CONTROLLING LENGTH GROWTH IN REINFORCEMENT LEARNING FOR CODE WITH ADAPTIVE BUDGETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) improves the reasoning accuracy of large language models but also inflates response length. We study how to curb this growth without suppressing useful reasoning, in competitive programming with training contexts of up to 256k tokens. We find that length grows most, proportionally, on problems the initial model already solves reliably, while accuracy gains concentrate on hard problems. Length penalties are a common remedy, but their shape and strength do not map directly to response length, which makes them hard to tune. Explicit rollout budgets instead limit length in tokens, yet penalizing truncated responses under a fixed budget shortens outputs at the expense of hard problems. Since easy and hard problems call for very different lengths, no single budget serves both. We therefore propose adaptive budgets, which set a separate token limit for each prompt. Each prompt is capped at a multiple of the initial model's median successful length, rarely solved prompts keep the full budget, and responses truncated at the cap are masked from the loss rather than penalized. Compared with a fixed-budget baseline, adaptive budgets cut mean response length by over 30% on LiveCodeBench with no loss in accuracy, and by 25% on the harder LiveCodeBench-Pro at a 1.5-point accuracy cost, improving on the accuracy–length tradeoff of tuned length penalties. Savings are largest on initially solved problems and transfer to AIME 2025/2026, with about 15% fewer tokens at similar accuracy.

1 INTRODUCTION

Extended reasoning helps large language models (LLMs) solve complex problems in mathematics and programming (DeepSeek-AI, 2025; Yang et al., 2026; Liu et al., 2025d). Intermediate steps let a model explore candidate solutions, check assumptions, and revise its approach. Reinforcement learning with verifiable rewards (RLVR) can reinforce these behaviors, improving accuracy while increasing response length (DeepSeek-AI, 2025). This growth raises training and inference costs and can produce *overthinking*: additional reasoning with little benefit, especially on problems that short responses already solve (Chen et al., 2024). Because correctness rewards do not directly penalize redundant steps, a model can become more accurate without becoming more efficient.

Prior work controls reasoning length during RL with explicit budgets or reward shaping. Some budget methods train models to follow a token allowance given in the prompt, leaving the choice of allowance to the user (Aggarwal & Welleck, 2025); others impose a single rollout limit on all training prompts, which can reduce accuracy when the limit is too tight (Hou et al., 2025; Wang et al., 2025). Reward-shaping methods add a length penalty to the reward to favor short, correct responses (Kimi Team, 2025; Cursor Research Team, 2026; Microsoft AI, 2026), and adaptive variants adjust the penalty to problem difficulty using solve rates or response-length statistics (Xiang et al., 2025; Liu et al., 2025b). However, a penalty's shape and strength do not directly specify how much responses will shorten: reaching a target reduction requires empirical tuning, and aggressive compression can sacrifice accuracy (Nohara et al., 2026).

Length control matters especially in competitive programming, where designing and verifying algorithms can require very long reasoning. Recent code-RL systems train with budgets of 120k tokens (Yang et al., 2026; Zhang et al., 2026), whereas representative length-control studies use limits of

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

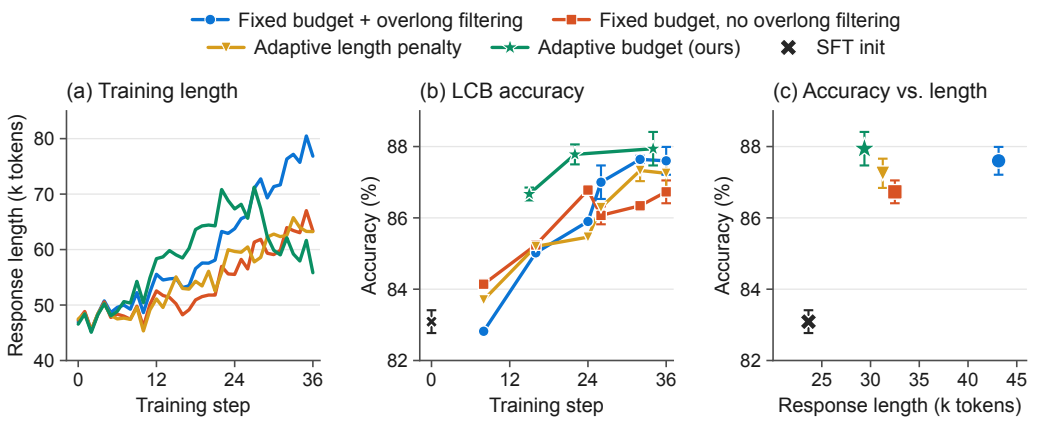

Figure 1: **Length growth and LiveCodeBench (LCB) accuracy across training interventions.** (a) Mean training-response length and (b) LCB accuracy (pass@1) through step 36. (c) Accuracy versus mean evaluation length for the checkpoints in Table 1. Fixed budgets use a 102k-token context window. Overlong filtering masks truncated responses from the loss; without it, they are scored like any other response. SFT init is the initial model. Error bars: one standard error across decoding seeds, where available; one training run per configuration.

around 16k tokens (Nohara et al., 2026; Wu et al., 2026a). Even at these long budgets, the rollout limit can cut off useful reasoning: some correct solutions to difficult problems exceed 100k tokens (§4.3). Both the limit and the treatment of truncated responses therefore affect whether long successful reasoning is preserved. Yet length growth and its control at this scale remain understudied.

To address this gap, we study long-reasoning RL for competitive programming. Our *baseline* adapts the Nemotron-Cascade 2 recipe (Yang et al., 2026): a fixed 102k-token context window with *overlong filtering*, which masks truncated responses from the policy loss (Yu et al., 2025). As Figure 1a,b shows, this baseline (*Fixed budget + overlong filtering*) improves coding accuracy but also produces substantially longer responses. We ask two questions: *(i) Where is length growth concentrated, and what learning signals contribute to it?* and *(ii) How can training achieve a better balance between reasoning accuracy and response length?*

For question (i), we find that the largest proportional growth occurs on problems the initial model already solves reliably, whereas accuracy gains concentrate on hard problems (§4.1). This suggests constraining length where the initial model already succeeds. By ablating the sign of the updates, we further show that *negative-only updates*, which only discourage failed responses, can produce early length growth and part of the coding gain before training collapses (§4.2).

For question (ii), we first ablate two basic design choices: whether to use overlong filtering and whether to raise the maximum budget (§4.3). We observe that penalizing truncated responses, instead of filtering them, shortens outputs but loses long successful solutions to difficult problems, and that simply raising the budget to 256k does not improve on the baseline. Together, these findings suggest three principles for selective length control: limit length where the initial model already succeeds, keep room for long attempts on difficult problems, and mask truncated responses rather than penalize them.

We implement these principles as *adaptive budgets* (§5). Before training, we measure and freeze each prompt's solve rate and median successful length under the initial model. During RL, we cap each prompt at this median length times a common multiplier, so prompts with longer successful solutions get more room, and responses truncated at this *adaptive cap* are masked from the loss. A solve-rate gate keeps the full 256k budget for rarely solved prompts. Our recipe thus expresses limits in tokens, relative to demonstrated solution lengths, which makes them easier to interpret than reward-shaping coefficients.

Figure 1 compares adaptive budgets with fixed budgets under both truncation treatments and with an adaptive length penalty whose strength depends on the solve rate (Microsoft AI, 2026). On

LiveCodeBench (LCB), adaptive budgets give shorter responses and higher measured accuracy than every other RL variant shown (Figure 1c).

