zhongbo Zhu, Zhuolin Yang, Zihan Liu, Zijie Yan, and Zsolt-Alon Wertheimer. Nemotron 3 ultra: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning, 2026. URL https://arxiv.org/abs/2606.15007.

NVIDIA Corporation. Nemotron-RL-ReasoningGym-v1. Hugging Face dataset, January 2026. URL https://huggingface.co/datasets/nvidia/Nemotron-RL-ReasoningGym-v1. CC BY 4.0.

Penghui Qi, Zichen Liu, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Optimizing anytime reasoning via budget relative policy optimization. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 23429–23451. Curran Associates, Inc., 2025. doi: 10.52202/085713-0793. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/21efcb6a9fe5c90d660d3f560bb6094d-Paper-Conference.pdf.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Amrith Setlur, Matthew Yang, Charlie Snell, Jeremiah Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for llms. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 127323–127361, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/ced46a50befedcb884ccf0cbe8c3ad23-Paper-Conference.pdf.

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

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh vahid, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 108018–108059. Curran Associates, Inc., 2025. doi: 10.52202/085713-3603. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/9b26ad15462c81548c0689188d2e8018-Paper-Conference.pdf.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp.

10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b623663fd9b874366f3ce019fdfdd44-Paper-Conference.pdf.

Zayne Sprague, Fangcong Yin, Juan Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 94118–94162, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ead542f13a38179d1b55b88610f959a1-Paper-Conference.pdf.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-1913. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/528d56195a2c77c808494c86fa7c77ad-Paper-Datasets_and_Benchmarks_Track.pdf.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

UK AI Security Institute. Inspect ai: Framework for large language model evaluations, May 2024. URL https://inspect.aisi.org.uk/.

Akifumi Wachi, Hirota Kinoshita, Shokichi Takakura, Rei Higuchi, and Taiji Suzuki. A relative-budget theory for reinforcement learning with verifiable rewards in large language model reasoning, 2026. URL https://arxiv.org/abs/2602.01523.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Budgetthinker: Empowering budget-aware llm reasoning with control tokens, 2025. URL https://arxiv.org/abs/2508.17196.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1819–1862, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.102. URL https://aclanthology.org/2024.naacl-long.102/.

Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang Truong, Chelsea Finn, and Nick Haber. Just enough thinking: Efficient reasoning with adaptive length penalties reinforcement learning, 2025. URL https://arxiv.org/abs/2506.05256.

Yuhui Xu, Hanze Dong, Lei Wang, Doyen Sahoo, Junnan Li, and Caiming Xiong. Scalable chain of thoughts via elastic reasoning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 145390–145412, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/eb0965da1d2cb3fbbbb8dbbad5fa0bfc-Paper-Conference.pdf.

Yihao Xue, Allan Zhang, Jianhao Huang, Amit Sahai, and Baharan Mirzasoleiman. Beyond what seems necessary: Hidden gains from scaling training-time reasoning length under outcome supervision, 2026. URL https://arxiv.org/abs/2602.00927.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Jingyang Yi, Jiazheng Wang, and Sida Li. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 39011–39043. Curran Associates, Inc., 2025. doi: 10.52202/085713-1302. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/377ca194437916417e3dbe0dc67254f1-Paper-Conference.pdf.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, juncai liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 113222–113244. Curran Associates, Inc., 2025. doi: 10.52202/085713-3775. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a4277440d50f1f15d2cb4c14f7e0c0d2-Paper-Conference.pdf.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. AdaptThink: Reasoning models can learn when to think. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3716–3730, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.184. URL https://aclanthology.org/2025.emnlp-main.184/.

James Xu Zhao, Bryan Hooi, and See-Kiong Ng. Test-time scaling in reasoning models is not effective for knowledge-intensive tasks yet. In *Agentic AI in the Wild: From Hallucinations to Reliable Autonomy*, 2026. URL https://openreview.net/forum?id=FQcrZAuVXR.

A Training and Evaluation Details

Table 4 reports the hyperparameters shared by the headline DeepScaleR and Reasoning Gym GRPO runs. All runs use 32 prompts per step, eight generations per prompt, a global rollout batch of 256, a micro-batch size of one, three epochs, and at most 300 optimizer steps. We use a 10-step linear warmup followed by cosine decay, bfloat16 precision, GRPO ratio clipping at 0.2, no reference-policy KL penalty, and seed 42. The rollout sampler uses temperature 1.0 and top-p 1.0. Each generated-token cap reserves an additional 2,048 tokens for the input. We validate every 10 steps and select the checkpoint with the highest validation accuracy.

Model	Training caps	Peak LR	Minimum LR	Optimizer	Weight decay	Training backend
Qwen3.5-4B	4k, 8k	5×10^{-7}	5×10^{-8}	Adam	0.01	Megatron
Gemma4-E2B	1k, 2k	3×10^{-6}	3×10^{-7}	FusedAdam	0.1	FSDP2
Nemotron-3-Nano-4B	1k, 2k	2×10^{-6}	2×10^{-7}	FusedAdam	0.1	FSDP2

Table 4: Hyperparameters for the headline fixed-budget GRPO runs. Settings not shown in the table are shared across model families as described above.

For evaluation, we follow each model's Hugging Face model-card recommendation. Table 5 lists the sampling parameters. All evaluations enable thinking, use sampling, draw multiple generations per problem, and set the generated-token limit to the evaluation budget. We explicitly set only temperature and top-k; all other sampling parameters use the defaults from the corresponding model card.

Model	Temperature	Top-k
Qwen3.5-4B	1.0	20
Gemma4-E2B	1.0	64
Nemotron-3-Nano-4B	1.0	Model-card default

Table 5: Evaluation sampling hyperparameters from the corresponding Hugging Face model cards (Qwen Team, 2026a; Gemma Team, 2026; NVIDIA, 2026). For Qwen, we use the recommended reasoning parameters across all benchmarks.

The reported token budget counts generated tokens and excludes prompt tokens. Truncated generations receive zero reward during training and zero accuracy credit during evaluation. Task-specific parsers and verifiers determine whether a completed final answer is correct.

Table 6: Difficulty and length statistics for the math training datasets, measured with Qwen3.5-4B in thinking mode. Truncation at 8K is the share of responses that do not finish within 8K tokens.

Dataset	**Pass Rate at 64K**	**Truncation at 8K**	**Median Output Tokens**
GSM8K	92%	24%	3,744
DeepScaleR	77%	79%	19,056
Nemotron-RL-Math-v2	31%	99%	60,879

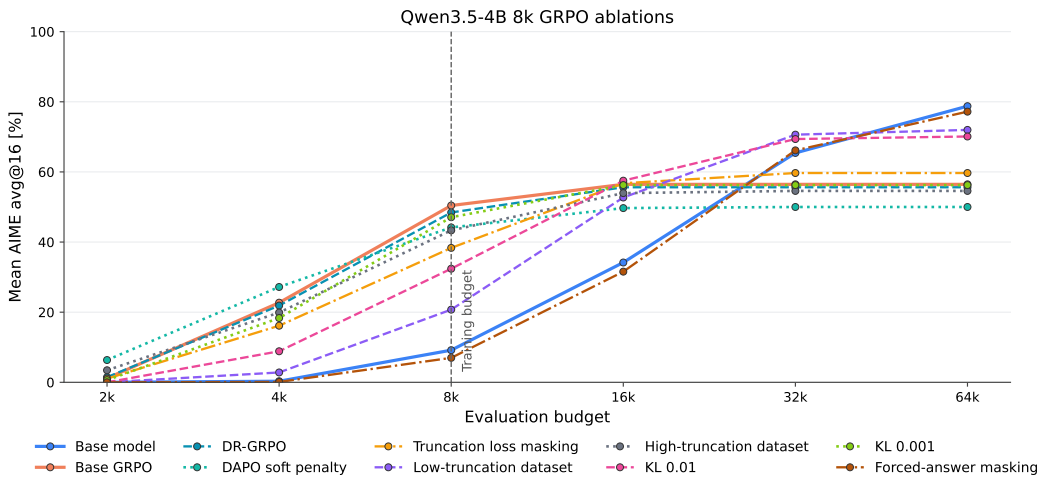


Figure 5: AIME avg@16 response curves for the Qwen3.5-4B 8k training ablations. The plot includes the base model, the standard 8k GRPO run, and all eight completed ablations. The two KL runs use $\beta = 0.001$ and $\beta = 0.01$. The vertical line marks the 8k training budget. Table 3 reports the exact values.

B Weak Budget Conditioning

We test whether a stated numeric budget can control generation without changing the available compute. In the budget-cue condition, we insert `You have a N-token reasoning budget.` immediately after the opening `<think>` tag. The no-cue condition leaves the prompt unchanged. Both conditions use the same 64k generation cap, while the cue states either 4k or 16k tokens. The checkpoints trained with these cues receive the same modified prompt during evaluation and are mechanically truncated at the stated budget. These 4k and 16k cue-conditioned checkpoints are separate training runs from the headline fixed-budget checkpoints, and the base-model control was rerun as part of this experiment. The resulting 66.7% no-cue base accuracy at 32k in Figure 7 therefore differs from the 64.4% headline value because it comes from an independent evaluation sample.

The results are presented in Figure 6. The base model does not follow either stated budget as a hard limit. The 4k cue leaves its AIME accuracy unchanged at 76.1% and shortens its mean response only from 27.3k to 23.4k tokens. For the 4k cue-conditioned checkpoint, the cue shortens the mean response from 5.6k to 2.5k tokens but also reduces accuracy from 52.2% to 38.3%. The 16k cue has a milder effect: the base model changes from 76.1% to 77.2% while again averaging 23.4k tokens, and the 16k cue-conditioned checkpoint shortens from 10.6k to 9.3k while accuracy falls from 65.0% to 59.4%. Thus, a cue can shift length, particularly for a checkpoint trained with that cue, but it does not provide calibrated budget control and can reduce accuracy.

