CATER: COMPETENCE-ADAPTIVE TRAINING FOR EFFICIENT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Efficient reasoning requires models to spend computation where it remains necessary. Yet long Chain-of-Thought (CoT) models can remain verbose on problems they already solve reliably, while uniform pressure to shorten responses can compromise reasoning on problems they have not yet mastered. Existing reinforcement learning approaches promote efficiency through outcome-based rewards, but sparse feedback provides limited guidance on how to eliminate unnecessary reasoning without sacrificing correctness. To resolve this problem, we propose CATER (Competence-Adaptive Training for Efficient Reasoning), a difficulty-adaptive framework that combines reinforcement learning with on-policy self-distillation, using the student's online success rate to assign a training signal to each problem. For consistently solved problems, concise distillation preserves existing capabilities with fewer reasoning tokens. For inconsistently solved problems, correctness-based reinforcement learning reinforces successful reasoning strategies. By adapting supervision to the student's evolving competence, our framework advances reasoning compression and capability development together. On Qwen3-4B, it reduces reasoning tokens by 39.1% on average across five benchmarks while maintaining comparable overall accuracy. These efficiency gains extend beyond mathematics to scientific reasoning and code generation.

1 INTRODUCTION

Extended CoT reasoning has become an effective way for large reasoning models to solve complex problems (Guo et al., 2025; Jaech et al., 2024). By spending more tokens on intermediate reasoning, these models can explore alternatives, check their progress, and recover from errors. This ability also creates a growing computational burden, as every additional token increases inference cost and delays the final answer. Moreover, extended reasoning is not equally useful for every problem (Suvra Ghosal et al., 2025; Fang et al., 2025). A model may repeatedly verify a familiar solution while still needing substantial exploration to solve a challenging question. Making reasoning broadly practical therefore requires learning to use computation according to actual need, producing concise solutions while retaining the reasoning necessary for correctness.

As shown in Figure 1, existing approaches encourage concise reasoning through prompting, supervised fine-tuning, and reinforcement learning. Prompting methods directly request shorter reasoning (Xu et al., 2025), while supervised approaches train on compressed demonstrations (Xia et al., 2025). Reinforcement-learning methods instead combine correctness with efficiency incentives, al-

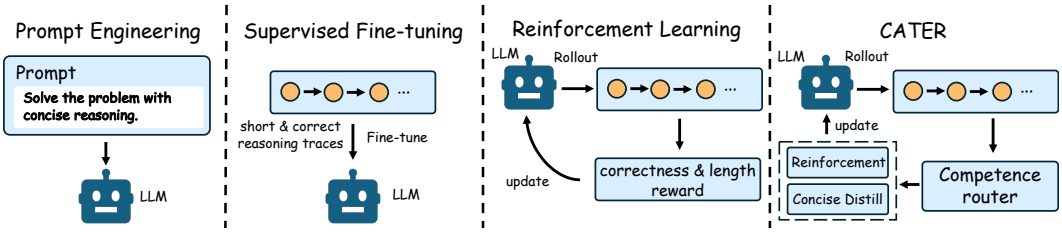


Figure 1: Comparison of different efficient reasoning approaches.

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.

lowing the model to discover shorter reasoning strategies through its own experience (Aggarwal & Welleck, 2025; Hou et al., 2025; Li et al., 2025). Such methods demonstrate that sequence-level objectives can effectively reduce reasoning length. However, the remaining challenge is determining whether a problem is actually ready for compression. For problems that the student solves inconsistently, sequence-level efficiency rewards do not indicate whether additional reasoning is redundant or still necessary for reliable success. Applying compression based on response-level efficiency signals can therefore conflate verbosity with useful computation. Conversely, once a problem is solved consistently, continuing to optimize correctness provides little information about which parts of an already successful reasoning process can be removed. The appropriate learning objective thus depends on the student's current competence.

Although on-policy distillation can provides dense token-level guidance on student-generated trajectories (Agarwal et al., 2024; Lu & Lab, 2025), it does not determine when compression is appropriate. Recent on-policy self-distillation further enables the model to obtain such guidance from itself by conditioning the teacher on privileged information or auxiliary instructions (Zhao et al., 2026). This makes distillation well suited to teaching how an existing reasoning process can be expressed more efficiently. However, dense compression guidance does not by itself determine when compression is appropriate. If the student still solves a problem inconsistently, encouraging concise behavior may optimize a reasoning process whose underlying capability has not yet become reliable. Conversely, once the student consistently succeeds, correctness-oriented optimization provides diminishing guidance on how to remove redundant computation. This suggests that reinforcement learning and on-policy distillation serve complementary roles rather than competing ones. Reinforcement learning is most useful for consolidating uncertain capability, while on-policy distillation is better suited to compressing capability that the student already demonstrates reliably.

To match the learning objective to the student's current competence, we introduce CATER, a difficulty-adaptive framework built around this allocation. The student samples multiple responses per problem, and verified outcomes determine the learning objective. All-correct groups receive concise on-policy self-distillation while mixed-outcome groups receive correctness-based group-relative policy gradients (Shao et al., 2024). A periodically refreshed self-teacher scores student prefixes using the problem and a brevity instruction. This separates the decision of where to apply compression from the strength of compression relative to correctness learning. As the student changes, newly sampled outcomes update the supervision assigned to each problem, allowing reasoning compression and capability development to proceed together.

In summary, our contributions are as follows:

- We propose a difficulty-adaptive framework that uses online success rates to coordinate compression with capability development, reducing redundant computation on mastered problems while prioritizing correctness on problems the model has yet to solve reliably.
- We combine concise on-policy self-distillation with group relative policy gradients, using gradient level balancing to control their relative strength and a periodically refreshed teacher to supply dense guidance.
- We demonstrate that competence-adaptive supervision yields a stronger accuracy–token trade-off across model scales and reasoning domains, preserving capability even under substantial compression.

2 Related Work

2.1 Efficient Reasoning

Recent work reduces reasoning cost through prompting, supervised compression, or reinforcement learning. Chain-of-Draft (Xu et al., 2025) encourages concise reasoning through prompting, while TokenSkip (Xia et al., 2025) learns from compressed traces. RL-based methods such as L1 (Aggarwal & Welleck, 2025), ThinkPrune (Hou et al., 2025), and DRPO (Li et al., 2025) optimize correctness together with length. More recent methods adapt efficiency objectives to problem difficulty, including Thinkless (Fang et al., 2025), CEEH (Luo et al., 2026b), LASER-D (Liu et al., 2025), and PACE (Feng et al., 2026), while online difficulty filtering uses current success rates to select informative problems for RL (Bae et al., 2026). CRISP (Sang et al., 2026) uses a conciseness-