In summary, our contributions are:

- We characterize where length grows in long-reasoning code RL, relative to where accuracy improves, and which update signals can produce this growth (§4.1–4.2).
- We show that both budget size and the treatment of truncated responses shape the balance between shorter responses and successful long reasoning (§4.3).
- We develop adaptive budgets (§5), which lie on the accuracy–length Pareto frontier: no fixed budget or length penalty we compare is both shorter and more accurate on either coding benchmark (§6.2). Relative to the baseline, our method reduces mean response length by over 30% on LCB with no accuracy loss, and by about 25% on the harder LiveCodeBench-Pro at a 1.5-point accuracy cost. The reduction is strongest on initially solved problems and transfers to AIME 2025 and 2026 with about 15% fewer tokens at similar measured accuracy (§6.3–6.4), and the recipe is also effective on another model family, Qwen3.5-4B (§6.5).

2 Related Work

Explicit budgets and truncation handling. L1 trains models to follow prompt-supplied token allowances (Aggarwal & Welleck, 2025), and ThinkPrune progressively tightens a training limit shared across prompts (Hou et al., 2025). In mathematical RL, DA-DLER and TRiMS instead choose each prompt's limit (2k or 4k tokens) from its current rollouts (Liu et al., 2025a; Bian et al., 2026). In competitive programming, Zhang et al. (2026) study randomized rollout caps and training budgets reaching 120k tokens. Relative-budget theory relates training budgets to initial-policy solution times in a stylized model (Wachi et al., 2026). Truncated attempts can be scored like other failures or masked from the loss (overlong filtering). DAPO studies overlong filtering and a soft penalty near the limit (Yu et al., 2025); DeepCoder credits overlong filtering for its success beyond the training budget (Luo et al., 2025b). Masking ablations in Skywork-OR1 and AceReason 1.1 show recipe- and stage-dependent effects in mathematics, with AceReason also reporting code transfer (He et al., 2025; Liu et al., 2025e). The Art of Efficient Reasoning jointly studies rollout limits, reward shaping, and masking in mathematical RL up to 16k tokens, evaluating transfer to code (Wu et al., 2026a). We compare both treatments in code RL at a fixed 102k budget and test a 256k budget; the findings motivate adaptive budgets (§5). Unlike DA-DLER and TRiMS, adaptive budgets fix each prompt's cap before training, as a multiple of the initial model's median successful length, rather than choosing it from current rollouts.

Length penalties and adaptive rewards. Reward shaping encourages brevity with length-dependent terms, such as group-relative length rewards in Kimi k1.5 (Kimi Team, 2025), concave costs in Composer 2 (Cursor Research Team, 2026), and pass-rate-weighted penalties in MAI-Thinking-1 (Microsoft AI, 2026). Adaptive methods adjust this pressure to the problem: ALP uses sampled solve rates (Xiang et al., 2025), LASER combines difficulty groups with response-length statistics (Liu et al., 2025b), and Apriel-Reasoner applies difficulty-aware penalties in code-inclusive RL (Pardinas et al., 2026). Reference lengths provide a complementary signal. O1-Pruner rewards responses shorter than frozen initial-model outputs, subject to accuracy, in PPO-style off-policy training (Luo et al., 2025a), whereas LAPO updates per-prompt successful-length medians for reward shaping and training-time budget guidance (Wu et al., 2025). Dynamic Outlier Truncation reshapes rewards by trimming unusually long responses after generation in all-correct groups (Wu et al., 2026b). These signals express preferences over lengths, but their parameters do not directly specify the learned response length; controlled sweeps show sensitivity to penalty strength and accuracy losses under aggressive compression (Nohara et al., 2026).

Many prior comparisons also rely on released checkpoints trained with different data or budgets (e.g., Xiang et al., 2025; Yi et al., 2025), whereas the controlled studies of Nohara et al. (2026) and Wu et al. (2026a) train with at most 16k tokens. Our length-penalty baselines share the initial model, training data, and optimization settings with adaptive budgets, and all of these runs use 102k–256k windows. Adaptive budgets yield shorter responses than both length penalties on both coding benchmarks at similar or higher accuracy (§6.2).

3 PRELIMINARIES AND EXPERIMENTAL SETUP

Reinforcement learning with verifiable rewards. A language model acts as a policy π_θ that samples a response y to a problem x. A verifier assigns an outcome reward $r(x, y)$; in competitive programming, a response contains reasoning and a final program, which receives reward one if it passes the tests and zero otherwise. RL maximizes the expected reward $J(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}[r(x, y)]$. Group Relative Policy Optimization (GRPO; Shao et al., 2024) estimates advantages by comparing G responses $y_1, \ldots, y_G$ sampled for the same problem from the rollout policy π_{old}. Following Liu et al. (2025c), we center rewards without dividing by their standard deviation:

$$A_i = r_i - \bar{r}, \qquad \bar{r} = \frac{1}{G}\sum_{j=1}^{G} r_j, \qquad r_i = r(x, y_i). \tag{1}$$

Each update maximizes a token-level clipped surrogate (Yu et al., 2025):

$$\mathcal{J}_{\text{clip}}(\theta) = \frac{1}{\sum_i m_i|y_i|}\sum_i m_i \sum_{t=1}^{|y_i|} \min\Big(\rho_{i,t}A_i,\ \text{clip}\big(\rho_{i,t}, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\big)A_i\Big), \tag{2}$$

where i ranges over the responses in the batch, each with the advantage from its own group, $\rho_{i,t} = \pi_\theta(y_{i,t} \mid x_i, y_{i,<t})/\pi_{\text{old}}(y_{i,t} \mid x_i, y_{i,<t})$ is the token-level importance ratio, and $m_i \in \{0, 1\}$ indicates whether response i enters the loss ($m_i = 1$ by default). A positive advantage reinforces all tokens of its response, and a negative one discourages them. A failed response receives a negative advantage when another response in its group succeeds.

Setup. Our initial model is a supervised fine-tuning (SFT) checkpoint of a Nemotron-H hybrid mixture-of-experts model with 31.6B total and 3.6B active parameters (NVIDIA, 2025). Every run trains it on the same sequence of 3,532 competitive-programming prompts (Yang et al., 2026) from multiple sources, including Codeforces and AtCoder. Our baseline adapts the code-RL recipe of Nemotron-Cascade 2 (Yang et al., 2026): binary verifier rewards, a fixed 102k-token budget for every prompt, and *overlong filtering*, which sets $m_i = 0$ for truncated responses. Appendix A.1 gives the remaining training details.

Evaluation. We evaluate coding on LiveCodeBench (LCB; Jain et al., 2025), using its v5 and v6 problems released between August 2024 and May 2025, and on LiveCodeBench-Pro problems from 2025 Q1–Q2 (LCBPro; Zheng et al., 2025). Transfer to mathematics is measured on AIME 2025 and 2026. We report pass@1 averaged over 8 samples per problem on LCB and LCBPro and over 16 on AIME (avg@8 and avg@16), together with the mean number of generated tokens over all responses. All evaluations use a 256k-token context window; Appendix A.3 gives the decoding settings. Tables 2 and 5 use LCBPro's medium tier (LCBPro medium), where runs differ most.

4 LENGTH GROWTH AND TRUNCATION IN CODE RL

4.1 WHERE DOES LENGTH GROW?

We compare the response lengths and accuracy of the SFT model and the baseline on LCB, grouping problems in two ways. *Initial-solvability* groups split problems into those the SFT model solves always (initially solved), sometimes (partially solved), or never (unsolved). *Difficulty* tiers use LCB's easy, medium, and hard labels. For each group, we measure the proportional increase in mean response length and the group's share of the net added tokens.