B.1 Cue Placement Ablations

The base-model ablation varies whether the cue is placed in the user prompt, system prompt, or immediately after the opening `<think>` tag. We evaluate no-cue and stated-budget-cue conditions at a shared 32k-token generation cap. The prompt states that the reasoning budget is either 4k or 16k tokens. We report these results in Figure 7.

Additionally, we train the base model with three listed budget cue placement variants at 4k reasoning cap (the prompt during training is faithful, meaning that it always states that the reasoning budget is 4k tokens). We evaluate with the cue in the same location used during training, also stating that the available budget is 4k, while the real truncation happens only at 32k. The results are displayed in Figure 8.

We observe that these alternative budget cue placements have a marginal effect on both base and trained models' output length and accuracy.

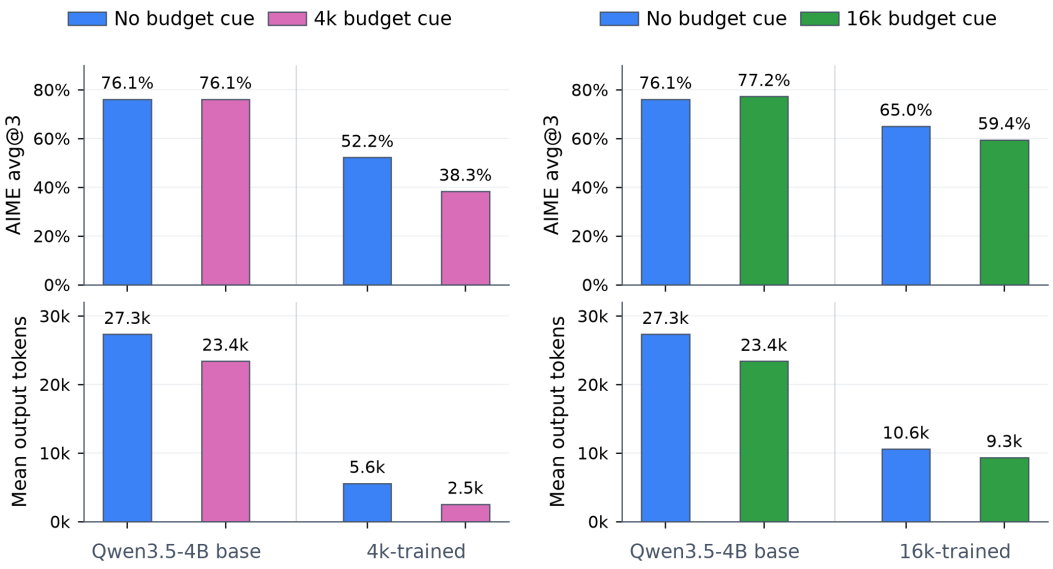


Figure 6: Budget-cue ablation under a shared 64k generation cap. Left: no cue versus a 4k cue for the base model and 4k cue-conditioned checkpoint. Right: no cue versus a 16k cue for the base model and 16k cue-conditioned checkpoint. The stated budget changes, but the available generation budget does not.

C Prompt and Termination Protocols

C.1 Budget-Forcing Prompt Presets

We match each budget-forcing preset to the model's chat template and the domain's answer format. Figures 9–17 report the complete rendered prompt for every preset. For each of them, we apply the same procedure: take a truncated model answer, insert the end-of-thinking token and benchmark-specific prefill, and allow the model a short response window for completing the answer.

For AIME, we allow 120 tokens for continuation and apply the standard last-5-lines scorer (meaning that we look for a parseable answer in the last five lines of the response). For GPQA, we allow 16 tokens and stop once the model has generated a newline. For LiveCodeBench v6, we allow 1024 additional tokens and attempt to extract a fenced Python block from the answer.

C.2 Extending Reasoning with `Wait`

We take the 4k-trained Qwen3.5-4B checkpoint's initial generation, discard its final answer, remove the end-of-thinking delimiter, append `Wait`, and ask it to continue the same reasoning trace. We repeat the insertion up to four times under a maximum 64k response length and evaluate on AIME24&25, GPQA-Diamond, and LiveCodeBench v6. For comparison, we report ordinary base and trained-checkpoint decoding together with answer forcing on the base model.

At 64k, upward forcing increases mean generation length by 2.58k tokens on AIME, 2.74k on GPQA-Diamond, and 2.65k on LiveCodeBench v6, but accuracy saturates by 16k on AIME and GPQA-Diamond and by 32k on LiveCodeBench v6. The intervention therefore makes the checkpoint spend more tokens without restoring test-time scaling.

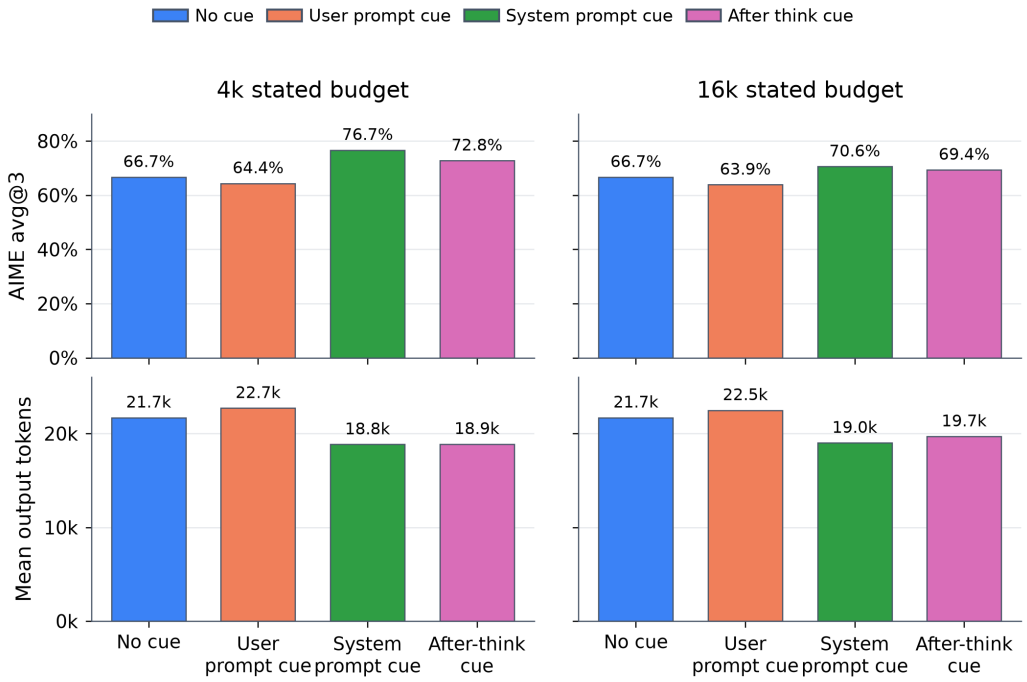


Figure 7: Base-model placement ablation at a shared 32k generation cap. Columns show 4k- and 16k-token budget cues; rows show combined AIME avg@3 and mean output tokens. The no-cue control is shared across cue placements.

Table 7: Exact values underlying Figure 18, shown as truncation rate in percent / mean generation length in thousands of tokens.

Task	Evaluation	4k	8k	16k	32k	64k
AIME	Base	100.00 / 4.00	95.00 / 7.88	66.67 / 14.37	33.33 / 22.56	6.67 / 28.10
	Base + answer forcing	0.56 / 0.72	1.11 / 1.37	1.11 / 4.58	0.00 / 11.04	0.00 / 22.98
	4k-trained	22.22 / 2.99	1.11 / 3.21	1.11 / 3.30	1.11 / 3.47	1.11 / 3.83
	4k-trained + forcing up	96.11 / 3.98	11.11 / 6.30	0.00 / 6.41	0.00 / 6.41	0.00 / 6.41
GPQA-Diamond	Base	95.79 / 3.97	67.00 / 7.33	16.67 / 10.43	0.51 / 11.14	0.00 / 11.15
	Base + answer forcing	0.51 / 0.17	1.52 / 2.51	1.18 / 8.13	0.00 / 10.01	0.00 / 10.01
	4k-trained	5.22 / 2.34	0.00 / 2.37	0.00 / 2.37	0.00 / 2.37	0.00 / 2.37
	4k-trained + forcing up	79.46 / 3.89	1.18 / 5.11	0.00 / 5.11	0.00 / 5.11	0.00 / 5.11
LCB-v6	Base	97.14 / 3.98	81.52 / 7.58	70.86 / 13.64	51.81 / 23.32	45.14 / 38.32
	Base + answer forcing	53.14 / 2.59	1.14 / 3.49	0.76 / 4.57	1.52 / 9.68	0.19 / 18.26
	4k-trained	44.00 / 2.69	22.48 / 3.95	3.62 / 4.73	1.71 / 5.08	1.33 / 5.54
	4k-trained + forcing up	69.33 / 3.51	45.33 / 5.78	10.10 / 7.71	0.00 / 8.20	0.00 / 8.20