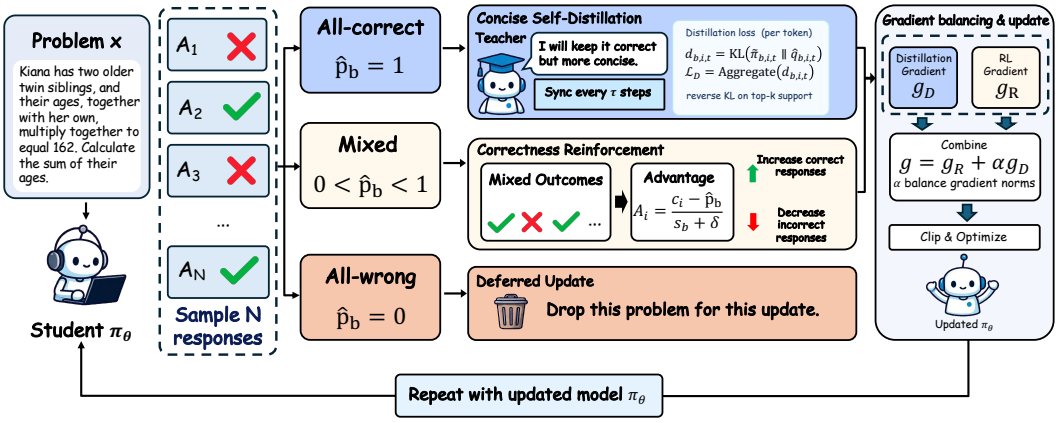


Figure 2: Overall framework of our method.

conditioned self-teacher with reverse-KL distillation, implicitly adapting compression across problems. This couples compression eligibility and strength within the same teacher signal. CATER instead uses verified on-policy outcomes to decide when compression is appropriate for the current student, leaving the teacher to guide how to compress.

2.2 On-Policy Distillation

On-policy distillation provides dense token-level supervision on student-generated trajectories (Agarwal et al., 2024; Lu & Lab, 2025), while OPSD enables self-distillation through privileged teacher context (Zhao et al., 2026). Recent work further studies the risks of privileged guidance (Yang et al., 2026; Kaur et al., 2026) and combines distillation with reward-based learning through token-level modulation, full-vocabulary objectives, uncertainty weighting, or trajectory-dependent supervision (Yang et al., 2026; Liu et al., 2026; Ke et al., 2026; Cai et al., 2026). In contrast, CATER gates dense distillation at the problem level, applying it only after repeated on-policy successes indicate that the underlying capability is sufficiently reliable.

3 Methodology

In this section, we first formulate the efficient reasoning problem, then introduce CATER. The overall framework of CATER is presented in Figure 2.

3.1 Problem Definition

We consider reasoning problems x drawn from a distribution $\mathcal{D}$. A student policy π_θ with trainable parameters θ generates a response z that contains intermediate reasoning and a final answer. Let $R(x, z) \in \{0, 1\}$ indicate whether the final answer is correct, and let $L(z)$ denote the number of reasoning tokens. We define:

$$\begin{aligned} p_\theta(x) &= \mathbb{E}_{z \sim \pi_\theta(\cdot|x)}[R(x, z)], \\ \ell_\theta(x) &= \mathbb{E}_{z \sim \pi_\theta(\cdot|x)}[L(z)] \end{aligned} \tag{1}$$

Where $p_\theta(x)$ is the student's success probability and $\ell_\theta(x)$ is its expected reasoning cost.

Given the initial parameters θ_0, efficient reasoning seeks to reduce the expected reasoning cost while preserving the initial model's accuracy. We express this goal as:

$$\begin{aligned} \min_\theta \quad & \mathbb{E}_{x \sim \mathcal{D}}[\ell_\theta(x)] \\ \text{subject to} \quad & \mathbb{E}_{x \sim \mathcal{D}}[p_\theta(x)] \geq \mathbb{E}_{x \sim \mathcal{D}}[p_{\theta_0}(x)] \end{aligned} \tag{2}$$

3.2 Online Competence Estimation

The student's current competence determines its current bottleneck. Unreliable problems require capability development, whereas reliably solved problems are ready for reasoning compression. At each update, the current student samples N responses $\{z_{b,i}\}_{i=1}^{N}$ for every problem x_b. A deterministic verifier assigns each response a binary reward $c_{b,i} = R(x_b, z_{b,i})$, from which we estimate competence as $\widehat{p}_b = N^{-1} \sum_{i=1}^{N} c_{b,i}$.

Recomputing this estimate from fresh rollouts allows the learning signal to evolve with the student. This allocation is motivated by the policy change required to improve correctness and efficiency. For a problem with true success probability p, the minimum KL change required to reach a target accuracy $a \geq p$ while limiting the expected length of correct responses to ℓ decomposes as:

$$C_x(a, \ell) = \mathrm{kl}(a \| p) + aI_x(\ell) \tag{3}$$

Where the two terms capture correctness improvement and compression within the distribution of correct responses, respectively. The decomposition suggests prioritizing correctness before reliable success is established and applying compression once the required capability is present.

Since p is unavailable during training, we route problems using $\widehat{p}_b$:

$$\mathcal{E} = \{(b,i) \mid \widehat{p}_b = 1\}, \qquad \mathcal{M} = \{(b,i) \mid 0 < \widehat{p}_b < 1\}, \qquad \mathcal{H} = \{(b,i) \mid \widehat{p}_b = 0\} \tag{4}$$

Responses in $\mathcal{E}$ receive concise self-distillation, while those in $\mathcal{M}$ receive correctness reinforcement from successful–unsuccessful contrasts. Problems in $\mathcal{H}$ are temporarily deferred. When all sampled responses fail, outcome feedback provides no reliable direction, while directly rewarding shorter responses may reinforce incorrect reasoning merely because it is concise. Since competence is re-estimated from fresh on-policy rollouts, these problems can later enter $\mathcal{M}$ once successful trajectories emerge, and eventually transition to $\mathcal{E}$ as competence becomes reliable.

We use outcome patterns to identify the learning signal supported by each rollout group. For a problem with current success probability p, the probability of entering concise distillation is p^N, concentrating compression on consistently solved problems as N increases. Mixed groups expose direct contrasts between successful and unsuccessful trajectories and therefore support correctness learning. Fresh on-policy rollouts continuously update these assignments, creating a curriculum in which capability development progressively opens new opportunities for compression.

3.3 Concise Self Distillation

For responses in $\mathcal{E}$, the student has already demonstrated reliable success, so the learning objective shifts from improving correctness to expressing the same capability more concisely. We use on-policy self distillation for this purpose. For each response sampled by the student, we run a forward pass through the teacher over the same response, with h included in the teacher context. Let q_ϕ denote the teacher policy. The two distributions are:

$$\begin{aligned} \pi_{b,i,t} &= \pi_\theta(\cdot \mid x_b, z_{b,i,<t}), \\ q_{b,i,t} &= q_\phi(\cdot \mid x_b, h, z_{b,i,<t}) \end{aligned} \tag{5}$$

Where t indexes positions relative to the response. Both distributions predict the same response token $z_{b,i,t}$, while $q_{b,i,t}$ reflects the additional preference for concise reasoning. Since the response prefixes are sampled from the current student, the supervision remains aligned with its behavior.