Much of the growth occurs where the initial model already succeeds, consistent with reports of growing verbosity on easy inputs in mathematical RL (Wu et al., 2026b). Initially solved problems grow most in proportion, more than doubling in length (Figure 2a). Their large ratio also partly reflects a short starting point: the SFT model already adapts length to difficulty and answers these problems briefly. RL thus spends tokens where brief answers already succeeded, eroding part of this adaptivity. Initially unsolved problems grow least in proportion, and after RL their mean length reaches the 102k training window.

Accuracy gains follow a different pattern. Mean length roughly doubles on easy and medium problems, yet easy-problem accuracy stays near its ceiling; on hard problems, length grows proportionally less, but accuracy gains the most (Figure 2b,c).

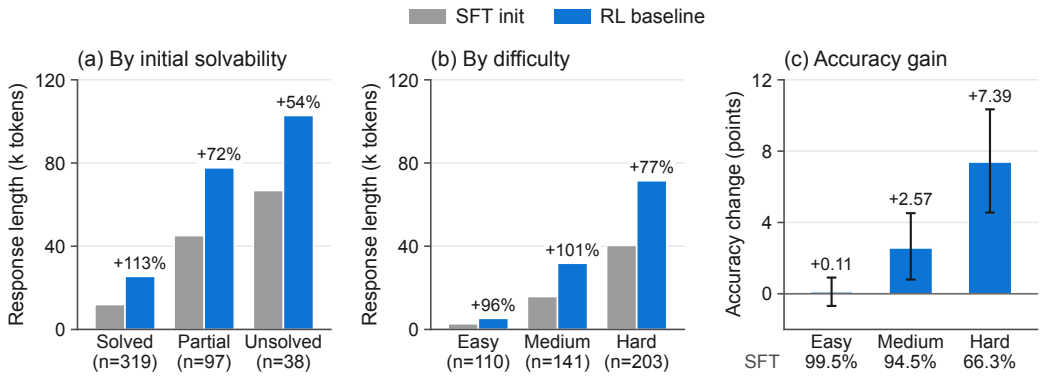

Figure 2: **LCB length grows most in proportion on initially solved problems, while accuracy gains concentrate on hard problems.** (a,b) Mean response length of the SFT model and the baseline, grouped by initial solvability or benchmark difficulty; annotations give percentage increases. Solved, partial, and unsolved denote 4/4, 1–3/4, and 0/4 SFT successes and account for 49%, 36%, and 15% of the net added tokens. (c) Accuracy change (baseline minus SFT) by difficulty tier; percentages below the tier names give SFT accuracy.

Longer traces contain more reasoning steps of similar size. To see what fills the added tokens, we split reasoning traces into paragraphs (Appendix B.2). After RL, the number of paragraphs roughly doubles while their size stays similar, the final answer becomes shorter, and episodes with checking or testing cues become more numerous with little change in length.

4.2 What Drives Length Growth?

Correct responses are not longer. A natural hypothesis is that longer responses succeed more often and are thus reinforced more. Among early training rollouts for the same prompt, however, correct responses are shorter than incorrect ones on average, partly because truncated responses almost always fail, and the gap persists among responses that finish within the training window (Appendix B.1). To find what drives growth, we therefore intervene on training updates.

Both update signs lengthen responses, negative ones faster. We separate the two parts of the correctness signal by training with only positive-advantage updates, which reinforce successful rollouts, or only negative-advantage updates, which discourage failed ones; advantages are computed from the full group before either sign is dropped. To isolate the sign from truncation handling, both runs use a 256k window, where their truncation is negligible (at most 0.1% of responses). Negative-only updates lengthen responses faster than positive-only updates (Figure 3a) and nearly as fast as full updates (Appendix B.3), and they improve LCB accuracy over the SFT model, in line with prior analyses of negative updates (Fatemi et al., 2025; Setlur et al., 2025; Zhu et al., 2025); positive-only updates yield a similar gain (Figure 3c). Negative-only training is unstable, however: after about 20 steps, training reward drops sharply and length falls (Figure 3a,b), consistent with negative gradients suppressing the likelihood of correct responses (Deng et al., 2025). Yet the sign is not the full story: applying the same total negative advantage to correct rollouts instead lengthens responses less than half as fast (Appendix B.3), suggesting that most of the growth comes from negative updates on failed rollouts rather than from negative updates in general.

4.3 Training Budgets and Overlong Filtering

A rollout limit turns unfinished attempts into failures, and truncation handling decides whether they receive negative updates (§4.2). By default ($m_i = 1$), a truncated response, which almost always

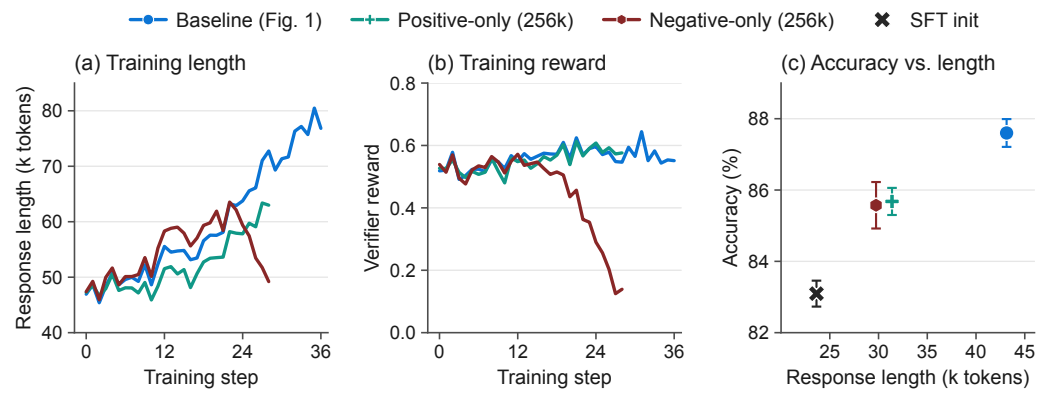

Figure 3: **Negative-only updates produce early length growth and a coding gain, followed by declining training reward.** (a) Mean training-response length and (b) verifier reward of the 256k sign-restricted runs, with the 102k baseline of Figure 1 as a reference (its window and filtering differ). (c) LCB accuracy versus mean response length for the SFT model, the baseline, and the best evaluated checkpoint of each sign-restricted run. Error bars: one standard error across decoding seeds.

fails verification, is scored like any other failure. We compare training with and without overlong filtering at the same 102k window and test whether a larger 256k window helps; all models are evaluated at 256k, and we analyze LCBPro, where the treatments differ most.

Filtering preserves accuracy on difficult problems. Without overlong filtering, responses are about 20% shorter overall and shorter in every difficulty tier (Figure 4a), but accuracy is 3.4 points lower. To locate the lost accuracy, we decompose accuracy by completion length (Figure 4b). Counting only successes that finish within the 102k training window, the two runs match. The gap opens beyond the window, where filtering retains many more successes, and it falls mostly on medium problems (Figure 4c). Penalizing truncated responses thus costs accuracy not by shortening responses per se but by losing long successes on challenging, partly solvable problems.