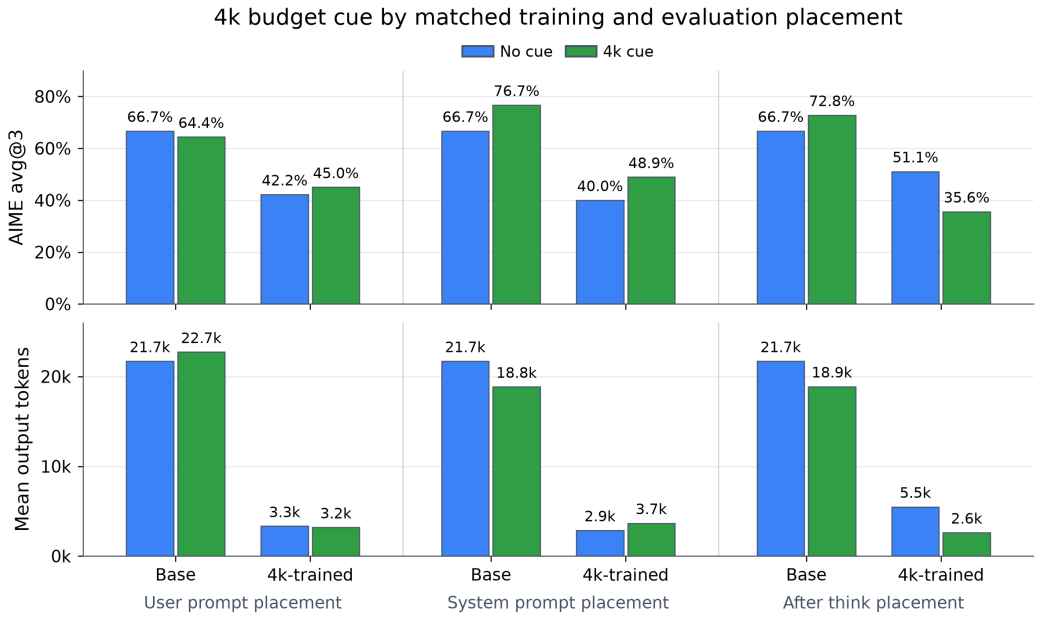


Figure 8: Matched-placement ablation for separately trained 4k cue-conditioned checkpoints at a shared 32k generation cap. Within each placement, the base model and the checkpoint trained with that placement are evaluated with no cue (blue) or a 4k budget cue (green). Rows show combined AIME avg@3 and mean output tokens.

SYSTEM
<|im_start|>system
<|im_end|>

USER
<|im_start|>user
Solve the following math problem step by step.
The last line of your response should be of the form ”ANSWER: ANSWER” (without quotes) where ANSWER is the answer to the problem.
Let N be the greatest four-digit positive integer with the property that whenever one of its digits is changed to 1, the resulting number is divisible by 7. Let Q and R be the quotient and remainder, respectively, when N is divided by 1000. Find Q+R.
Remember to put your answer on its own line at the end in the form ”ANSWER: ANSWER” (without quotes) where ANSWER is the answer to the problem, and you do not need to use a \{}boxed command.
<|im_end|>

ASSISTANT
<|im_start|>assistant
<think>
Let N be the greatest four-digit positive integer such that whenever one of its digits is changed to 1, the resulting number is divisible by 7.
Let the decimal representation of N be $d_3d_2d_1d_0$, so $N = 1000d_3 + 100d_2 + 10d_1 + d_0$.
Since N is a four-digit integer, $d_3 \in 1, 2, \ldots, 9$ and $d_2, d_1, d_0 \in 0, 1, \ldots, 9$.
The condition states that for each position $i \in 0, 1, 2, 3$, if we replace d_i with 1, the new number must be divisible by 7.
Let M_i denote the number obtained by changing the digit at position i to 1.
We are given $M_i \equiv 0 \pmod 7$ for all i=0, 1, 2, 3.
Let's write N in terms of its digits.
d_0: units place, coefficient 1.
d_1: tens place, coefficient 10.
d_2: hundreds place, coefficient 100.
d_3: thousands place, coefficient 1000.
...
Diffs:
$(1) - (2) : (6d_3 - 6d_3) + (2d_2 - 2d_2) + (3d_1 - d_0) + (1 - 3) \equiv 0 \implies 3d_1 - d_0 - 2 \equiv 0 \implies 3d_1 - d_0 \equiv 2.$
My previous calc was $3d_1 - d_0 - 4 \equiv 0$, which was using d_0 term in (2) having coeff 1, and 3 term in (1)?
Wait, Eq 2 has constant term 3? No, d_1 becomes 1. The coefficient is 10 -¿ 3.
In N, term is $d_1 \cdot 10 \to 3d_1$. In M_1, term is $1 \cdot 10 \to 3$. Constant term is $d_0 \cdot 1$.
So Eq 2 is $6d_3 + 2d_2 + 3 + d_0 \equiv 0$. Correct.
Eq 1 is $6d_3 + 2d_2 + 3d_1 + 1 \equiv 0$

</think>
Final Answer:
\boxed{10}

— Evaluator insertion — Generated continuation

Figure 9: Complete math budget-forced example for Qwen3.5-4B on AIME 2024. The initial reasoning call reached the 4,096-token cap and was shortened here. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<bos><|turn>system
<|think|>
<turn|>

USER
<|turn>user
Solve the following math problem step by step. The last line of your response should be of the form "ANSWER: $ANSWER" (without quotes) where $ANSWER is the answer to the problem.
Every morning Aya goes for a 9-kilometer-long walk and stops at a coffee shop afterwards. When she walks at a constant speed of s kilometers per hour, the walk takes her 4 hours, including t minutes spent in the coffee shop. When she walks $s + 2$ kilometers per hour, the walk takes her 2 hours and 24 minutes, including t minutes spent in the coffee shop. Suppose Aya walks at $s + \frac{1}{2}$ kilometers per hour. Find the number of minutes the walk takes her, including the t minutes spent in the coffee shop.
Remember to put your answer on its own line at the end in the form "ANSWER: $ANSWER" (without quotes) where $ANSWER is the answer to the problem, and you do not need to use a \boxed command.

<turn|>

ASSISTANT
<|turn>model
<|channel>thought
Let D be the distance of the walk, $D = 9$ kilometers. Let s be the speed of walking in kilometers per hour (km/h). Let t be the time spent in the coffee shop in minutes.
We are given two scenarios.
Scenario 1: Speed is s km/h. Time taken for the walk is 4 hours, including t minutes in the coffee shop.
The time spent walking is T_1. $T_1 = D/s = 9/s$ hours. The total time elapsed is $T_1 + t/60$ hours. We are given that the total time is 4 hours. $9/s + t/60 = 4$ (Equation 1)
Scenario 2: Speed is $s + 2$ km/h. Time taken for the walk is 2 hours and 24 minutes, including t minutes in the coffee shop. 2 hours and 24 minutes is $2 + 24/60 = 2 + 0.4 = 2.4$ hours. $T_2 = D/(s + 2) = 9/(s + 2)$ hours. The total time elapsed is $T_2 + t/60$ hours. $9/(s + 2) + t/60 = 2.4$ (Equation 2)
We have a system of two equations with two variables s and t.
Let's simplify the equations.
Equation 1: $9/s + t/60 = 4\ t/60 = 4 - 9/s\ t = 60(4 - 9/s)\ t = 240 - 540/s$ (Equation 1')
Equation 2: $9/(s+2) + t/60 = 2.4\ t/60 = 2.4 - 9/(s+2)\ t = 60(2.4 - 9/(s+2))\ t = 144 - 540/(s+2)$ (Equation 2')
Now we equate the expressions for t: $240 - 540/s = 144 - 540/(s + 2)$
Let's solve for s. $240 - 144 = 540/s - 540/(s + 2)\ 96 = 540[1/s - 1/(s + 2)]\ 96 = 540[(s + 2 - s)/(s(s + 2))]$
$96 = 540[(2)/(s^2 + 2s)]\ 96 = 1080/(s^2 + 2s)$
$s^2 + 2s = 1080/96$
Let's simplify the fraction $1080/96$. $1080/96$. Both are divisible by 12. $1080/12 = 90\ 96/12 = 8\ 1080/96 = 90/8$.
$90/8 = 45/4$.
$s^2 + 2s = 45/4\ 4s^2 + 8s = 45\ 4s^2 + 8s - 45 = 0$
We can solve this quadratic equation for s using the quadratic formula: $s = [-8 \pm \sqrt{8^2 - 4(4)(-45)}]/(2 \cdot 4)$
$s = [-8 \pm \sqrt{64 + 16(45)}]/8\ 16 \times 45 = 16 \times (40 + 5) = 640 + 80 = 720.\ s = [-8 \pm \sqrt{64 + 720}]/8$
$s = [-8 \pm \sqrt{784}]/8$
We need $\sqrt{784}$. $20^2 = 400$, $30^2 = 900$. It ends in 4, so the root ends

<channel|>
Final Answer:
\boxed{30} — Evaluator insertion — Generated continuation

Figure 10: Complete math budget-forced example for Gemma4-E2B on AIME 2024. The initial reasoning call reached the 1,024-token cap and is presented in full here. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<|im_start|>system
<|im_end|>

USER
<|im_start|>user
Solve the following math problem step by step.
The last line of your response should be of the form "ANSWER: ANSWER" (without quotes) where ANSWER is the answer to the problem.
Let N be the greatest four-digit positive integer with the property that whenever one of its digits is changed to 1, the resulting number is divisible by 7. Let Q and R be the quotient and remainder, respectively, when N is divided by 1000. Find Q+R.
Remember to put your answer on its own line at the end in the form "ANSWER: ANSWER" (without quotes) where ANSWER is the answer to the problem, and you do not need to use a \{}boxed command.
<|im_end|>