We match the two distributions over continuations that the student currently considers plausible. Let $S_{b,i,t}$ contain the k most probable tokens under $\pi_{b,i,t}$, and let $\widetilde{\pi}_{b,i,t}$ and $\widetilde{q}_{b,i,t}$ denote the two distributions restricted to this support and renormalized to sum to one. Their reverse Kullback–Leibler divergence is:

$$d_{b,i,t} = \sum_{v \in S_{b,i,t}} \widetilde{\pi}_{b,i,t}(v) \log \frac{\widetilde{\pi}_{b,i,t}(v)}{\widetilde{q}_{b,i,t}(v)} \tag{6}$$

This objective transfers the teacher's response to the conciseness instruction across the student's likely continuations. We then aggregate capped token losses over $\mathcal{E}$ as:

$$\mathcal{L}_D = \frac{1}{\max(1, D_D)} \sum_{(b,i) \in \mathcal{E}} \sum_{t=1}^{T_{b,i}} \min(d_{b,i,t}, \epsilon_D) \tag{7}$$

Where $\epsilon_D > 0$ is the token loss cap, $T_{b,i}$ is the full response length, and D_D is the globally aggregated number of response tokens in $\mathcal{E}$. Distillation covers both reasoning and final answer tokens, whereas $L(z)$ counts reasoning tokens alone. The teacher is initialized from the initial student and synchronized with the current student every τ updates. Freezing it within each interval stabilizes the distillation targets, while periodic synchronization keeps the conciseness guidance aligned with the evolving student throughout training.

3.4 Correctness Reinforcement

For responses in $\mathcal{M}$, the student's current bottleneck remains correctness. Mixed outcomes provide a group relative signal that increases the likelihood of successful responses and decreases that of unsuccessful ones. We optimize:

$$\mathcal{L}_R = -\frac{1}{\max(1, D_R)} \sum_{(b,i)\in\mathcal{M}} \frac{c_{b,i} - \widehat{p}_b}{s_b + \delta} \sum_{t=1}^{T_{b,i}} \log \pi_\theta(z_{b,i,t} \mid x_b, z_{b,i,<t}) \tag{8}$$

Where s_b is the population standard deviation of the rewards in group b, δ is a numerical stabilizer, and D_R is the globally aggregated number of response tokens in $\mathcal{M}$. As correctness improves and the sampled responses become consistently successful, online competence estimation naturally routes the problem to concise self distillation.

3.5 Coordinating the Learning Signals

Online competence determines what each problem should learn, while gradient coordination determines how strongly the two learning signals shape the shared policy update. This separation prevents the effective balance between capability development and reasoning compression from being determined by the number of routed responses or by differences in raw loss scale. Let the weighted branch gradients be:

$$g_D = \lambda_D \nabla_\theta \mathcal{L}_D, \qquad g_R = \lambda_R \nabla_\theta \mathcal{L}_R \tag{9}$$

Where λ_D and λ_R are fixed loss coefficients. When both branch norms are positive, we use the reinforcement gradient as the magnitude reference and combine the two signals according to:

$$g = g_R + \alpha g_D, \qquad \alpha = \frac{\rho}{1-\rho} \frac{\|g_R\|_2}{\|g_D\|_2} \tag{10}$$

Where g is the combined gradient, α is the distillation rescaling factor, and $\rho \in (0, 1)$ specifies the desired distillation share of the sum of the two branch gradient norms after rescaling. We treat α as a constant when constructing the update. The combined gradient is then clipped before the optimizer updates student model parameters.

This coordination makes the relative emphasis of the two objectives explicit and stable across batches. Together with online routing, it forms a closed training loop in which fresh outcomes identify the student's bottleneck, the corresponding learning signal addresses that bottleneck, and gradient balancing preserves the tradeoff between acquiring capability and compressing reasoning.

4 Experiments

4.1 Experimental Setting

4.1.1 Models and Datasets

We conduct our main experiments on Qwen3-1.7B and Qwen3-4B-Thinking-2507 (Yang et al., 2025) to enable fair comparisons with baselines evaluated on the same model. Training uses an easy subset of DeepScaleR provided by (Wu et al., 2026a). To reduce the influence of sampling variability on our overall conclusions, we evaluate across five benchmarks covering mathematical reasoning on AIME25 (Zhang & Math-AI, 2025), MATH-500 (Lightman et al., 2024), and GSM8K (Cobbe et al., 2021), scientific reasoning on GPQA-Diamond (Rein et al., 2023), and code generation on LiveCodeBench (Jain et al., 2025).

Table 1: Main results across model scales. We report accuracy (%) and average reasoning tokens.

	AIME25			MATH500		GSM8K		GPQA-D		LCB	
Method	**Avg@8** ↑	*Tok.* ↓	**Pass@8** ↑	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓
Qwen3-4B-Thinking-2507											
Base	75.00	20,201	83.33	95.40	7,003	95.15	1,526	68.18	8,534	75.26	12,644
Training-free methods											
CCoT	74.58	20,136	90.00	95.00	5,564	95.15	1,200	64.14	8,104	74.22	12,224
CoD	74.58	19,550	86.67	94.60	3,924	95.68	970	65.66	6,711	74.12	10,942
TALE-EP†	75.42	21,633	86.67	95.00	10,917	95.00	4,970	64.14	13,277	74.31	22,463
MI	65.00	10,234	90.00	95.60	2,764	93.25	1,146	66.67	9,475	66.35	10,649
ReBalance	74.58	20,847	86.67	94.80	6,790	95.30	1,477	67.68	8,395	74.50	12,478
Post-training methods											
CoT-Valve	71.25	20,461	83.33	88.60	5,289	77.86	180	66.67	9,282	73.55	12,212
O1-Pruner	70.42	16,720	83.33	95.40	3,695	94.54	1,133	64.65	6,226	74.60	11,029
Art	74.58	17,773	83.33	95.20	3,644	95.60	1,410	63.64	7,448	75.55	11,314
Kimi	73.75	14,551	86.67	95.40	4,019	95.15	1,044	64.65	6,008	73.08	9,594
CRISP	74.17	17,792	86.67	94.60	4,593	95.68	702	65.15	6,150	73.46	10,524
CATER	**76.67**	**14,467**	**86.67**	**95.20**	**3,671**	**95.45**	**767**	**65.15**	**5,076**	**74.03**	**8,964**
Qwen3-1.7B											
Base	34.58	16,976	60.00	90.20	5,811	90.52	2,156	40.91	10,178	53.55	11,645
Training-free methods											
CCoT	32.50	14,282	60.00	88.00	3,977	90.30	1,089	41.92	6,941	52.61	10,614
CoD	32.50	14,678	63.33	87.00	3,791	89.54	896	39.90	6,204	52.89	10,036
TALE-EP†	30.83	17,380	53.33	87.80	8,046	89.23	2,922	37.37	12,683	52.23	22,731
MI	17.50	10,795	36.67	81.00	3,018	86.73	655	24.24	10,229	19.43	7,807
ReBalance	35.00	17,175	53.33	87.60	5,327	89.61	2,036	42.93	9,882	52.23	11,353
Post-training methods											
CoT-Valve	11.67	8,387	33.33	45.00	2,266	53.60	227	28.79	1,849	23.32	5,248
O1-Pruner	21.25	11,109	30.00	85.60	3,603	88.32	1,401	34.34	5,262	51.85	8,456
Art	33.75	11,377	60.00	89.60	4,186	91.74	2,235	39.90	6,966	53.36	8,902
Kimi	31.25	13,519	46.67	88.00	3,963	90.22	1,632	41.41	8,309	51.56	10,613
CRISP	35.42	14,744	56.67	88.80	4,011	90.14	1,376	38.38	7,788	49.95	9,849
CATER	**35.42**	**14,124**	**56.67**	**90.00**	**3,619**	**90.75**	**1,495**	**39.39**	**7,724**	**53.84**	**10,199**

4.1.2 Implementation Details

Each training update uses 32 problems with six responses per problem and a maximum generation length of 10,000 tokens. We set the teacher synchronization interval to $\tau = 8$ and the target distillation gradient share to $\rho = 0.1$. During evaluation, the maximum generation length is 32,768 tokens. For AIME25, we sample eight responses per problem and report Avg@8 and Pass@8. For the remaining benchmarks, we report pass@1 accuracy. Reasoning efficiency is measured by the average number of reasoning tokens across all generated responses. Model-specific optimization configurations and further implementation details are provided in Appendix B.