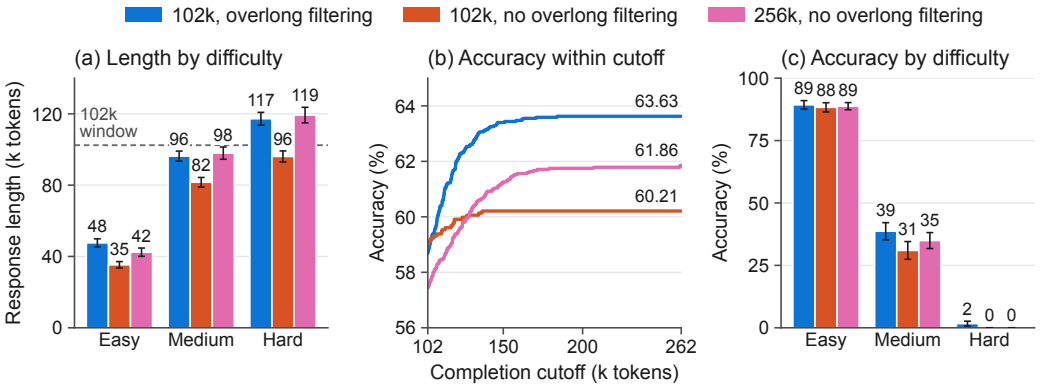

Figure 4: **On LCBPro, filtering preserves long successes, and a larger training window does not close the gap.** Runs train with a 102k window with or without overlong filtering, or with a 256k window without it (mean lengths 71.0k, 57.1k, and 68.8k tokens). (a) Mean response length by difficulty; dashed line: 102k training window. (b) Accuracy from responses that finish within each completion cutoff, starting at 102,400 tokens and including all shorter successes; this decomposes the 256k evaluation rather than re-running inference. (c) Accuracy by difficulty. Error bars: one standard error.

A larger fixed budget does not guarantee better learning. A larger window could instead give unfinished attempts room to succeed, so we also train with a 256k fixed budget without overlong

filtering. Its responses are slightly longer than the baseline's on medium and hard problems (Figure 4a), yet it trails the baseline by 1.8 points (Figure 4b,c). Its training is also more unstable, possibly because noisy rollouts arise more easily in longer contexts, which could be one reason why prior work extends the context in stages (Chen et al., 2025; Luo et al., 2025b; He et al., 2025; Microsoft AI, 2026). A 256k run with overlong filtering behaves similarly, likely because truncation at this window is rare (Appendix C.4), so we omit that run in the results.

These results suggest three principles: (i) limit growth where the initial model already succeeds (§4.1); (ii) preserve room for long attempts on difficult problems; and (iii) mask rather than penalize the attempts that tighter limits truncate.

5 Adaptive Budgets from Initial-Model Solutions

Adaptive budgets implement these principles with two components: a per-prompt cap derived from the initial model, which serves (i) and (ii), and overlong filtering at that cap, which serves (iii).

Budget construction. We sample K_{ref} responses per training prompt x from the initial model π_0 and freeze their solve rate $\widehat{p}_0(x)$ and median successful length $L_{\text{ref}}(x)$, so budgets do not expand with the policy's responses. With multiplier α, solve-rate gate τ, prompt length $P(x)$, and context ceiling C_{high}, the generation budget is

$$B(x) = \begin{cases} \min\{\lfloor \alpha L_{\text{ref}}(x) \rfloor,\, C_{\text{high}} - P(x)\}, & \widehat{p}_0(x) > \tau, \\ C_{\text{high}} - P(x), & \text{otherwise}, \end{cases} \tag{3}$$

where $\lfloor \alpha L_{\text{ref}}(x) \rfloor$ is the adaptive cap. The gate preserves room for long attempts on rarely solved problems, including those without a success, and avoids high-variance caps from few successes. The reference pass is a one-time, parallelizable, generation-only cost below that of RL training, and common pipelines already filter training prompts by the initial model's pass rate (Kimi Team, 2025; He et al., 2025), so the solve rates and successful lengths can be taken directly from that pass.

Responses stopped by the adaptive cap are masked from the loss ($m_i = 0$), as in overlong filtering, and the caps apply only during training. Algorithm 1 (Appendix C.1) summarizes the procedure.

6 Experiments

6.1 Configuration and Baselines

Adaptive budgets. We sample $K_{\text{ref}} = 16$ reference responses per prompt and use $\alpha = 1.75$, $\tau = 0.25$, and a 256k context ceiling.

Baselines. Besides the fixed budgets of §4.3 (the baseline, and the 102k and 256k budgets without overlong filtering), we compare two length penalties. The *linear length penalty*, related to Kimi k1.5 (Kimi Team, 2025), adds a group-normalized length term to the rewards of all-correct groups. The *adaptive length penalty* combines prior ideas: its strength depends on the solve rate, as in ALP and MAI-Thinking-1 (Xiang et al., 2025; Microsoft AI, 2026), and, as in ShorterBetter and Short-RL (Yi et al., 2025; Yuan et al., 2025), it penalizes length relative to the shortest correct response in each group (Appendix A.2). All methods share the baseline's SFT checkpoint, training data, and settings. For each method, we report the checkpoint with the best LCB accuracy.

6.2 Coding Accuracy and Response Length

Adaptive budgets use 32% fewer tokens than the baseline on LCB at similar accuracy (Table 1), even with a larger mean per-prompt training budget (112k vs. 102k tokens; Appendix C.1). On LCB, all RL-trained methods reach similar accuracy, so they differ mainly in length, and adaptive budgets give the shortest responses: 6% shorter than the adaptive length penalty and 16% shorter than the linear one. On LCBPro, adaptive budgets use about 25% fewer tokens than the baseline at a 1.5-point cost, whereas every other method loses more accuracy and shortens responses less.

Training intervention	LiveCodeBench		LiveCodeBench-Pro	
	Acc. (%)	Tokens (k)	Acc. (%)	Tokens (k)
SFT init	$83.09_{\pm 0.56}$	23.6	$55.71_{\pm 0.76}$	44.6
Fixed budget (102k) + overlong filtering (baseline)	$87.61_{\pm 0.29}$	43.1	$\mathbf{63.63}_{\pm 0.51}$	71.0
Fixed budget (102k), no overlong filtering	$86.73_{\pm 0.42}$	$32.5_{-24.7\%}$	$60.21_{\pm 0.67}$	$\underline{57.1}_{-19.6\%}$
Fixed budget (256k), no overlong filtering	$87.17_{\pm 0.45}$	$39.2_{-9.1\%}$	$61.86_{\pm 0.55}$	$68.8_{-3.0\%}$
Linear length penalty	$\underline{87.72}_{\pm 0.38}$	$35.1_{-18.6\%}$	$61.22_{\pm 0.31}$	$64.4_{-9.3\%}$
Adaptive length penalty	$87.25_{\pm 0.30}$	$\underline{31.3}_{-27.5\%}$	$61.00_{\pm 0.48}$	$58.7_{-17.3\%}$
Adaptive budget (ours)	$\mathbf{87.94}_{\pm 0.42}$	$\mathbf{29.4}_{-31.9\%}$	$\underline{62.09}_{\pm 0.49}$	$\mathbf{53.6}_{-24.5\%}$

Table 1: **Coding accuracy and response length.** Token subscripts: change relative to the baseline. **Bold**/underline: best/second best among RL-trained methods.