ASSISTANT
<|im_start|>assistant
<think>
We need largest four-digit number N such that if you change any digit to 1 (presumably any single digit, any position), the resulting number is divisible by 7.
Interpretation: For each digit position i (thousands, hundreds, tens, ones), replace that digit with 1, keep other digits same, resulting 4-digit number divisible by 7.
We need N such that all four numbers formed by changing one digit to 1 are multiples of 7.
We need greatest such N (largest). Then compute Q = floor(N/1000) (the thousand digit), R = N mod 1000 (last three digits). Then Q+R.
Let's denote N = abcd (digits a,b,c,d each 0-9, a≥1). Changing each digit to 1 gives:
N_a = 1bcd (thousands digit becomes 1) = 1000*1 + 100*b + 10*c + d = 1000 + 100b + 10c + d
N_b = a1cd = 1000a + 100*1 + 10c + d = 1000a + 100 + 10c + d
N_c = ab1d = 1000a + 100b + 10*1 + d = 1000a + 100b + 10 + d
N_d = abc1 = 1000a + 100b + 10c + 1
All must be ≡0 mod7.
Thus we have congruences:
(1) 1000 + 100b + 10c + d ≡ 0 (mod7)
(2) 1000a + 100 + 10c + d ≡ 0 (mod7)
(3) 1000a + 100b + 10 + d ≡ 0 (mod7)
(4) 1000a + 100b + 10c + 1 ≡ 0 (mod7)
...
Now d ≡ 4c mod7: if c=0 → d ≡0. d digit 0-9 → d=0 or 7? but must satisfy 0-9 with mod condition maybe 0 or 7 (since 7≡0 mod7). Both okay.
Check (3): 6a +2b + d ≡4. Compute 6a≡12≡5. 2b=8≡1. So 5+1+d ≡4 → 6+d ≡4 → d ≡4-6 ≡ -2 ≡5 mod7. So d ≡5 mod7. But we previously got d ≡0 mod7. Conflict. So a=2 not possible.
Thus a must be 9.
Now a=9, b=4.
Compute (4): 6a =6*9=54 ≡ 54 mod7: 7*7=49 remainder5. So 5. 2b=8≡1. So 5+1+3c ≡6 → 6 +3c ≡6 → 3c ≡0 → c ≡0 mod7 → c =0 or7.
Now (3): 6a +2b + d ≡4. 6a ≡5, 2b≡1 → sum 6 + d ≡4 → d ≡5 (since -2 ≡5). So d ≡5 mod7. So d =5 or 12? digit 5 only (since 5<10). Also d ≡? check also d ≡4c mod7 from earlier.
If c=0 → d ≡0 mod7, but we need d ≡5 mod7 → conflict.
If
</think>
Final Answer:
\boxed{1300}

— Evaluator insertion — Generated continuation

Figure 11: Complete math budget-forced example for Nemotron-3-Nano-4B on AIME 2024. The initial reasoning call reached the 2,048-token cap and was shortened here. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<|im_start|>system
<|im_end|>

USER
<|im_start|>user
Answer the following multiple choice question. The last line of your response should be of the following format: 'ANSWER: LETTER' (without quotes) where LETTER is one of A,B,C,D. Think step by step before answering.
Two quantum states with energies E1 and E2 have a lifetime of 10^{-9} sec and 10^{-8} sec, respectively. We want to clearly distinguish these two energy levels. Which one of the following options could be their energy difference so that they can be clearly resolved?
A) $10^{-9}eV$
B) $10^{-4}eV$
C) $10^{-11}eV$
D) $10^{-8}eV$
<|im_end|>

ASSISTANT
<|im_start|>assistant
The user wants me to answer a multiple-choice question about quantum states, energy levels, and time-energy uncertainty principle.
1. Analyze the problem:
* Two quantum states have energies E_1 and E_2.
* They have lifetimes $\tau_1 = 10^{-9}$ sec and $\tau_2 = 10^{-8}$ sec.
* We want to "clearly distinguish" or resolve these two energy levels. This means we need to determine the minimum energy separation (resolution) required given the natural linewidth associated with their finite lifetimes.
* The relevant physics principle is the Time-Energy Uncertainty Principle: $\Delta E \cdot \tau \geq \hbar/2$ (or sometimes approximated as $\Delta E \approx \hbar/\tau or \Delta E \cdot \tau \sim h$). The width of the energy level line shape (natural broadening) is directly related to the lifetime of the state. Specifically, the full width at half maximum (FWHM) for a Lorentzian distribution is often taken as $\Gamma = \hbar/\tau$ in many contexts, though strictly speaking $\Delta E \Delta t \sim \hbar$. Let's assume the criterion for resolution is that the energy difference $\Delta E_{level} = |E_1 - E_2|$ must be greater than or comparable to the sum of the uncertainties (widths) of the individual states.
...
Is there any nuance where the answer might be derived differently?
Some texts use $\Delta E = h/\tau$.
$h \approx 4.14 \times 10^{-15}$ eV·s.
$\Delta E_1 \approx 4.14 \times 10^{-6}$ eV.
$\Delta E_2 \approx 4.14 \times 10^{-7}$ eV.
Still, the critical threshold is $\sim 10^{-6}$ eV.
Option B (10^{-4}) is still the only one exceeding this.
The order of magnitude remains consistent.
Are there cases where "clearly resolved" means much larger than the linewidth?
Usually "clearly resolved" implies seeing two distinct peaks. Rayleigh criterion suggests sum of widths.
Even sum is $\sim 10^{-6}$ eV.
So 10^{-4} eV is very clear.
10
</think>
ANSWER: B

— Evaluator insertion — Generated continuation

Figure 12: Complete science budget-forced example for Qwen3.5-4B on GPQA-Diamond. The initial reasoning call reached the 4,096-token cap and was shortened here. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<bos><|turn>system
<|think|>
<turn|>

USER
<|turn>user
Answer the following multiple choice question. The last line of your response should be of the following format: 'ANSWER: LETTER' (without quotes) where LETTER is one of A,B,C,D. Think step by step before answering.
Consider a system of three spins S_1, S_2, and S_3, each of which can take spin $+1$ or -1. The energy is $E = -J(S_1S_2 + S_1S_3 + S_2S_3)$.
Find the partition function Z of the system ($\beta = 1/(kT)$, where k is the Boltzmann constant and T is temperature).
A) $Z = 6e^{2J\beta} + 2e^{-2J\beta}$
B) $Z = 2e^{2J\beta} + 6e^{-2J\beta}$
C) $Z = 2e^{3J\beta} + 6e^{-J\beta}$
D) $Z = 2e^{-3J\beta} + 6e^{J\beta}$
<turn|>

ASSISTANT
<|turn>model
<|channel>thought
The user has presented a problem asking for the partition function Z of three spins S_1, S_2, and S_3, with interaction energy $E = -J(S_1S_2 + S_1S_3 + S_2S_3)$.
The spins can take values +1 or -1.
The partition function Z is defined as:
$Z = \sum_{\text{states}} e^{-\beta E}$
where $\beta = 1/(kT)$.
The system has $2^3 = 8$ possible microstates. We calculate E for each state and sum the Boltzmann factors $e^{-\beta E}$.
The energy function is $E = -J(S_1S_2 + S_1S_3 + S_2S_3)$.
Let's list the possible spin configurations (S_1, S_2, S_3) and calculate E.
The term $S_1S_2 + S_1S_3 + S_2S_3$ is the sum of interaction terms. Since $S_i \in \{-1, +1\}$, the product S_iS_j is either +1 (if $S_i = S_j$) or -1 (if $S_i \neq S_j$).
Let N_{same} be the number of pairs (i, j) such that $S_i = S_j$.
$S_1S_2 = 1$ if $S_1 = S_2$, -1 otherwise.
$S_1S_3 = 1$ if $S_1 = S_3$, -1 otherwise.
$S_2S_3 = 1$ if $S_2 = S_3$, -1 otherwise.
$E = -J \sum_{i<j} S_iS_j$.
We can analyze the configurations based on the number of +1 spins and -1 spins.
Case 1: All spins are the same (3 states).
1a) (+1, +1, +1):
$S_1S_2 = 1$, $S_1S_3 = 1$, and $S_2S_3 = 1$.
$S_1S_2 + S_1S_3 + S_2S_3 = 1 + 1 + 1 = 3$.
$E = -3J$.
$e^{-\beta E} = e^{3J\beta}$.
There is 1 such state.
1b) (-1, -1, -1):
$S_1S_2 = 1$, $S_1S_3 = 1$, and $S_2S_3 = 1$.
$S_1S_2 + S_1S_3 + S_2S_3 = 3$.
$E = -3J$.
$e^{-\beta E} = e^{3J\beta}$.
There is 1 such state.
Total contribution from Case 1: $2e^{3J\beta}$.
Case 2: Two spins are the same, one is different (6 states).
2a) (+1, +1, -1): (and permutations: (+1, -1, +1), (-1, +1, +1)).
For (+1, +1, -1):
$S_1S_2 = 1$, $S_1S_3 = -1$, and $S_2S_3 = -1$.
$S_1S_2 + S_1S_3 + S_2S_3 = 1 - 1 - 1 = -1$.
$E = -J(-1) = J$.
$e^{-\beta E} = e^{-J\beta}$.
There are 3 such states with energy J.
Contribution: $3e^{-J\beta}$.
2b) (-1, -1, +1): (and permutations: (-1, +1, -1), (+1, -1, -1)).
For (-1, -1, +1):
$S_1S_2 = 1$, $S_1S_3 = -1$, and $S_2S_3 = -1$.
$S_1S_2 + S_1S_3 + S_2S_3 = 1 - 1 - 1 = -1$.
$E = J$.
$e^{-\beta E} = e^{-J\beta}$.
There are 3 such states
<channel|>
ANSWER: B

— Evaluator insertion — Generated continuation

Figure 13: Complete science budget-forced example for Gemma4-E2B on GPQA-Diamond. The initial reasoning call reached the 1,024-token cap and is presented in full here. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<|im_start|>system
<|im_end|>

USER
<|im_start|>user
Answer the following multiple choice question. The last line of your response should be of the following format: 'ANSWER: LETTER' (without quotes) where LETTER is one of A,B,C,D. Think step by step before answering.
A spin-half particle is in a linear superposition 0.5—↑⟩+$\sqrt{3}$/2—↓⟩ of its spin-up and spin-down states. If —↑⟩ and —↓⟩ are the eigenstates of σ_z , then what is the expectation value up to one decimal place, of the operator 10σ_z+5σ_x ? Here, symbols have their usual meanings
A) 1.65
B) -0.7
C) 0.85
D) -1.4
<|im_end|>