4.1.3 Baselines

We compare against three categories of baselines: (i) *prompting methods*, including CCoT, CoD, and the budget-aware TALE-EP (Renze & Guven, 2024; Xu et al., 2025; Han et al., 2025); (ii) *training-free adaptation methods*, including MI for model interpolation and ReBalance for confidence-based activation steering (Wu et al., 2026b; Li et al., 2026); and (iii) *post-training methods*, including CoT-Valve, O1-Pruner, the reward-shaped RL method Art, and the self-distillation method CRISP (Ma et al., 2025; Luo et al., 2026a; Wu et al., 2026a; Sang et al., 2026), alongside a Kimi-style length-reward RL baseline (Team et al., 2025). All methods are evaluated under the same settings. Further implementation details are provided in Appendix C.

4.2 Main Results

As reported in Table 1, CATER achieves a strong balance between correctness and reasoning cost. On Qwen3-4B, it reduces reasoning tokens by 39.1% on average while maintaining comparable overall accuracy. The gains are most evident on AIME25 and GSM8K, where reasoning is shortened

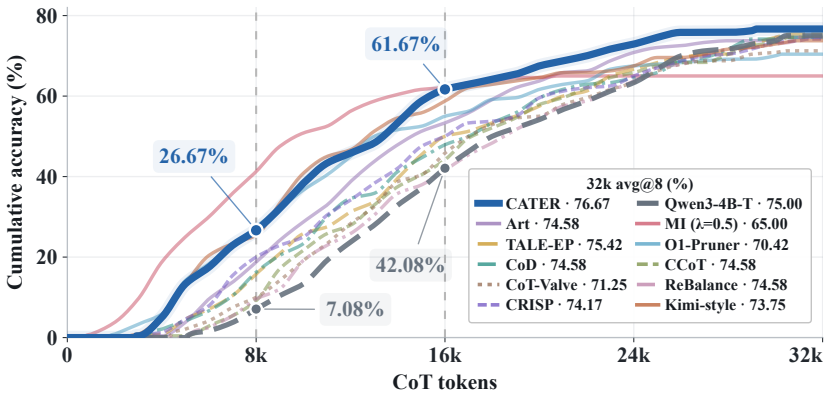


Figure 3: Cumulative accuracy as a function of CoT token budget on AIME25 with Qwen3-4B-Thinking-2507.

Table 2: Ablation results across five reasoning benchmarks.

Components					AIME25		MATH500		GSM8K		GPQA-D		LCB	
ROUTE	INST.	SYNC	BALANCE	RL	**Avg@8** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓
✓	✓	✓	✓	✓	**76.67**	**14,467**	**95.20**	**3,671**	**95.45**	**767**	**65.15**	**5,076**	**74.03**	**8,964**
×	✓	✓	✓	✓	72.92	14,077	95.40	3,609	94.92	718	64.65	5,151	72.80	9,710
✓	×	✓	✓	✓	74.17	18,593	95.20	5,068	95.68	1,051	63.64	6,838	74.41	11,099
✓	✓	×	✓	✓	74.58	16,760	95.40	4,689	94.69	887	68.18	5,880	73.08	10,063
✓	✓	✓	×	✓	72.50	12,711	95.00	3,477	94.77	730	63.13	4,339	73.55	8,063
×	×	×	×	✓	75.83	17,072	95.60	4,757	94.77	942	63.64	6,056	74.03	9,716
×	✓	✓	×	×	53.33	9,380	93.40	2,189	94.92	435	57.07	3,234	65.50	7,536

by 28.4% and 49.7%, respectively, while accuracy improves over the base model. By contrast, baselines that compress more aggressively often suffer substantial accuracy degradation, suggesting that uniformly shortening reasoning can remove computation still necessary for solving the problem.

Figure 3 provides a finer-grained view of this trade-off. CATER moves a substantially larger fraction of correct solutions into lower reasoning budgets, increasing cumulative accuracy from 7.08% to 26.67% at 8k tokens and from 42.08% to 61.67% at 16k tokens relative to the base model. Importantly, methods that concentrate solutions even more aggressively at small budgets can lose considerable full-budget accuracy, whereas CATER reaches 76.67% Avg@8 at 32k. This suggests that competence-aware compression primarily removes computation from problems that can already be solved reliably, rather than forcing uncertain problems into uniformly shorter trajectories.

The same pattern holds on Qwen3-1.7B, where CATER reduces reasoning tokens by 24.3% while preserving overall performance and improving results on AIME25, GSM8K, and LiveCodeBench. The consistency across model scales suggests that reasoning can be shortened without disrupting capability development when compression follows demonstrated competence, leaving uncertain problems sufficient room to improve through correctness-oriented learning.

4.3 ABLATION STUDY

Table 2 ablates the main components of CATER, where ROUTE, INST., SYNC, and BALANCE denote difficulty-aware routing, the conciseness instruction, periodic teacher synchronization, and dynamic gradient balancing, respectively. For W/O BALANCE, we calibrate a fixed OPSD coefficient from the main run's gradient logs to yield a 10% share of cumulative branch gradient norms along that reference trajectory. Removing routing reducing AIME25 Avg@8 from 76.67 to 72.92 and lowering accuracy on three other benchmarks without consistently shortening responses. This supports deferring compression until the student develops the corresponding capability. Removing the con-

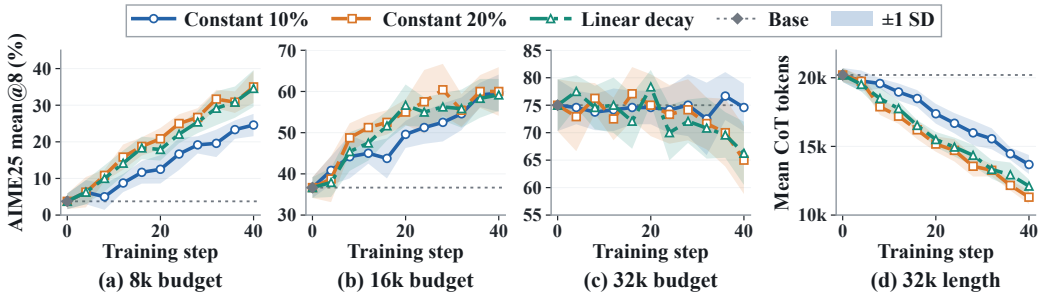


Figure 4: Effect of the distillation gradient share on AIME25 accuracy and reasoning length. Shaded regions denote ± 1 standard deviation.