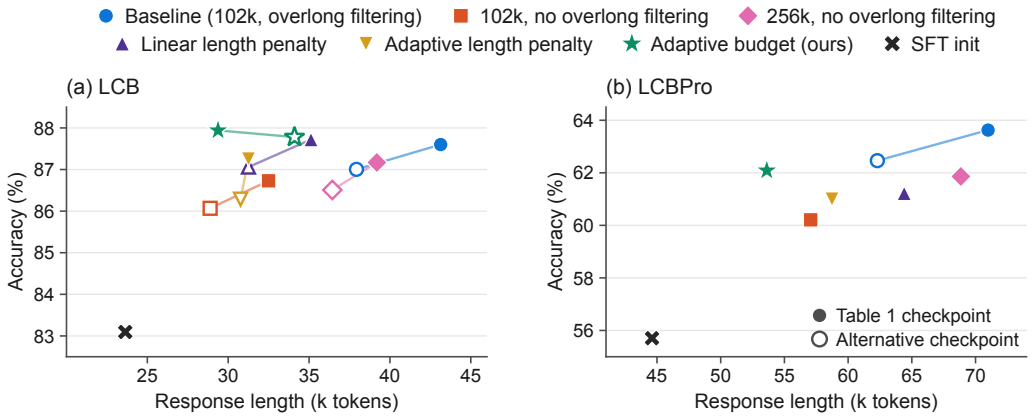

Figure 5: **Adaptive budgets lie on the Pareto frontier: no plotted checkpoint of another method is both shorter and more accurate.** Accuracy against mean response length on (a) LCB and (b) LCBPro. Filled markers are the checkpoints in Table 1, hollow markers joined to them are alternative checkpoints, and the cross is the SFT model.

Figure 5 plots accuracy against response length. Because the choice of checkpoint also trades accuracy for length, the figure adds alternative checkpoints of each run. In both panels, adaptive budgets lie on the Pareto frontier, toward the upper left.

6.3 Where Are Responses Shortened?

Adaptive budgets shorten responses most where the baseline grew most in proportion (Figure 7, Appendix C.2). Relative to the baseline, our method removes 38% of tokens on initially solved LCB problems but 21% on initially unsolved ones (32% and 19% on LCBPro). Accuracy on initially solved problems is unchanged, whereas LCBPro's accuracy loss falls on partially solved and unsolved problems.

6.4 Transfer to Mathematical Reasoning

On AIME 2025 and 2026, outside the RL training domain, adaptive budgets use 15% fewer tokens than the baseline (32.2k vs. 37.8k) at similar accuracy (92.19% vs. 92.08%; the SFT model reaches 91.77% with 28.2k tokens), so the learned shortening transfers beyond code.

6.5 Another Model Family

We rerun the baseline, the 102k run without overlong filtering, and adaptive budgets on Qwen3.5-4B (Qwen Team, 2026) with the same data and settings except a looser $\alpha = 2.5$, as its baseline grows far more (21.2k to 102.8k LCB tokens). Adaptive budgets use 36% fewer LCB tokens than

the baseline at similar accuracy (70.59% vs. 70.04%) and 20% fewer on LCBPro medium (8.97% vs. 9.19%), whereas removing overlong filtering costs 7.0 LCB points (Table 5, Appendix C.3).

6.6 Design Variants

	LiveCodeBench		LCBPro medium	
Training variant	Acc. (%)	Tokens (k)	Acc. (%)	Tokens (k)
Baseline	87.61	43.1	38.68	96.3
Adaptive budget	87.94	29.4	35.90	75.8
Adaptive budget w/o gate	87.09	28.4	32.48	72.5
Adaptive budget w/o overlong filtering	86.43	27.6	32.37	72.4
Adaptive budget ($\alpha = 1.5$)	86.65	27.4	33.55	71.6
Adaptive budget ($\alpha = 2.0$)	87.56	38.9	38.46	96.8
Two-level budget (74.4k / 256k)	86.12	30.4	30.13	74.8

Table 2: **Design variants of adaptive budgets.**

Table 2 changes one component of adaptive budgets at a time. Removing the solve-rate gate shortens responses slightly but lowers accuracy on both benchmarks, most on LCBPro medium (3.4 points). Scoring capped responses as failures instead of masking them also shortens responses but costs 1.5 points on LCB and 3.5 on LCBPro medium, consistent with §4.3. The multiplier sets the tradeoff: $\alpha = 1.5$ lowers accuracy on both benchmarks, whereas $\alpha = 2$ keeps the baseline's accuracy but barely shortens responses. Unlike a penalty coefficient, α can be calibrated before training by the share of initial rollouts its caps stop: $\alpha = 1.5$, 1.75, and 2 stop 10.4%, 4.9%, and 2.3%, and their checkpoints' LCB responses are 16%, 25%, and 65% longer than the initial model's. Length responds nonlinearly to α, but the stopped share, estimable from the reference samples, gives a practical tuning target (Appendix C.4). Finally, we test a two-level budget, similar to DA-DLER (Liu et al., 2025a), which keeps the solve-rate gate and masking but replaces per-prompt caps with one shared cap: prompts above the gate receive the mean adaptive cap (74.4k tokens), and the rest keep the full 256k window. It shortens responses about as much as adaptive budgets but loses 1.8 points on LCB and 5.8 on LCBPro medium, so per-prompt scaling preserves accuracy.

Training cost. Adaptive budgets do not lengthen training steps: over the first 12 steps, when training responses average about 50k tokens in both runs, a step takes 38.4 minutes versus 38.3 for the baseline. With shorter rollouts late in training, adaptive budgets are also slightly faster overall, taking 27.2 hours for the first 36 steps versus 28.3 for the baseline.

7 Discussion and Conclusion

Our study contributes both an analysis and a method for the accuracy–length tradeoff in long-reasoning code RL. The analysis shows where and why length grows: RL adds much of its extra length to problems the initial model already solves, while accuracy gains concentrate on hard problems; negative updates on failed rollouts alone can drive early growth; and penalizing truncated responses loses long successes on difficult problems, while enlarging the fixed budget does not always help. Adaptive budgets turn these findings into a recipe that caps each prompt at a multiple of the initial model's median successful length, except rarely solved prompts, and masks responses truncated at the cap. Our method uses over 30% fewer tokens than the baseline on LCB and 15% fewer on AIME at similar accuracy, and about 25% fewer on LCBPro at a 1.5-point cost, shortens responses most on initially solved problems, carries over to Qwen3.5-4B, and yields shorter responses than the length penalties we tested at similar or higher accuracy.

Limitations and future work. Adaptive budgets limit whole responses and thus do not separate redundant from useful reasoning; the accuracy cost on LCBPro suggests that some useful reasoning is also cut. Separating the two would require identifying and penalizing redundant steps within a response instead of assigning one advantage to all of its tokens, a promising direction for future work. Due to limited compute, we experimented only with relatively small models.

AI USE STATEMENT

In this work, we used generative AI tools to help polish the manuscript, and to search and discover related literature. We did not use generative AI tools to generate data. We have reviewed all AI-assisted work: we read every AI edit to the manuscript and retrieved related work. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix A.1 describe the initial model, training data, objective, and optimization settings. Section 5, Algorithm 1, and Appendix C.1 specify how adaptive budgets are constructed and enforced, and §6.1 gives their settings. Appendix A.2 defines the length-penalty baselines, Appendix B.3 the update-signal controls, §6.5 and Appendix C.3 the Qwen3.5-4B runs, and §6.6 the design variants. Section 3 and Appendix A.3 give the evaluation benchmarks, metrics, and decoding settings. All evaluation benchmarks are public.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling How Long A Reasoning Model Thinks With Reinforcement Learning. *arXiv preprint arXiv:2503.04697*, 2025. URL https://arxiv.org/abs/2503.04697.

Tingcheng Bian, Jinchang Luo, Mingquan Cheng, Jinyu Zhang, Xiaoling Xia, Ni Li, Yan Tao, and Haiwei Wang. TRiMS: Real-Time Tracking of Minimal Sufficient Length for Efficient Reasoning via RL. *arXiv preprint arXiv:2603.17449*, 2026. URL https://arxiv.org/abs/2603.17449.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT Think That Much for 2+3=? On the Overthinking of o1-Like LLMs. *arXiv preprint arXiv:2412.21187*, 2024. URL https://arxiv.org/abs/2412.21187.

Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. AceReason-Nemotron: Advancing Math and Code Reasoning through Reinforcement Learning. *arXiv preprint arXiv:2505.16400*, 2025. URL https://arxiv.org/abs/2505.16400.

Cursor Research Team. Composer 2 Technical Report. *arXiv preprint arXiv:2603.24477*, 2026. URL https://arxiv.org/abs/2603.24477.

DeepSeek-AI. DeepSeek-R1 Incentivizes Reasoning in LLMs through Reinforcement Learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

Wenlong Deng, Yi Ren, Muchen Li, Danica J. Sutherland, Xiaoxiao Li, and Christos Thrampoulidis. On the Effect of Negative Gradient in Group Relative Deep Reinforcement Optimization. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/6adaf0cbeba11705d4ea67a62044f63d-Paper-Conference.pdf.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. Concise Reasoning via Reinforcement Learning. *arXiv preprint arXiv:2504.05185*, 2025. URL https://arxiv.org/abs/2504.05185.

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. *arXiv preprint arXiv:2505.22312*, 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. *arXiv preprint arXiv:2504.01296*, 2025. URL https://arxiv.org/abs/2504.01296.

Naman Jain, King 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 *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Kimi Team. Kimi k1.5: Scaling Reinforcement Learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025. URL https://arxiv.org/abs/2501.12599.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. DLER: Doing Length pEnalty Right - Incentivizing More Intelligence per Token via Reinforcement Learning. *arXiv preprint arXiv:2510.15110*, 2025a. URL https://arxiv.org/abs/2510.15110.

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. *arXiv preprint arXiv:2505.15612*, 2025b. URL https://arxiv.org/abs/2505.15612.

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. *arXiv preprint arXiv:2503.20783*, 2025c. URL https://arxiv.org/abs/2503.20783.

Zihan Liu, Yang Chen, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. AceMath: Advancing Frontier Math Reasoning with Post-Training and Reward Modeling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3993–4015, 2025d.

Zihan Liu, Zhuolin Yang, Yang Chen, Chankyu Lee, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. AceReason-Nemotron 1.1: Advancing Math and Code Reasoning through SFT and RL Synergy. *arXiv preprint arXiv:2506.13284*, 2025e. URL https://arxiv.org/abs/2506.13284.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. O1-Pruner: Length-Harmonizing Fine-Tuning for O1-Like Reasoning Pruning. *arXiv preprint arXiv:2501.12570*, 2025a. URL https://arxiv.org/abs/2501.12570.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepCoder: A Fully Open-Source 14B Coder at O3-mini Level. Notion blog, https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025b.

Microsoft AI. MAI-Thinking-1: Building a Hill-Climbing Machine. Technical report, Microsoft AI, 2026. URL https://microsoft.ai/pdf/mai-thinking-1.pdf.

Daisuke Nohara, Taishi Nakamura, and Rio Yokota. On the Optimal Reasoning Length for RL-Trained Language Models. *arXiv preprint arXiv:2602.09591*, 2026. URL https://arxiv.org/abs/2602.09591.

NVIDIA. Nemotron 3 Nano: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning. *arXiv preprint arXiv:2512.20848*, 2025. URL https://arxiv.org/abs/2512.20848.

Rafael Pardinas, Ehsan Kamalloo, David Vazquez, and Alexandre Drouin. Apriel-Reasoner: RL Post-Training for General-Purpose and Efficient Reasoning. *arXiv preprint arXiv:2604.02007*, 2026. URL https://arxiv.org/abs/2604.02007.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents. https://qwen.ai/blog?id=qwen3.5, 2026.

Amrith Setlur, Matthew Y. R. Yang, Charlie Snell, Jeremy Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to Explore Enables Extrapolation of Test-Time Compute for LLMs. *arXiv preprint arXiv:2506.09026*, 2025. URL https://arxiv.org/abs/2506.09026.

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. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

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. *arXiv preprint arXiv:2602.01523*, 2026. URL https://arxiv.org/abs/2602.01523.

Boxin Wang, Chankyu Lee, Nayeon Lee, Sheng-Chieh Lin, Wenliang Dai, Yang Chen, Yangyi Chen, Zhuolin Yang, Zihan Liu, Mohammad Shoeybi, et al. Nemotron-Cascade: Scaling Cascaded Reinforcement Learning for General-Purpose Reasoning Models. *arXiv preprint arXiv:2512.13607*, 2025.

Taiqiang Wu, Zenan Xu, Bo Zhou, and Ngai Wong. The Art of Efficient Reasoning: Data, Reward, and Optimization. *arXiv preprint arXiv:2602.20945*, 2026a. URL https://arxiv.org/abs/2602.20945.

Wei Wu, Liyi Chen, Congxi Xiao, Tianfu Wang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, Yao Hu, and Hui Xiong. Anti-Length Shift: Dynamic Outlier Truncation for Training Efficient Reasoning Models. *arXiv preprint arXiv:2601.03969*, 2026b. URL https://arxiv.org/abs/2601.03969.

Xingyu Wu, Yuchen Yan, Shangke Lyu, Linjuan Wu, Yiwen Qiu, Yongliang Shen, Weiming Lu, Jian Shao, Jun Xiao, and Yueting Zhuang. LAPO: Internalizing Reasoning Efficiency via Length-Adaptive Policy Optimization. *arXiv preprint arXiv:2507.15758*, 2025. URL https://arxiv.org/abs/2507.15758.

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. *arXiv preprint arXiv:2506.05256*, 2025. URL https://arxiv.org/abs/2506.05256.

Zhuolin Yang, Zihan Liu, Yang Chen, Wenliang Dai, Boxin Wang, Sheng-Chieh Lin, Chankyu Lee, Yangyi Chen, Dongfu Jiang, Jiafan He, Renjie Pi, Grace Lam, Nayeon Lee, Alexander Bukharin, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nemotron-Cascade 2: Post-Training LLMs with Cascade RL and Multi-Domain On-Policy Distillation. *arXiv preprint arXiv:2603.19220*, 2026. URL https://arxiv.org/abs/2603.19220.

Jingyang Yi, Jiazheng Wang, and Sida Li. ShorterBetter: Guiding Reasoning Models to Find Optimal Inference Length for Efficient Reasoning. *arXiv preprint arXiv:2504.21370*, 2025. URL https://arxiv.org/abs/2504.21370.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan 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, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. *arXiv preprint arXiv:2503.14476*, 2025. URL https://arxiv.org/abs/2503.14476.

Danlong Yuan, Tian Xie, Shaohan Huang, Zhuocheng Gong, Huishuai Zhang, Chong Luo, Furu Wei, and Dongyan Zhao. Shorten After You're Right: Lazy Length Penalties for Reasoning RL. *arXiv preprint arXiv:2505.12284*, 2025. URL https://arxiv.org/abs/2505.12284.

Qianfan Zhang, Tianyu Guo, Xuandi Ren, Jiale Chen, Ming Ding, Ran Xin, and Xia Xiao. Scaling Reasoning Tokens via RL and Parallel Thinking: Evidence From Competitive Programming. *arXiv preprint arXiv:2604.01302*, 2026. URL https://arxiv.org/abs/2604.01302.