ASSISTANT
<|im_start|>assistant
<think>
We need to answer a multiple choice about expectation value of operator O = 10 σ_z + 5 σ_x for a spin-1/2 state $|\psi\rangle$ = 0.5 | ↑⟩ + $\sqrt{3}$/2 | ↓⟩, where | ↑⟩ and | ↓⟩ are eigenstates of σ_z.
We are to find expectation value $\langle\psi|O|\psi\rangle$ up to one decimal place.
We need to compute expectation for O = 10 σ_z + 5 σ_x.
We have expectation = 10 $\langle\psi|\sigma_z|\psi\rangle$ + 5 $\langle\psi|\sigma_x|\psi\rangle$.
First compute $\langle\psi|\sigma_z|\psi\rangle$.
We know σ_z | ↑⟩ = +1 | ↑⟩, σ_z | ↓⟩ = -1 | ↓⟩.
Hence $\langle\psi|\sigma_z|\psi\rangle = (0.5)^*\langle\uparrow| + (\sqrt{3}/2)^*\langle\downarrow|(\sigma_z)(0.5|\uparrow\rangle + \sqrt{3}/2|\downarrow\rangle) =$?
Simplify: $\sigma_z|\psi\rangle = 0.5|\uparrow\rangle - (\sqrt{3}/2)|\downarrow\rangle$ (since σ_z flips sign on ↓). Then expectation =
$\langle\psi|\sigma_z|\psi\rangle = \langle\psi|(0.5|\uparrow\rangle - (\sqrt{3}/2)|\downarrow\rangle) = (0.5 * 0.5) + ((\sqrt{3}/2) * (\sqrt{3}/2))$? Wait careful.
Let's compute: $\langle\psi|\sigma_z|\psi\rangle = (0.5 * \langle\uparrow| + (\sqrt{3}/2) * \langle\downarrow|)(\sigma_z(0.5|\uparrow\rangle + (\sqrt{3}/2)|\downarrow\rangle))$
But $(\sigma_z(0.5|\uparrow\rangle) = 0.5|\uparrow\rangle, (\sigma_z((\sqrt{3}/2)|\downarrow\rangle) = -(\sqrt{3}/2)|\downarrow\rangle)$. So we have =
$(0.5 * \langle\uparrow|)(0.5|\uparrow\rangle) + (\sqrt{3}/2 * \langle\downarrow|)(-(\sqrt{3}/2)|\downarrow\rangle) = 0.5 * 0.5 - (\sqrt{3}/2) * (\sqrt{3}/2) = 0.25 - (3/4) = 0.25 - 0.75 = -0.5.$
Thus $\langle\psi|\sigma_z|\psi\rangle = -0.5$.
Alternatively could compute directly using probabilities: Expectation value of $\sigma_z = (p_up - p_down) = (p_up - p_down)$ because eigenstates with eigenvalues +1 and -1. Here $p_up = |0.5|^2 = 0.25, p_down = |\sqrt{3}/2|^2 = 3/4 = 0.75$. So expectation = 0.25 - 0.75 = -0.5. Yes consistent.
Thus $10 * \langle\sigma_z\rangle = 10 * (-0.5) = -5$.
Now compute $\langle\psi|\sigma_x|\psi\rangle$.
σ_x flips spin: $\sigma_x|\uparrow\rangle = |\downarrow\rangle, \sigma_x|\downarrow\rangle = |\uparrow\rangle$.
Thus $\sigma_x|\psi\rangle = 0.5|\downarrow\rangle + (\sqrt{3}/2)|\uparrow\rangle$.
Thus $\langle\psi|\sigma_x|\psi\rangle = \langle\psi|(0.5|\downarrow\rangle + (\sqrt{3}/2)|\uparrow\rangle)$. Compute:
$\langle\psi| = 0.5\langle\uparrow| + \sqrt{3}/2\langle\downarrow|$.
So expectation = $(0.5\langle\uparrow|)(0.5|\downarrow\rangle) + (\sqrt{3}/2\langle\downarrow|)(\sqrt{3}/2|\uparrow\rangle) = 0.5 * 0 + (\sqrt{3}/2) * (\sqrt{3}/2)\langle\downarrow|\uparrow\rangle$?? Wait need inner product:
$\langle\downarrow|\uparrow\rangle = 0$. So the first term is zero because $\langle\uparrow|\downarrow\rangle = 0$. So expectation = $(\sqrt{3}/2) * (\sqrt{3}/2) * \langle\downarrow|\uparrow\rangle$? That's zero.
Thus $\langle\psi|\sigma_x|\psi\rangle = 0$.
Alternatively expectation value of $\sigma_x = \langle\sigma_x\rangle = 0$ if the state is real phase? Actually generic expectation depends on off-diagonal elements of σ_x in the state basis. Since state is (a,b) with a=0.5, b=$\sqrt{3}/2$, real. Then expectation ¡σ_x¿ = 2 Im(a* b). Since a and b are both real, Im = 0. So expectation = 0. Yes.
Thus 5 * $\langle\sigma_x\rangle$ = 0.
Hence expectation value = -5 + 0 = -5.
But expected answer choices: A) 1.65, B) -0.7, C) 0.85, D) -1.4. None is -5. Something off? Let's double-check the scaling: The operator is 10σ_z + 5σ_x. Could they intend expectation per unit? Something else? Let's compute again but consider that expectation of σ_z = (p_up - p_down) = -0.5. So 10 * (-0.5) = -5. Then plus 5*0 = -5. So result -5.
But choices are much smaller magnitude. Could we have mistaken sign of $\langle\sigma_z\rangle$? Perhaps expectation =
$(p_up - p_down) = 0.25 - 0.75 = -0.5$. Yes.
Thus answer -5.
But options are far off. Maybe they want expectation value per operator components: like expectation of $\sigma_z = p_up - p_down = -0.5$. So expectation value of 10 σ_z is -5. Add 5 σ_x expectation of σ_x = (2 Im(a* b)) = 0. So final -5. But none matches.
Let's verify calculations for $\langle\psi|\sigma_x|\psi\rangle = 2\,\mathrm{Re}(a^*b)$? Wait formula: $\langle\sigma_x\rangle = 2\,\mathrm{Im}(a^*b)$? Actually for any state $|\psi\rangle = a|\uparrow\rangle + b|\downarrow\rangle$ (real coefficients), expectation of $\sigma_x = 2\,\mathrm{Im}(a^*b)$? That's for off-diagonal in Pauli basis? Let's recall: $\sigma_x = [[0, 1], [1, 0]]$.
$\langle\psi|\sigma_x|\psi\rangle = |a * b + b * a| = 2Re(a * b)$. Actually let's compute: $\langle\psi|\sigma_x|\psi\rangle = a * (\sigma_x|\psi\rangle)$?? Actually compute:
$|\psi\rangle = a|\uparrow\rangle + b|\downarrow\rangle$.
$\sigma_x|\psi\rangle = a|\downarrow\rangle + b|\uparrow\rangle$.
Thus $\langle\psi|\sigma_x|\psi\rangle = (a^*\langle\uparrow| + b^*\langle\downarrow|)(a|\downarrow\rangle + b|\uparrow\rangle) = a^*a\langle\uparrow|\downarrow\rangle + a^*b\langle\uparrow|\uparrow\rangle + b^*a\langle\downarrow|\uparrow\rangle + b^*b\langle\downarrow|\downarrow\rangle = 0 + a^*b + 0 + b^*b$? Wait careful:
$\langle\uparrow|\uparrow\rangle = 1 => a * b * 1 = a * b$.
$\langle\downarrow|\downarrow\rangle = 1 => b * b = |b|^2$.
But also cross term $a * b\langle\uparrow|\uparrow\rangle = a * b$.
Also $b * a\langle\downarrow|\uparrow\rangle = 0$.
Thus expectation = $a * b + b * b$? Actually let's recalc:
$\langle\psi|\sigma_x|\psi\rangle = (a * \langle\uparrow| + b * \langle\downarrow|)(\sigma_x|\psi\rangle) = (a * \langle\uparrow| + b * \langle\downarrow|)(a|\downarrow\rangle + b|\uparrow\rangle) = a * a\langle\uparrow|\downarrow\rangle + a * b\langle\uparrow|\uparrow\rangle + b * a\langle\downarrow|\uparrow\rangle + b * b\langle\downarrow|\downarrow\rangle$.
Now ⟨↑|
</think>
ANSWER: B

— Evaluator insertion — Generated continuation

Figure 14: Complete science budget-forced example for Nemotron-3-Nano-4B on GPQA-Diamond. The initial reasoning call reached the 2,048-token cap and is presented in full here. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<|im_start|>system
You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests.
<|im_end|>

USER
<|im_start|>user
Question:
You are given an integer X not less than 2.
Find the positive integer N such that N! = X.
Here, N! denotes the factorial of N, and it is guaranteed that there is exactly one such N.
Input
The input is given from Standard Input in the following format:
X
Output
Print the answer.
Constraints
- $2 \leq X \leq 3 \times 10^{18}$
- There is exactly one positive integer N such that N! = X.
- All input values are integers.
...
<|im_end|>