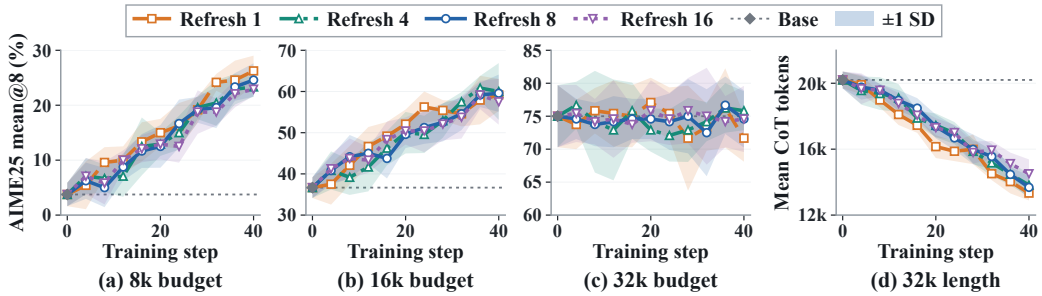


Figure 5: Effect of the teacher synchronization interval on AIME25 accuracy and reasoning length. Shaded regions denote ± 1 standard deviation.

ciseness instruction or freezing the teacher increases reasoning length across all five benchmarks, while replacing dynamic balancing with the fixed coefficient produces shorter responses at a consistent accuracy cost. The two single-objective variants reveal complementary failure modes. RL-only largely preserves correctness but remains consistently more verbose, whereas OPSD-only produces the shortest responses while reducing AIME25 Avg@8 to 53.33. Together, these results show that efficient reasoning requires coordinating capability improvement and compression according to the student's current competence.

4.4 Analysis

Our method relies on a simple principle that compression should follow competence rather than compete with it. We therefore examine whether the compression signal is sufficiently controlled, whether competence should be updated online, and how reliably it can be estimated during training.

4.4.1 Balancing Compression and Capability

As shown in Figure 4, stronger distillation accelerates reasoning compression and improves accuracy under the restrictive 8k budget, but eventually hurts performance when sufficient reasoning budget is available. In contrast, $\rho = 0.1$ reduces the average reasoning length from 20,201 to 14,467 tokens while improving 32k Avg@8 from 75.00 to 76.67. This suggests that compression is most effective when it removes redundancy without suppressing reasoning that is still useful for correctness.

Teacher synchronization introduces a related trade-off, see Figure 5. Synchronizing every step compresses fastest but yields lower final 32k accuracy, while infrequent synchronization leaves the teacher increasingly behind the evolving student. An interval of $\tau = 8$ provides a stable intermediate timescale, allowing the compression target to evolve with competence without changing too rapidly.

Table 3: Online competence routing versus a frozen compression set on Qwen3-4B-Thinking-2507.

	AIME25			MATH500		GSM8K		GPQA-D		LCB	
Routing	**Avg@8** ↑	*Tok.* ↓	**Pass@8** ↑	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓
Frozen	71.67	13,960	90.00	95.20	3,577	94.77	661	60.61	4,670	72.23	8,524
Online	**76.67**	**14,467**	**86.67**	**95.20**	**3,671**	**95.45**	**767**	**65.15**	**5,076**	**74.03**	**8,964**

Table 4: Effect of the number of training rollouts per problem on accuracy (%) and average reasoning tokens.

	AIME25			MATH500		GSM8K		GPQA-D		LCB	
Rollouts	**Avg@8** ↑	*Tok.* ↓	**Pass@8** ↑	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓	**Acc.** ↑	*Tok.* ↓
$N = 2$	72.92	14,668	86.67	95.40	3,836	95.15	763	64.65	5,160	73.46	9,239
$N = 4$	74.58	15,073	90.00	95.20	3,952	95.07	789	64.14	5,306	73.55	9,317
$\boldsymbol{N = 6}$	**76.67**	**14,467**	**86.67**	**95.20**	**3,671**	**95.45**	**767**	**65.15**	**5,076**	**74.03**	**8,964**
$N = 8$	75.00	14,081	86.67	95.40	3,551	95.22	701	64.14	4,834	72.80	8,688

4.4.2 Online Routing versus Frozen Compression

The comparison in Table 3 demonstrates the importance of adapting compression eligibility to the student's evolving competence. Although a frozen compression set yields consistently shorter responses, it also reduces accuracy across benchmarks, suggesting that some problems are compressed before the model can solve them reliably. Online routing avoids this mismatch and improves AIME25 Avg@8 by 5.00 points and GPQA-D accuracy by 4.54 points, with consistent gains on the remaining benchmarks. These results show that compression is most effective when it is introduced in step with capability development.

4.4.3 Competence Estimation and Evolution

In Table 4, we examine how sampled rollouts affect competence estimation. A problem with true success probability p enters the easy regime with probability p^N, so small groups can mistake unstable success for mastery. Increasing N from 2 to 6 improves accuracy on AIME25, GSM8K, GPQA-D, and LCB, while using more rollouts provides no further benefit. We therefore use $N = 6$ to balance reliable routing with sufficient compression supervision.

The routing dynamics reveal how capability growth changes the role of each problem in training. As shown in Figure 6, the fraction of all-correct groups rises while mixed and all-wrong groups become less frequent. A problem with no successful rollout cannot yet provide a useful correctness contrast, so its training is deferred. As the student improves, a newly successful trajectory can make that problem suitable for correctness reinforcement. With further progress, consistent success can make it eligible for concise self-distillation. Deferral thus preserves the possibility of learning from problems the student cannot solve today. By continually reassessing competence, CATER allows capability development to expand the set of problems that can be trained and eventually compressed.

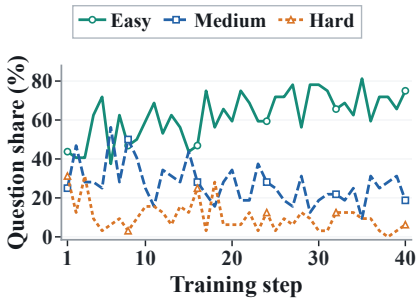


Figure 6: Routing dynamics.

5 Conclusion

We introduced competence-adaptive training for efficient reasoning. Online outcomes route inconsistently solved problems to correctness reinforcement and reliably solved problems to concise self-distillation. Across model scales and five benchmarks, this approach reduces reasoning tokens while preserving overall accuracy. The evolving routing distribution supports the central idea that improving competence creates new opportunities for compression.

AI USE STATEMENT

We used generative AI tools to aid and polish the writing of this paper, including revisions for clarity and concision. We also used these tools to search for and identify relevant literature. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 presents the CATER algorithm, while Sections 4.1.1 and 4.1.2 describe the datasets and evaluation protocol. Appendix A states the assumptions and provides derivations for the theoretical results. Appendices B and C give the training configurations and baseline implementation details. The supplementary materials include the training and evaluation code, together with example launch scripts.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The twelfth international conference on learning representations*, 2024.