Zihan Zheng, Zerui Cheng, Zeyu Shen, Shang Zhou, Kaiyuan Liu, Hansen He, Dongruixuan Li, Stanley Wei, Hangyi Hao, Jianzhu Yao, Peiyao Sheng, Zixuan Wang, Wenhao Chai, Aleksandra Korolova, Peter Henderson, Sanjeev Arora, Pramod Viswanath, Jingbo Shang, and Saining Xie. LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming? *arXiv preprint arXiv:2506.11928*, 2025. URL https://arxiv.org/abs/2506.11928.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The Surprising Effectiveness of Negative Reinforcement in LLM Reasoning. *arXiv preprint arXiv:2506.01347*, 2025. URL https://arxiv.org/abs/2506.01347.

A Experimental Details

A.1 Training Details

Optimization. Each update uses 128 prompts with $G = 16$ responses each, sampled at temperature 1.0 and top-p 1.0. The prompts follow a fixed order in which every prompt appears once before any repeats, so 36 updates cover all 3,532 prompts about 1.3 times. We use Adam with $(\beta_1, \beta_2) = (0.9, 0.98)$, $\epsilon = 10^{-8}$, no weight decay, and gradient-norm clipping at 1.0; the learning rate warms up over three steps from 3×10^{-7} to 4×10^{-6} and then stays constant. The ratio clip in Equation 2 uses $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$, without a KL penalty or entropy bonus.

A.2 Length Penalties

Linear length penalty. Related to Kimi k1.5 (Kimi Team, 2025), this penalty adds $\lambda\,[0.5 - (L_i - L_{\min})/(L_{\max} - L_{\min})]$ to the reward of each complete, correct response i in groups whose complete responses are all correct; $L_{\min}$ and $L_{\max}$ are the lengths of the shortest and longest complete responses in the group. We tune λ over $\{0.2, 0.4\}$ and report $\lambda = 0.2$.

Adaptive length penalty. This penalty combines solve-rate weighting as in ALP and MAI-Thinking-1 (Xiang et al., 2025; Microsoft AI, 2026), a concave length cost as in Composer 2 (Cursor Research Team, 2026), and the shortest correct response in the group as the length reference, as in ShorterBetter and Short-RL (Yi et al., 2025; Yuan et al., 2025). For the non-truncated correct responses $\mathcal{C}$ of a group, with shortest length $L_{\min}$, it subtracts a length penalty o_i from the reward of each response $i \in \mathcal{C}$:

$$o_i = \min\{1, \lambda\, p\, t_i\}, \qquad t_i = \max\{0, \log L_i - \log L_{\min}\},$$

where p is the group's solve rate and λ is a coefficient; we tune λ over $\{0.8, 0.4, 0.2\}$ and select $\lambda = 0.8$.

A.3 Evaluation

All evaluations, including AIME, sample at temperature 1.0 and top-p 0.95 with a 262,144-token (256k) context. Throughout, 102k denotes 102,400 tokens and 256k denotes 262,144.

B Supporting Length Analyses

B.1 Length of Correct and Incorrect Responses

Table 3 compares correct and incorrect responses to the same training prompt in the baseline's rollouts at steps 4–8. For each prompt with both outcomes, we compute the difference between its mean correct and mean incorrect response length, then average these differences equally across prompts. We repeat the calculation after removing truncated responses. Negative values mean that correct responses are shorter.

Rollout step	All rollouts	Completed only
4	-5.5 $[-8.1, -3.1]$	-2.8 $[-4.9, -0.7]$
5	-5.5 $[-8.1, -3.0]$	-3.1 $[-5.4, -0.9]$
6	-6.5 $[-9.3, -3.8]$	-3.9 $[-6.4, -1.5]$
7	-5.3 $[-8.1, -2.4]$	-2.7 $[-5.3, -0.2]$
8	-7.8 $[-10.2, -5.4]$	-4.5 $[-6.6, -2.4]$
Pooled	-6.1 $[-7.3, -5.0]$	-3.4 $[-4.4, -2.4]$

Table 3: **Within-prompt length difference between correct and incorrect responses.** Correct minus incorrect, in thousands of tokens, for the baseline's training rollouts at steps 4–8, with 95% confidence intervals.

Correct responses are shorter at every step. Pooled across steps, they are 6.1k tokens shorter when all rollouts are retained (95% CI: $[5.0, 7.3]$k shorter). Removing truncated responses reduces but

does not reverse the difference: correct completed responses remain 3.4k tokens shorter (95% CI: $[2.4, 4.4]$k shorter). Thus early training shows no within-prompt length advantage for successful responses.

B.2 Structure of Longer Traces

Each LCB response consists of a reasoning trace followed by a final answer that presents the program. We split the reasoning trace into paragraphs at blank lines and mark a paragraph if it contains a checking or testing cue: a case-insensitive whole-word match to a fixed list of words and phrases such as *verify*, *check*, *test*, *simulate*, *double-check*, *dry run*, *edge case*, and *sample input*. A marked *episode* is a maximal run of consecutive marked paragraphs; a single marked paragraph is thus an episode, and two runs separated by an unmarked paragraph count as two. Table 4 compares SFT and baseline responses: after RL, traces contain about twice as many paragraphs and episodes, both of similar size; episodes become modestly denser, and the final answer shortens. Because the marking is lexical, an episode shows where checking or testing language appears, not that a check was performed.

Measurement	SFT	Baseline	Change (%)
Reasoning tokens (k)	21.7	41.5	+91.6
Final-answer tokens (k)	1.93	1.48	−23.2
Paragraphs per response	341.7	683.9	+100.2
Tokens per paragraph	63.4	60.7	−4.3
Marked episodes per response	30.7	67.5	+119.6
Marked episodes per 10k reasoning tokens	14.2	16.3	+14.6
Tokens per marked episode	127.2	123.9	−2.6

Table 4: **Structural measurements of LCB responses** from the SFT model and the baseline. Lengths and counts are means per response; per-paragraph, per-token, and per-episode quantities are ratios of pooled totals. A marked episode is a maximal run of consecutive paragraphs that each contain a checking or testing cue.

B.3 Update-Signal Controls

The sign-restricted runs of §4.2 and the controls below share the baseline's initialization, data order, batch size, and optimizer, and train with a 256k window without overlong filtering; advantages are computed from the full group before either sign is dropped. Figure 6 adds two controls and the full-update run, which is the 256k fixed budget of §4.3. Over the first 12 steps, full and negative-only updates lengthen responses by 25.3% and 23.1%. Shuffling rewards within each group keeps the group's advantage values but breaks their link to correctness, and leaves length essentially unchanged (+0.2%). The second control, sign-flipped positives, gives each correct response the negative of its advantage and all others zero, placing negative updates on correct instead of failed responses; it lengthens responses by 9.9% over steps 1–19, versus 21.4% for negative-only updates.