ASSISTANT
<|im_start|>assistant
<think>
The user wants a Python program to solve the following problem:
Given an integer X ($2 \leq X \leq 3 \times 10^{18}$), find a positive integer N such that $N! = X$.
It is guaranteed that such an N exists and is unique.
Constraints analysis:
X can be up to 3×10^{18}.
Let's estimate the maximum value of N.
$10! = 3{,}628{,}800 \approx 3.6 \times 10^{6}$.
$20! \approx 2.4 \times 10^{18}$.
$21! \approx 5.1 \times 10^{19}$.
Since $X \leq 3 \times 10^{18}$, N will not exceed 20. Wait, checking exact values.
$20! = 2{,}432{,}902{,}008{,}176{,}640{,}000 < 3 \times 10^{18}$.
$21! = 51{,}090{,}942{,}171{,}709{,}440{,}000 > 3 \times 10^{18}$.
...
Is there any interpretation where X is not an integer? "All input values are integers." OK.
Format: "Print the answer."
Let's double check the sample outputs.
Sample 1:
Input: 6
Output: 3
Matches.
Sample 2:
Input: 2432902008176640000
Output: 20
Matches.
Corner cases:
X = 2. Loop runs once. 'fact' becomes 2. 'n' becomes 2. Prints 2. Correct.
X = 20!.
Suppose X = 20!.
Loop continues until 'fact' becomes 20!.
Inside loop, 'fact' is initially 19!. Condition '$fact < X$' holds (19! < 20!).
Body executes: 'fact' becomes 20!
</think>
```python
import sys
def solve():
    # Read all input from standard input
    input_data = sys.stdin.read().split()
    not input_data:
        return
    # Parse the integer X
    # Using int() automatically handles large integers in Python
    x_val = int(input_data[0])
    n = 1
    fact = 1
    # We need to find N such that N! = X.
    # Given $X <= 3 * 10^{18}$, N will be at most 20 because $21! > 3 * 10^{18}$.
    # Therefore, a simple loop calculating factorials sequentially is efficient enough.
    while fact ¡ x_val:
        n += 1
        fact *= n
    # The problem guarantees that exactly one such positive integer N exists.
    # With this logic, 'fact' will eventually equal 'x_val' and the loop terminates.
    print(n)
if __name__ == '__main__':
solve()
```

— Evaluator insertion — Generated continuation

Figure 15: Complete code budget-forced example for Qwen3.5-4B on LiveCodeBench v6. The initial reasoning call reached the 4,096-token cap and was shortened here; full task description was also shortened. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<bos><|turn>system
<|think|>
You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests.
<turn|>

USER
<|turn>user
Question:
You are given an integer array nums of size n. For each index i where $0 <= i < n$, define a subarray nums[start ... i] where start = max(0, i - nums[i]).
Return the total sum of all elements from the subarray defined for each index in the array.
Example 1:
Input: nums = [2,3,1]
Output: 11
Explanation:
...
<turn|>

ASSISTANT
<|turn>model
<|channel>thought
Thinking Process:
1. **Understand the Goal:** The task is to calculate the sum of elements for a specific subarray defined for every index i in the input array 'nums'.
For each index i, the subarray is `nums[start ... i]`, where $\text{start} = \max(0, i - \text{nums}[i])$. We need the total of these subarray sums.
2. **Analyze the definition:**
Let n be the length of `nums`. We iterate i from 0 to $n - 1$.
For a given i:
a. Calculate $\text{start}_i = \max(0, i - \text{nums}[i])$.
b. The subarray is $\text{nums}[\text{start}_i], \ldots, \text{nums}[i]$.
c. Calculate $S_i = \sum_{j=\text{start}_i}^{i} \text{nums}[j]$.
d. The final result is $\sum_{i=0}^{n-1} S_i$.
3. **Example 1 Walkthrough: 'nums = [2, 3, 1]' (n=3)**
* **i = 0:**
nums[0] = 2.
$\text{start}_0 = \max(0, 0 - 2) = \max(0, -2) = 0$.
Subarray: `nums[0..0]` = [2]; $S_0 = 2$.
* **i = 1:**
nums[1] = 3.
$\text{start}_1 = \max(0, 1 - 3) = \max(0, -2) = 0$.
Subarray: `nums[0..1]` = [2, 3]; $S_1 = 5$.
* **i = 2:**
nums[2] = 1.
$\text{start}_2 = \max(0, 2 - 1) = \max(0, 1) = 1$.
Subarray: `nums[1..2]` = [3, 1]; $S_2 = 4$.
* Total sum: $S_0 + S_1 + S_2 = 2 + 5 + 4 = 11$. (Matches example output.)
...
5. **Implementation Strategy (Direct Simulation):**
A straightforward approach is to iterate through i. For each i, calculate start, and then calculate the sum from start to i.
* N is up to 100. An $O(N^2)$ solution is perfectly acceptable.
* Outer loop for i: 0 to N-1.
* Inner calculation (finding start and summing): O(i - start + 1), which is O(
<channel|>
```python
from typing import List
class Solution:
    def subarraySum(self, nums: List[int]) -¿ int:
        n = len(nums)
        total_sum = 0
        # Iterate through all ending indices i
        for i in range(n):
            # 1. Determine the starting index: start = max(0, i - nums[i])
            start = max(0, i - nums[i])
            # 2. Calculate the sum of the subarray nums[start ... i]
            current_subarray_sum = 0
            for j in range(start, i + 1):
                current_subarray_sum += nums[j]
            # 3. Add this sum to the total
            total_sum += current_subarray_sum
        return total_sum
```
— Evaluator insertion — Generated continuation

Figure 16: Complete code budget-forced example for Gemma4-E2B on LiveCodeBench v6. The initial reasoning call reached the 1,024-token cap and was shortened here; full task description was also shortened. Pink text is injected by the evaluator; blue text is generated by the model afterward.

SYSTEM
<|im_start|>system
You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests.
<|im_end|>

USER
<|im_start|>user
Question:
You are given an integer X not less than 2.
Find the positive integer N such that N! = X.
Here, N! denotes the factorial of N, and it is guaranteed that there is exactly one such N.
Input
The input is given from Standard Input in the following format:
X
Output
Print the answer.
Constraints
- $2 \leq X \leq 3 \times 10^{18}$
- There is exactly one positive integer N such that N!=X.
- All input values are integers.
...
<|im_end|>

ASSISTANT
<|im_start|>assistant
<think>
Okay, let's see. The problem is: given an integer X (at least 2), find the positive integer N such that N! equals X. And it's guaranteed there's exactly one such N.
First, I need to think about how factorials grow. N! for N starting at 1 is 1, 2, 6, 24, 120, etc. Since X is at least 2, N is at least 2. But the input X can be as large as 3e18. So, what's the maximum possible N here?
Let me calculate some factorials to estimate. Let's see:
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
...
But note: X can be as large as 3e18, but the factorial for n=20 is 2.43e18, which is within the range of Python integers (Python integers can be arbitrarily large, but for n up to 20, the factorial is manageable).
So the code:
Read X.
n = 1
fact = 1
while $fact < X$:
  n += 1
  fact *= n
print(n)
Yes.
Wait, but what if X is 1? But constraints say $X >= 2$, so no problem.
Another edge case: X=2 (n=2). The loop runs once: n becomes 2, fact=2, then
</think>
```python
def main():
  X = int(input().strip())
  n = 1
  fact = 1
  while $fact < X$:
    n += 1
    fact *= n
  print(n)
if __name__ == '__main__':
main()
```
— Evaluator insertion — Generated continuation

Figure 17: Complete code budget-forced example for Nemotron-3-Nano-4B on LiveCodeBench v6. The initial reasoning call reached the 2,048-token cap and was shortened here; full task description was also shortened. Pink text is injected by the evaluator; blue text is generated by the model afterward.

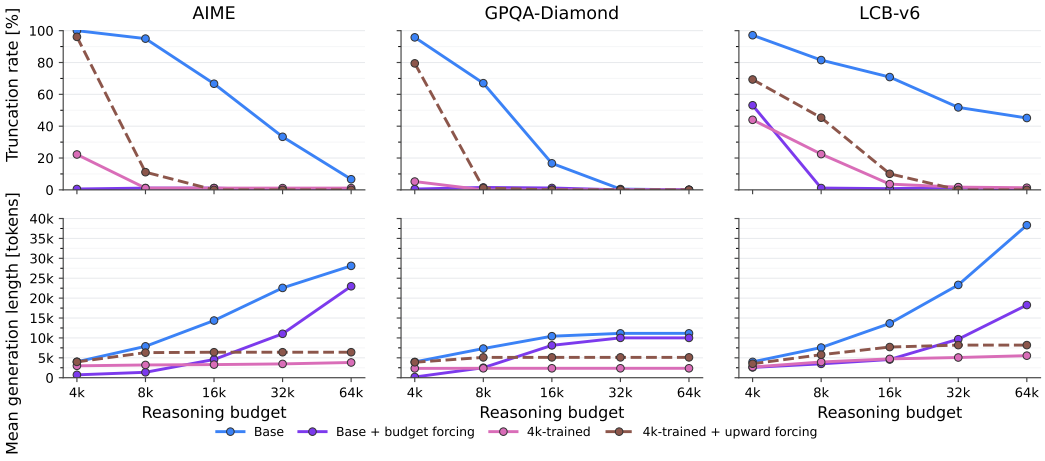


Figure 18: Truncation rate and mean generation length for ordinary base-model decoding, base-model answer forcing, the 4k-trained checkpoint, and forcing the trained checkpoint upward with `Wait`. Answer forcing makes the base model finish with fewer tokens, while forcing up lengthens the trained checkpoint's traces without restoring test-time scaling.

D EXACT MAIN-FIGURE VALUES

Table 8: Exact accuracy and mean output-length values underlying Figure 1. Accuracy is reported in percent and length in thousands of tokens.