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

Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, Jeongyeon Nam, and Donghyun Kwak. Online difficulty filtering for reasoning oriented reinforcement learning. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 700–719, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.30. URL https://aclanthology.org/2026.eacl-long.30/.

Qiye Cai, Yichuan Ma, Peiji Li, Yongkang Chen, Qipeng Guo, Yicheng Zou, Linyang Li, Xiaocheng Feng, and Bing Qin. H^2sd: Hybrid hindsight self-distillation, 2026. URL https://arxiv.org/abs/2607.18955.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Thinkless: Llm learns when to think. *arXiv preprint arXiv:2505.13379*, 2025.

Ruixiang Feng, Yuntao Wen, Silin Zhou, Ke Shi, Yifan Wang, Ran Le, Zhenwei An, Zongchao Chen, Chen Yang, Guangyue Peng, Yiming Jia, Dongsheng Wang, Tao Zhang, Lisi Chen, Yang Song, Shen Gao, and Shuo Shang. PACE: Prefix-protected and difficulty-aware compression for efficient reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30884–30903, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1545. URL https://aclanthology.org/2026.findings-acl.1545/.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, 2025.

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.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Naman Jain, 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*, volume 2025, pp. 58791–58831, 2025.

Simran Kaur, Narutatsu Ri, Yinghui He, Liam Fowl, and Sanjeev Arora. Rethinking on-policy self-distillation for thinking models, 2026. URL https://arxiv.org/abs/2607.05184.

Junlong Ke, Zichen Wen, Weijia Li, Conghui He, and Linfeng Zhang. Respecting self-uncertainty in on-policy self-distillation for efficient llm reasoning, 2026. URL https://arxiv.org/abs/2605.13255.

Gang Li, Yan Chen, Ming Lin, and Tianbao Yang. Drpo: Efficient reasoning via decoupled reward policy optimization. *arXiv preprint arXiv:2510.04474*, 2025.

Yulin Li, Tengyao Tu, Li Ding, Junjie Wang, Huiling Zhen, Yixin Chen, Yong Li, and Zhuotao Tian. Efficient reasoning with balanced thinking. *arXiv preprint arXiv:2603.12372*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

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, 2025. URL https://arxiv.org/abs/2505.15612.

Yifeng Liu, Shiyuan Zhang, Yifan Zhang, and Quanquan Gu. Self-distilled policy gradient, 2026. URL https://arxiv.org/abs/2606.04036.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Haotian Luo, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Xiaochun Cao, Dacheng Tao, Naiqiang Tan, and Li Shen. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14242–14257, 2026a.

Qin-Wen Luo, Sheng Ren, Xiang Chen, Rui Liu, Jun Fang, Naiqiang Tan, and Sheng-Jun Huang. Compress the easy, explore the hard: Difficulty-aware entropy regularization for efficient llm reasoning, 2026b. URL https://arxiv.org/abs/2602.22642.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. Cot-valve: Length-compressible chain-of-thought tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6025–6035, 2025.

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. *arXiv preprint arXiv:2311.12022*, 2023.

Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. *arXiv preprint arXiv:2401.05618*, 2024.

Hejian Sang, Yuanda Xu, Zhengze Zhou, Ran He, Zhipeng Wang, and Jiachen Sun. Crisp: Compressed reasoning via iterative self-policy distillation. *arXiv preprint arXiv:2603.05433*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. Does thinking more always help? understanding test-time scaling in reasoning models. *arXiv e-prints*, pp. arXiv–2506, 2025.

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.

Taiqiang Wu, Zenan Xu, Bo Zhou, and Ngai Wong. The art of efficient reasoning: Data, reward, and optimization. *arXiv preprint arXiv:2602.20945*, 2026a.

Taiqiang Wu, Runming Yang, Tao Liu, Jiahao Wang, and Ngai Wong. Revisiting model interpolation for efficient reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8624–8638, 2026b.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3351–3363, 2025.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*, 2025.

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 http://arxiv.org/abs/2505.09388v1.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 2026.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

A Further Details on Competence-Adaptive Training

A.1 Accuracy Stability Near Mastery

Competence routing prioritizes problems the student solves reliably. These problems provide successful trajectories in which redundant steps can be removed. Their accuracy is also locally more

stable near mastery under a controlled policy update, allowing concise supervision to target established capabilities.

We make the second statement precise under a finite response space and a differentiable, positive trajectory policy with fixed support. For a problem x, write $p_x(\theta) = \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}R(x,z)$, $s_\theta(z) = \nabla_\theta \log \pi_\theta(z \mid x)$, and $F_x = \mathbb{E}_{\pi_\theta}[s_\theta(z)s_\theta(z)^\top]$. For any parameter direction v, normalization gives $\mathbb{E}_{\pi_\theta} s_\theta = 0$. The score-function identity and Cauchy–Schwarz then yield

$$\begin{aligned}\left|\frac{\mathrm{d}}{\mathrm{d}\eta}p_x(\theta+\eta v)\Big|_{\eta=0}\right| &= \left|\mathbb{E}_{\pi_\theta}\left[(R(x,z)-p_x)v^\top s_\theta(z)\right]\right| \\ &\le \sqrt{p_x(1-p_x)\, v^\top F_x v}\end{aligned} \tag{11}$$

For updates of comparable Fisher-metric size, the bound on first-order accuracy change decreases as $p_x \to 1$. Small compression updates on mastered problems can therefore retain a more stable success probability. Near $p_x = 0$, the same factor decreases, but successful trajectories are scarce. This leads to concise guidance on all-correct groups, correctness contrasts on mixed groups, and deferral of all-wrong groups.

A.2 A KL Decomposition for Correctness and Compression

Equation 3 separates the policy change required for correctness from the change required to shorten correct reasoning. Fix a problem x and a finite set Ω_x of terminal responses under a fixed generation budget. Let $P = \pi_\theta(\cdot \mid x)$ be positive on Ω_x, let $\mathcal{C} = \{z : R(x,z) = 1\}$, and suppose $p = P(\mathcal{C}) \in (0,1)$. Write $P_+ = P(\cdot \mid \mathcal{C})$ and $P_- = P(\cdot \mid \mathcal{C}^c)$. We optimize over response distributions Q on Ω_x:

$$\begin{aligned}C_x(a,\ell) &= \inf_{\substack{Q:\, Q(\mathcal{C})\ge a \\ \mathbb{E}_Q[L(z)|\mathcal{C}]\le\ell}} \mathrm{KL}(Q\|P), \\ I_x(\ell) &= \inf_{\rho:\, \mathbb{E}_\rho L(z)\le\ell} \mathrm{KL}(\rho\|P_+)\end{aligned} \tag{12}$$

Where $a \in [p,1]$, ρ is supported on $\mathcal{C}$, and a and ℓ specify the desired accuracy and correct-response length.