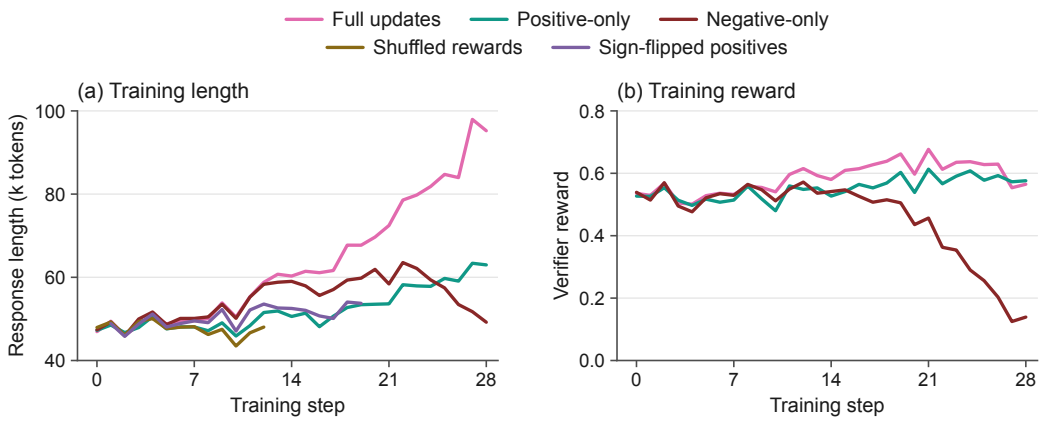

Figure 6: **Update-signal controls under a 256k window without overlong filtering.** (a) Mean training-response length and (b) verifier reward; panel (b) shows the runs with logged rewards. One run per variant; curves are unsmoothed.

C Adaptive-Budget Details and Additional Results

C.1 Reference Construction

We sample $K_{\text{ref}} = 16$ responses per training prompt from the SFT model at temperature 1.0 and top-p 1.0 and record each prompt's solve rate $\widehat{p}_0(x)$ and rounded median successful length $L_{\text{ref}}(x)$. With $\tau = 0.25$, a prompt needs at least five successes to receive an adaptive cap. Of all prompts, 80% receive caps; the others keep the full remaining context. With $\alpha = 1.75$, the caps have a mean of 74.4k tokens (median 72.8k, maximum 168k). Including the prompts that keep the full context, the mean budget per prompt is 112.1k tokens, above the baseline's 102k, even though 68% of prompts receive less than 102k: adaptive budgets reallocate the budget rather than shrink it.

Algorithm 1 RL with adaptive budgets

Require: Initial model π_0, prompts $\mathcal{D}$, reference count K_{ref}
Require: Multiplier α, solve-rate gate τ, context ceiling C_{high}, group size G
1: Sample and verify K_{ref} responses per prompt x.
2: Store each prompt's solve rate $\widehat{p}_0(x)$ and median successful length $L_{\text{ref}}(x)$; freeze them.
3: Initialize $\pi_\theta \leftarrow \pi_0$.
4: **for** each training batch **do**
5: Compute $B(x)$ with Equation 3; sample G responses within each budget.
6: Compute advantages (Equation 1).
7: Mask responses stopped by an adaptive cap ($m_i = 0$); update π_θ with Equation 2.
8: **end for**

C.2 Token Reduction by Initial Solvability

Figure 7 breaks down the token reduction and accuracy change of adaptive budgets relative to the baseline by initial solvability (§6.3).

C.3 Qwen3.5-4B

Table 5 reports the runs of §6.5. They start from Qwen3.5-4B (Qwen Team, 2026) and use the training data and settings of the corresponding main runs, except that adaptive budgets use $\alpha = 2.5$. As in Table 1, each run is represented by its checkpoint selected by LCB accuracy.

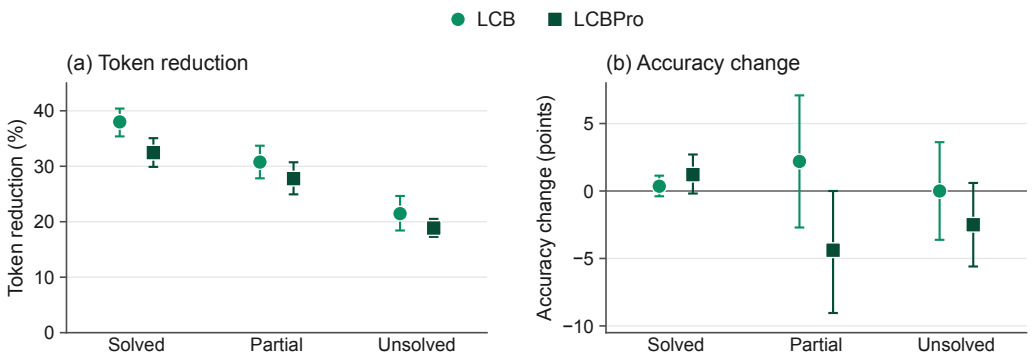

Figure 7: **Adaptive budgets shorten responses most on initially solved problems.** (a) Token reduction and (b) accuracy change of adaptive budgets relative to the baseline, by initial solvability (4/4, 1–3/4, and 0/4 SFT successes, as in Figure 2).

	LiveCodeBench		LCBPro medium	
Training intervention	Acc. (%)	Tokens (k)	Acc. (%)	Tokens (k)
Initial model (Qwen3.5-4B)	61.78	21.2	5.34	45.8
Fixed budget (102k) + overlong filtering (baseline)	<u>70.04</u>	102.8	**9.19**	186.1
Fixed budget (102k), no overlong filtering	63.05	<u>82.2</u>	5.98	<u>169.5</u>
Adaptive budget ($\alpha = 2.5$, ours)	**70.59**	**65.4**	<u>8.97</u>	**148.8**

Table 5: **Coding accuracy and response length on Qwen3.5-4B. Bold**/<u>underline</u>: best/second best among RL-trained methods.

C.4 Truncation During Training

Figure 8 tracks the share of training rollouts that reach their limit; under overlong filtering and adaptive budgets, these rollouts are masked from the loss. The baseline's share grows from 3% at step 0 to 24–30% at steps 32–36, so about a quarter of its late rollouts, each at the full 102k length, contribute no gradient. Without filtering, truncated responses count as failures, and truncation stays below 17% at 102k and 7% at 256k. Adaptive budgets cap about 80% of prompts and never reach the 256k ceiling; at step 0, their caps stop 4.9% of rollouts, the share used to calibrate α in §6.6. Their cap stops exceed the baseline's truncation until about step 28, peaking at 29% at step 21, and then fall to 12% by step 36 as responses shorten (Figure 1a).

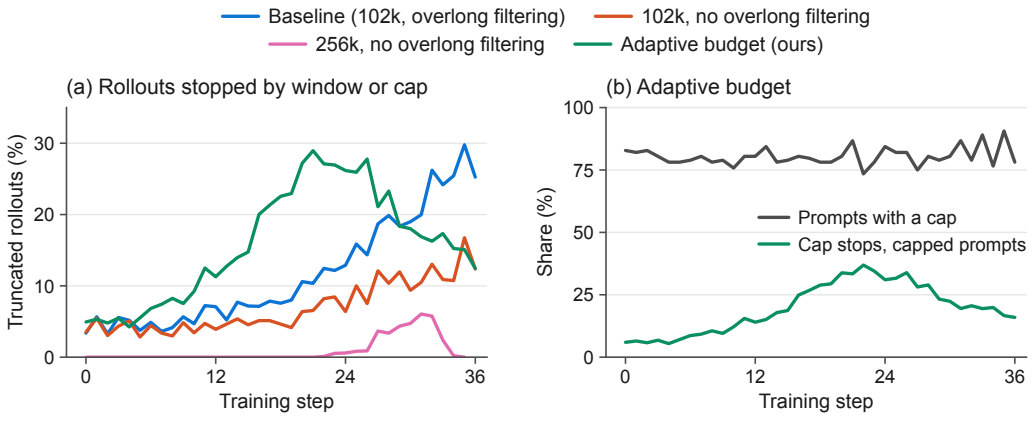

Figure 8: **Truncation during training.** (a) Share of training rollouts stopped by the context window (fixed budgets) or by the adaptive cap (adaptive budgets). (b) For adaptive budgets, the share of prompts that receive a cap and the share of their rollouts stopped by it. One run per configuration.