Checkpoint	Metric	Evaluation budget					
		2k	4k	8k	16k	32k	64k
Qwen3.5-4B							
Base	Accuracy	0.00	0.00	5.00	33.33	64.44	75.56
	Length	2.00	4.00	7.88	14.37	22.56	28.10
4k-trained	Accuracy	11.67	38.89	40.56	40.56	40.56	40.56
	Length	1.95	2.99	3.21	3.30	3.47	3.83
8k-trained	Accuracy	1.67	22.78	51.67	55.56	55.56	55.56
	Length	2.00	3.80	6.02	6.56	6.56	6.56

Checkpoint	Metric	Evaluation budget					
		512	1k	2k	4k	8k	16k
Gemma4-E2B							
Base	Accuracy	0.00	0.00	1.11	14.44	34.44	39.44
	Length	0.51	1.00	2.00	3.85	6.20	6.81
1k-trained	Accuracy	2.22	11.67	17.78	18.33	18.33	18.33
	Length	0.51	0.90	1.21	1.26	1.26	1.26
2k-trained	Accuracy	0.56	7.78	18.89	19.44	19.44	19.44
	Length	0.51	0.94	1.33	1.37	1.37	1.37
Nemotron-3-Nano-4B							
Base	Accuracy	0.00	5.00	23.33	40.56	55.00	60.00
	Length	0.51	0.99	1.83	3.05	4.43	5.48
1k-trained	Accuracy	10.56	27.22	34.44	34.44	34.44	34.44
	Length	0.49	0.82	1.02	1.05	1.07	1.11
2k-trained	Accuracy	2.22	17.78	36.11	41.67	42.78	42.78
	Length	0.51	0.95	1.53	1.85	1.89	1.89

Table 9: Exact AIME accuracy values underlying Figure 2, reported in percent.

Model	Checkpoint	Evaluation budget					
		2k	4k	8k	16k	32k	64k
Qwen3.5-4B	Base	0.00	0.00	5.00	33.33	64.44	75.56
	RG 4k-trained	11.11	31.11	43.89	50.56	50.56	50.56

Model	Checkpoint	Evaluation budget					
		512	1k	2k	4k	8k	16k
Gemma4-E2B	Base	0.00	0.00	1.11	14.44	34.44	39.44
	RG 2k-trained	0.00	3.89	22.78	24.44	24.44	24.44
Nemotron-3-Nano-4B	Base	0.00	5.00	23.33	40.56	55.00	60.00
	RG 2k-trained	0.00	5.00	23.89	42.78	49.44	49.44

Table 10: Exact accuracy values underlying Figure 3, reported in percent.

Task	Checkpoint	Evaluation budget					
		2k	4k	8k	16k	32k	64k
Qwen3.5-4B							
GPQA	Base	0.00	3.70	30.30	69.02	78.11	78.28
	4k-trained	32.66	67.17	69.19	69.19	69.19	69.19
	8k-trained	4.88	41.92	73.06	76.09	76.09	76.09
LCB-v6	Base	0.00	2.67	17.52	27.81	41.90	44.38
	4k-trained	26.86	31.05	35.05	35.43	35.43	35.43
	8k-trained	14.10	23.43	36.19	43.05	43.43	43.43

Task	Checkpoint	Evaluation budget					
		512	1k	2k	4k	8k	16k
Gemma4-E2B							
GPQA	Base	0.00	8.25	30.30	40.57	40.57	40.57
	1k-trained	11.45	31.82	39.39	39.56	39.56	39.56
	2k-trained	6.06	27.95	38.72	39.23	39.23	39.23
LCB-v6	Base	9.52	12.95	21.33	33.52	41.52	41.90
	1k-trained	13.14	22.67	31.62	33.71	33.71	33.71
	2k-trained	12.57	24.00	33.14	36.76	36.76	36.76

Task	Checkpoint	Evaluation budget							
		512	1k	2k	4k	8k	16k	32k	64k
Nemotron-3-Nano-4B									
GPQA	Base	0.67	3.20	13.97	27.61	37.04	43.60	50.00	51.52
	1k-trained	4.04	16.84	31.31	42.42	50.67	51.85	52.53	52.53
	2k-trained	1.85	5.89	20.20	35.19	45.79	52.69	55.39	55.56
LCB-v6	Base	7.05	13.71	18.86	24.76	34.48	42.29	50.10	51.81
	1k-trained	16.38	21.33	25.90	27.43	32.57	37.33	42.67	42.86
	2k-trained	8.38	17.14	23.24	27.43	32.95	39.24	47.05	47.62

Table 11: Exact GPQA-Diamond and LiveCodeBench v6 accuracy values for the Reasoning Gym checkpoints in Figure 3, reported in percent.

Model	Task	Evaluation budget					
		2k	4k	8k	16k	32k	64k
Qwen3.5-4B, RG 4k	GPQA	26.77	59.09	71.21	71.55	71.55	71.55
	LCB-v6	23.62	29.71	33.52	39.43	40.57	40.57

Model	Task	Evaluation budget					
		512	1k	2k	4k	8k	16k
Gemma4-E2B, RG 2k	GPQA	2.02	21.21	35.86	36.20	36.20	36.20
	LCB-v6	16.76	23.62	33.14	37.33	37.33	37.33

Model	Task	Evaluation budget							
		512	1k	2k	4k	8k	16k	32k	64k
Nemotron-3-Nano-4B, RG 2k	GPQA	0.67	4.38	17.85	34.51	43.43	51.01	52.02	52.19
	LCB-v6	10.29	16.76	21.14	26.10	31.43	38.29	45.52	47.05

E TRUNCATION-RATE CURVES

Figure 19 reports the fraction of responses that reach the evaluation cap without completing across all model families, tasks, and fixed-budget checkpoints. Table 12 reports the exact values.

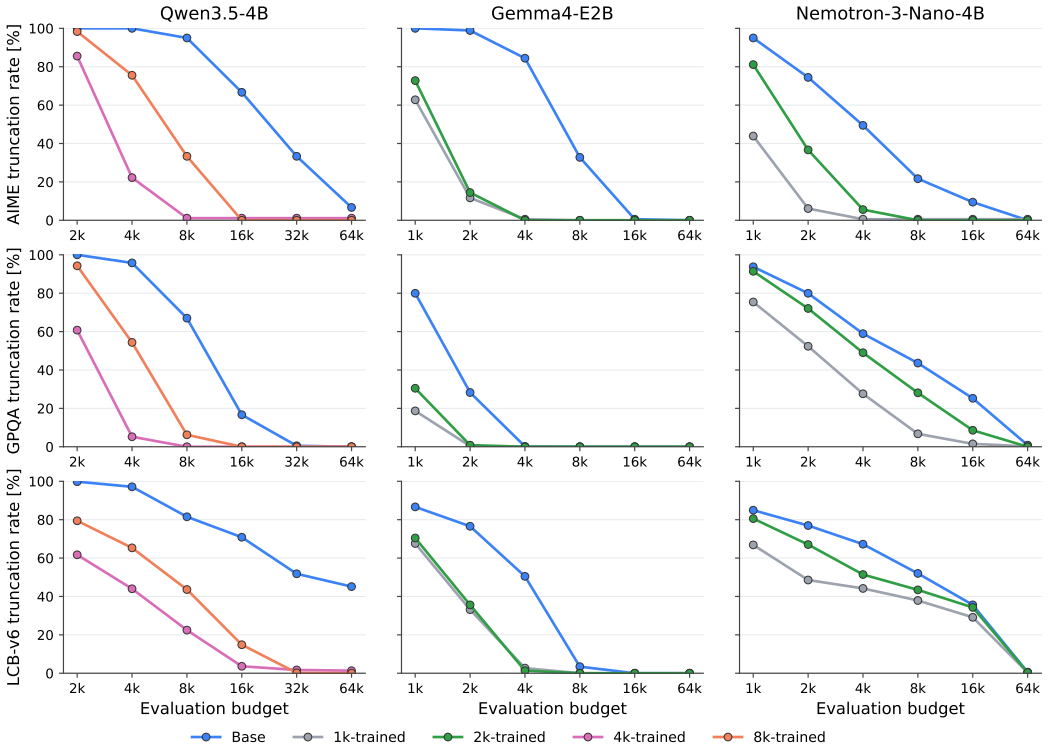


Figure 19: Truncation rates across evaluation budgets on AIME, GPQA-Diamond, and LiveCodeBench v6. Fixed-budget training substantially reduces truncation, particularly near and above the training budget.

Table 12: Exact truncation rates (%) underlying Figure 19.

Qwen3.5-4B

Checkpoint	2k	4k	8k	16k	32k	64k
AIME						
Base	100.00	100.00	95.00	66.67	33.33	6.67
4k-trained	85.56	22.22	1.11	1.11	1.11	1.11
8k-trained	98.33	75.56	33.33	0.00	0.00	0.00
GPQA-Diamond						
Base	100.00	95.79	67.00	16.67	0.51	0.00
4k-trained	60.77	5.22	0.00	0.00	0.00	0.00
8k-trained	94.28	54.38	6.23	0.00	0.00	0.00
LiveCodeBench v6						
Base	99.81	97.14	81.52	70.86	51.81	45.14
4k-trained	61.71	44.00	22.48	3.62	1.71	1.33
8k-trained	79.43	65.33	43.62	14.86	0.19	0.00

Gemma4-E2B

Checkpoint	1k	2k	4k	8k	16k	64k
AIME						
Base	100.00	98.89	84.44	32.78	0.56	0.00
1k-trained	62.78	11.67	0.56	0.00	0.00	0.00
2k-trained	72.78	14.44	0.00	0.00	0.00	0.00
GPQA-Diamond						
Base	79.97	28.28	0.17	0.00	0.00	0.00
1k-trained	18.69	0.34	0.00	0.00	0.00	0.00
2k-trained	30.47	0.84	0.00	0.00	0.00	0.00
LiveCodeBench v6						
Base	86.67	76.57	50.48	3.43	0.00	0.00
1k-trained	67.62	33.14	2.67	0.00	0.00	0.00
2k-trained	70.48	35.62	1.33	0.00	0.00	0.00