For any Q with $b = Q(\mathcal{C}) \in (0,1)$, denote its correct and incorrect conditional distributions by Q_+ and Q_-. Splitting the KL sum over $\mathcal{C}$ and $\mathcal{C}^c$ yields the chain rule

$$\mathrm{KL}(Q\|P) = \mathrm{kl}(b\|p) + b\,\mathrm{KL}(Q_+\|P_+) + (1-b)\,\mathrm{KL}(Q_-\|P_-) \tag{13}$$

For a fixed feasible success mass b, the last term is minimized by $Q_- = P_-$. The optimal correct conditional distribution has KL cost $I_x(\ell)$. Consequently, the optimum at this fixed b is $f(b) = \mathrm{kl}(b\|p) + bI_x(\ell)$. Since

$$f'(b) = \log\frac{b(1-p)}{p(1-b)} + I_x(\ell) \ge 0 \qquad \text{for } b \ge a \ge p \tag{14}$$

The minimum over $b \in [a,1]$ occurs at $b = a$. Hence $C_x(a,\ell) = \mathrm{kl}(a\|p) + aI_x(\ell)$, including the endpoint $a = 1$ by continuity. Infeasible length targets have infinite cost, and restricting Q to a neural policy class gives a lower bound. At a common accuracy target a, the correctness cost $\mathrm{kl}(a\|p)$ decreases as competence p approaches a, while $I_x(\ell)$ quantifies the cost of shortening successful reasoning.

The compression cost has a useful explicit form. Let $Z_+(\beta) = \mathbb{E}_{z\sim P_+} e^{-\beta L(z)}$. For finite Ω_x,

$$I_x(\ell) = \sup_{\beta\ge 0}\{-\beta\ell - \log Z_+(\beta)\} \tag{15}$$

To see this, define $P_\beta^+(z) = P_+(z)e^{-\beta L(z)}/Z_+(\beta)$. Any feasible ρ satisfies

$$\mathrm{KL}(\rho\|P_+) = \mathrm{KL}(\rho\|P_\beta^+) - \beta\mathbb{E}_\rho L(z) - \log Z_+(\beta) \ge -\beta\ell - \log Z_+(\beta) \tag{16}$$

When ℓ lies strictly between the shortest correct-response length and $\mathbb{E}_{P_+}L$, a suitable $\beta > 0$ makes $\mathbb{E}_{P_\beta^+}L = \ell$ and attains equality. At or above the original mean, P_+ attains zero cost; at the shortest feasible length, the optimum conditions P_+ on the minimum-length correct responses. Moreover, $\frac{\mathrm{d}}{\mathrm{d}\beta}\mathbb{E}_{P_\beta^+}L = -\mathrm{Var}_{P_\beta^+}(L) \le 0$. Exponential tilting therefore moves probability toward shorter correct trajectories while retaining their correctness. Concise self-distillation supplies a token-level training signal for this objective.

A.3 Routing Probabilities with Finite Rollouts

For a fixed student policy and a question with true sampled-success probability $p \in (0, 1)$, let K be the number of correct responses among N independent rollouts. Then $K \sim \text{Binomial}(N, p)$ and the probabilities of the three CATER routes are

$$\Pr(K = N) = p^N, \qquad \Pr(0 < K < N) = 1 - p^N - (1-p)^N, \qquad \Pr(K = 0) = (1-p)^N \quad (17)$$

In particular, for a problem with $p \leq \tau < 1$, the probability of an all-correct group is at most τ^N. Increasing N makes accidental all-correct classification less likely for any fixed $p < 1$, at the cost of additional rollouts. In training, p changes with the student, while each route is computed from fresh samples. These statements motivate the rollout-count analysis in Section 4.4.3.

B Experiment Details

At each update, we sample $N = 6$ responses per question from the current student. Both models use the native chat template with thinking enabled. The box below shows the teacher's user message for a question that does not already request a boxed answer. The student receives the same message without the final conciseness instruction. If the question already specifies a boxed answer, we omit the added answer-format instruction. The teacher then scores each student-generated prefix conditioned on its user message.

Teacher prompt

User: *[Question]*

Please reason step by step, and put your final answer within `\boxed{}`. Keep your reasoning as brief as possible: avoid unnecessary repetition and irrelevant details, preserve the key reasoning steps and checks needed for correctness, and present the final answer in the format requested by the task.

At each update, we sample $N = 6$ responses for each of $P = 32$ questions using temperature 1.0, top-p 0.95, and top-k 20, with a limit of 10,000 generated tokens per response. At each distilled token, we select the top four tokens from detached student logits and normalize the student and teacher distributions over this shared support. The per-token objective is $\text{KL}(\widetilde{\pi}_\theta \| \widetilde{q}_\phi)$, capped at 0.05. The reinforcement branch uses full-vocabulary log probabilities of sampled tokens and group-relative correctness advantages normalized by the population standard deviation plus 10^{-4}. We set $\lambda_D = 1$ and $\lambda_R = 0.05$ and refresh the teacher every eight updates.

Across the three model configurations, we freeze the base weights and train LoRA adapters on `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. For the MoE model, the adapters also cover the corresponding expert projections. We use LoRA rank 64 and alpha 128, with zero dropout and no trainable bias. Optimization uses AdamW with a learning rate of 10^{-5}, $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, and zero weight decay; attention uses FlashAttention 2. Evaluation covers AIME25 (30 problems, eight responses each), MATH-500 (500 problems), GSM8K (1,319), GPQA-Diamond (198), and LiveCodeBench (1,055), with one response per problem on the latter four benchmarks. For AIME25, we report Avg@8 over 240 responses and Pass@8 over 30 problems. All evaluations use thinking-enabled decoding with temperature 0.6, top-p 0.95, top-k 50, and a generation limit of 32,768 tokens.

C Baselines

CCoT. Concise Chain-of-Thought (CCoT) (Renze & Guven, 2024) prompts the base model to reason briefly. We prepend "Think step-by-step through the problem to ensure you have the correct answer. Be concise." to each question.

CoD. Chain of Draft (CoD) (Xu et al., 2025) elicits minimal intermediate reasoning steps. We prepend "Think step by step, but only keep minimum draft for each thinking step, with 5 words at most." to each question. The five-word target is conveyed through the prompt.

TALE-EP. TALE-EP (Han et al., 2025) estimates a token budget for each question and incorporates it into the answer prompt. We obtain one estimate from the base model and use the parsed value as an advisory budget. Questions with invalid estimates use the standard prompt.

MI. Model interpolation (MI) (Wu et al., 2026b) combines model weights to control reasoning length. We interpolate with coefficient 0.5 between Qwen3-4B-Thinking-2507 and Qwen3-4B-Instruct-2507 for the 4B setting. For 1.7B, we combine the base model with a published Qwen3-1.7B-SFT model.

ReBalance. ReBalance (Li et al., 2026) uses confidence signals to adjust activation steering during generation. We fit a model-specific steering direction and controller on 500 separate mathematical training questions using hidden states, PCA, ridge regression, and discriminant analysis.

CoT-Valve. CoT-Valve (Ma et al., 2025) learns parameter adjustments that control CoT length. We adapt it using the 7,473 official GSM8K training questions, reserving 64 for development and fitting on the remaining 7,409. Short human-written solutions are formatted with the model's thinking and boxed-answer template. We train a rank-2 LoRA adapter with alpha 8 for five epochs (580 updates), global batch size 64, learning rate 10^{-5}, and maximum sequence length $4{,}096$. Evaluation uses the final adapter at strength 1.0.