Nemotron-3-Nano-4B

Checkpoint	1k	2k	4k	8k	16k	64k
AIME						
Base	95.00	74.44	49.44	21.67	9.44	0.00
1k-trained	43.89	6.11	0.56	0.56	0.56	0.56
2k-trained	81.11	36.67	5.56	0.00	0.00	0.00
GPQA-Diamond						
Base	93.77	79.97	58.92	43.60	25.25	0.84
1k-trained	75.42	52.36	27.61	6.73	1.52	0.17
2k-trained	91.41	72.05	48.99	28.11	8.59	0.00
LiveCodeBench v6						
Base	84.95	76.95	67.24	52.00	35.62	0.57
1k-trained	66.86	48.57	44.19	37.90	29.14	0.38
2k-trained	80.57	67.05	51.43	43.43	34.29	0.38

F TRANSFER OUTPUT LENGTHS

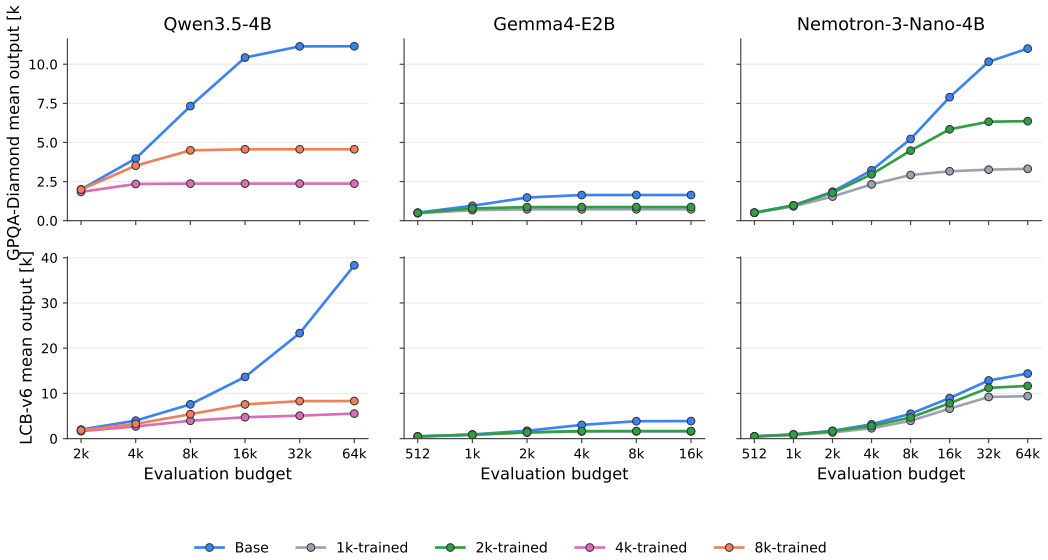


Figure 20: Mean output-length response curves on GPQA-Diamond and LiveCodeBench v6 after fixed-budget mathematical-reasoning training.

G PASS@3 RESPONSE CURVES

Figures 21–23 report pass@3 across evaluation budgets for each model family. The metric counts a problem as solved when at least one of three sampled responses is correct. It preserves the main-text pattern: fixed-budget training gives large gains at small budgets, while above-budget gains are more limited and depend on the task.

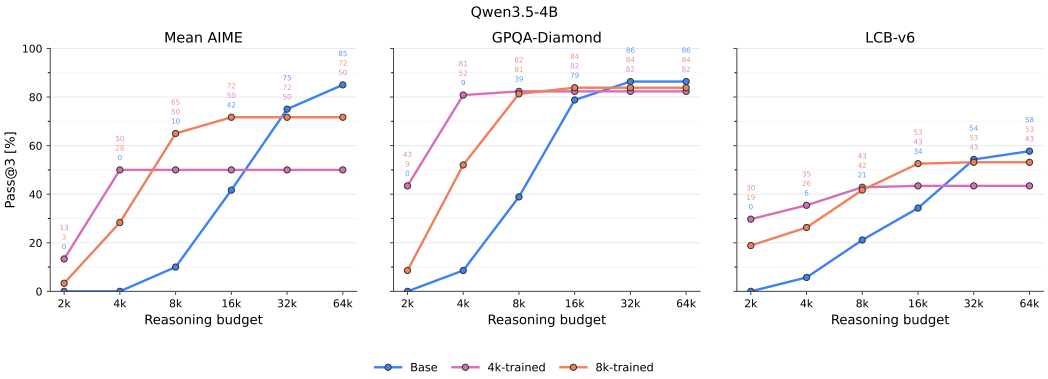


Figure 21: Pass@3 response curves for Qwen3.5-4B on AIME, GPQA-Diamond, and LiveCodeBench v6.

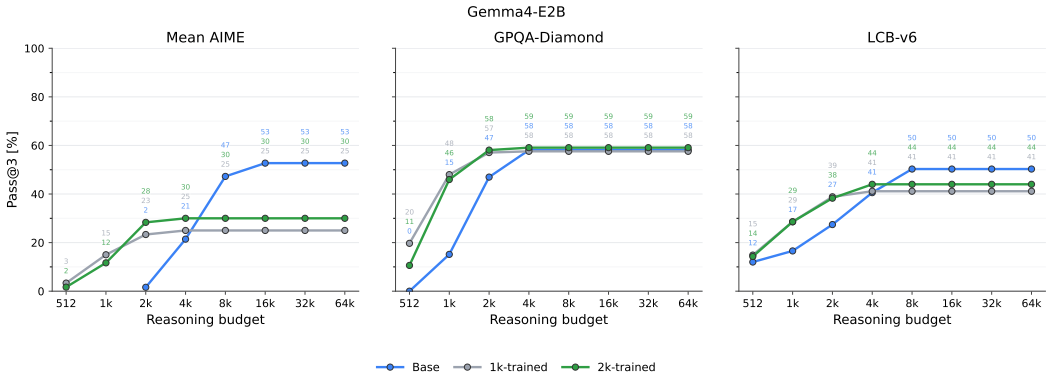


Figure 22: Pass@3 response curves for Gemma4-E2B on AIME, GPQA-Diamond, and LiveCodeBench v6.

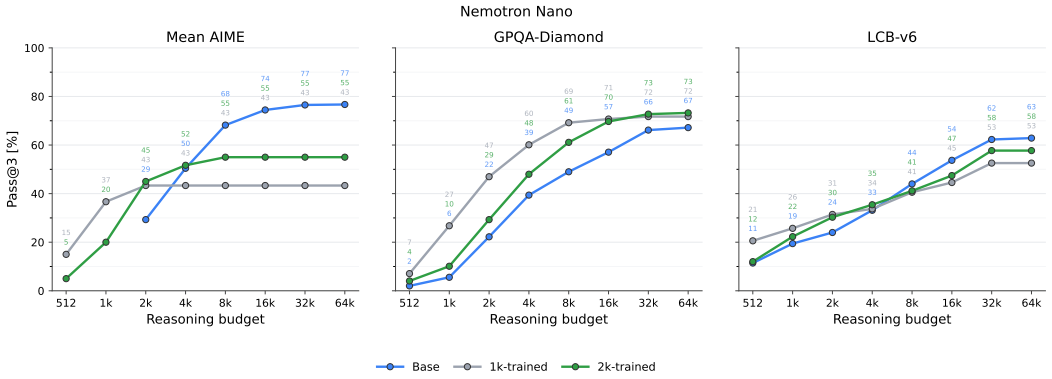


Figure 23: Pass@3 response curves for Nemotron-3-Nano-4B on AIME, GPQA-Diamond, and LiveCodeBench v6.

H BUDGET FORCING ACROSS MODEL FAMILIES

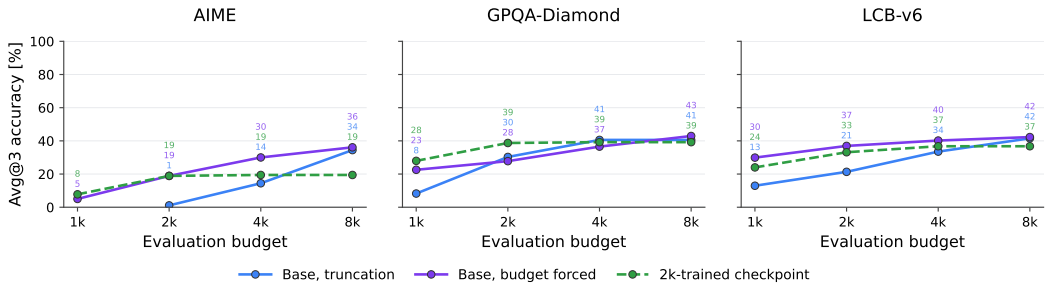


Figure 24: Gemma4-E2B accuracy across evaluation budgets. Each panel compares ordinary base-model truncation, budget-forced completion of the base model, and the 2k fixed-budget-trained checkpoint.

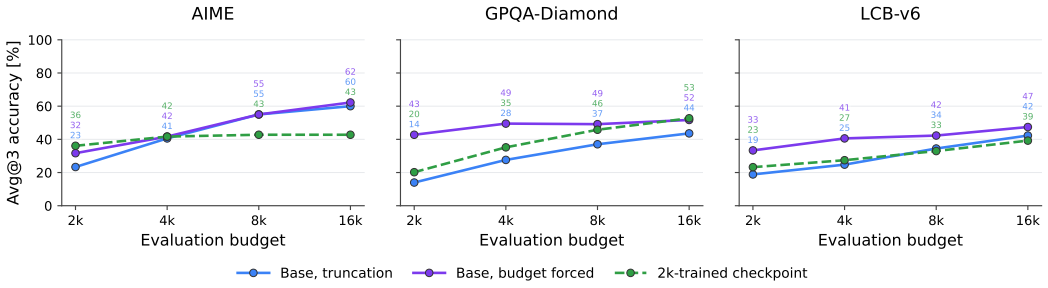


Figure 25: Nemotron-3-Nano-4B accuracy across evaluation budgets. Each panel compares ordinary base-model truncation, budget-forced completion of the base model, and the 2k fixed-budget-trained checkpoint.