O1-Pruner. O1-Pruner (Luo et al., 2026a) uses presampled trajectories to favor concise reasoning under correctness constraints. We sample 12 trajectories for each of 5,000 screened MATH training questions, with a generation limit of 8,192 tokens. Length-and-correctness weights determine two training trajectories per question, yielding 10,000 examples. Training uses a frozen model reference, a clipped sequence-probability ratio, one epoch (313 updates), global batch size 32, and learning rate 2×10^{-6}. We update the final 21 of 28 layers for 1.7B and the final 27 of 36 layers for 4B.

Art. Art (Wu et al., 2026a) combines training data selection, length-aware rewards, and reinforcement learning to improve reasoning efficiency. We evaluate its released checkpoints under the common evaluation protocol.

Kimi-style RL. Our Kimi-style RL baseline applies a group-relative length reward inspired by Kimi k1.5 (Team et al., 2025) within our group-relative policy-gradient framework. Both model scales use the 9,630-question mathematical pool, 32×6 responses per update, rank-64 LoRA, 40 updates, and learning rate 10^{-5}. Let $c_j \in \{0, 1\}$ denote correctness and L_j denote the whole-response length of response j. With $L_{\min}$ and $L_{\max}$ computed within its group, the length bonus is

$$b_j = \begin{cases} 0, & L_{\max} = L_{\min}, \\ \frac{1}{2} - \frac{L_j - L_{\min}}{L_{\max} - L_{\min}}, & L_{\max} > L_{\min},\ c_j = 1, \\ \min\left(0, \frac{1}{2} - \frac{L_j - L_{\min}}{L_{\max} - L_{\min}}\right), & L_{\max} > L_{\min},\ c_j = 0 \end{cases} \tag{18}$$

The reward is $r_j = c_j + \alpha_t b_j$, where $\alpha_t = 0$ for updates 1–8 and $\alpha_t = 0.2$ for updates 9–40. We normalize rewards within each group using the population standard deviation plus 10^{-4}, then apply the token-normalized, group-relative policy-gradient update with coefficient 0.05. Training includes all outcome groups. Within an all-wrong group, shorter responses can receive positive relative advantages.

CRISP. We reproduce the CRISP (Sang et al., 2026) in same settings. A method that compresses reasoning through on-policy self-distillation, using the same model conditioned on a conciseness instruction as the teacher.

D Additional Results

D.1 AIME 2026 Across Model Scales

Table 5 extends the comparison to AIME 2026. At 1.7B, CATER improves Avg@8 from 34.17 to 36.67 while reducing average reasoning length by 21.5%, with Pass@8 unchanged at 63.33. At 4B,

CATER raises Avg@8 from 73.33 to 77.92 and reduces average total response length by 32.8%. TALE-EP attains the highest 4B Avg@8 at 78.75, with an average response length of 18,234 tokens versus CATER's 13,399. The length columns use different counting conventions across scales, as indicated in the table.

Table 5: AIME 2026 accuracy and reasoning length.

	Qwen3-1.7B			Qwen3-4B-Thinking-2507		
Method	**Avg@8**	**Pass@8**	**Avg. Tokens**	**Avg@8**	**Pass@8**	**Avg. Tokens**
Base	34.17	63.33	17,263	73.33	90.00	19,929
CCoT	35.42	63.33	13,515	78.33	90.00	18,507
CoD	33.75	60.00	13,737	77.08	86.67	17,991
TALE-EP	35.42	63.33	15,036	78.75	90.00	18,234
MI	11.67	36.67	10,604	67.92	86.67	9,480
ReBalance	35.83	63.33	16,112	75.83	86.67	19,305
CoT-Valve	3.33	26.67	9,150	71.25	86.67	19,230
O1-Pruner	22.92	40.00	10,590	77.50	90.00	14,662
Art	38.33	63.33	10,669	77.08	90.00	15,608
Kimi	33.33	53.33	12,644	75.42	90.00	13,994
CATER	**36.67**	**63.33**	**13,544**	**77.92**	**90.00**	**13,399**

D.2 AIME 2025 at 30B Scale

Table 6 shows that CATER also improves efficiency at Qwen3-30B-A3B-Thinking-2507. Relative to the base model, it increases Avg@8 from 80.83 to 82.50 and Pass@8 from 86.67 to 93.33 while reducing average reasoning length by 12.6%.

D.3 Effect of LoRA Rank

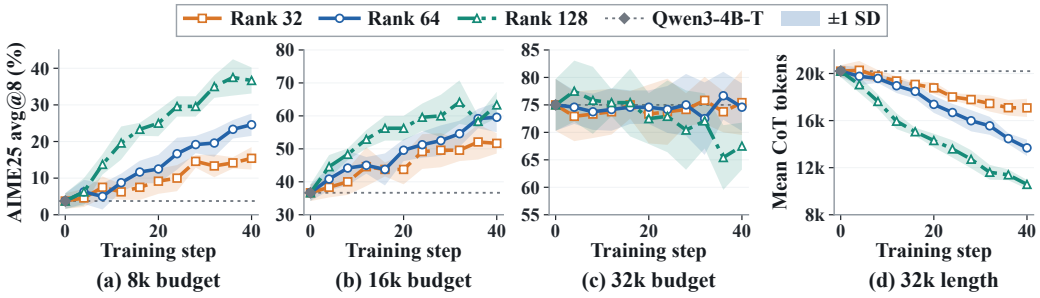


Figure 7: Effect of LoRA rank on AIME25 Avg@8 under 8k, 16k, and 32k generation budgets, and on mean reasoning length under the 32k budget. Shaded regions denote ± 1 standard deviation.

Figure 7 shows that LoRA rank changes the pace of reasoning compression. Rank 128 gains accuracy fastest under the 8k and 16k budgets and achieves the highest accuracy at both budgets by step 40. Under the 32k budget, its Avg@8 instead falls from 77.50 at step 4 to 67.50 at step 40 as mean reasoning length drops from 19,050 to 10,577 tokens. At step 40, ranks 32 and 64 retain higher 32k Avg@8 of 75.42 and 74.58, respectively. These trajectories show that stronger compression benefits tight inference budgets, while continued compression at rank 128 reduces accuracy when more reasoning tokens are available.

D.4 Training Compute

Table 7 reports the total training compute for each method. CATER uses 2.36 EFLOPs, within the range of the compared methods, and achieves the strong accuracy–reasoning-length trade-off reported in Table 1 at a comparable compute scale.

Table 6: AIME 2025 results at Qwen3-30B-A3B-Thinking-2507.

Method	**Avg@8** ↑	*Tok.* ↓	**Pass@8** ↑
Base	80.83	16,683	86.67
Art	81.25	14,271	93.33
CATER	**82.50**	**14,578**	**93.33**

Table 7: Approximate total training compute of the evaluated methods.

Method	**Training FLOPs (EFLOPs)**
RL-only	1.9558
O1-Pruner	2.8265
Kimi-style reward	2.0758
CATER	**2.3619**

D.5 AIME 2025 Compression Cases

Table 8 shows two AIME25 problems on which the CATER reduces mean CoT length. Both models follow the AIME25 evaluation settings in Appendix B.

Table 8: Selected AIME25 cases. CoT lengths are averaged over eight responses. Reductions are relative to the base model.

Problem	**Correct**	**Base CoT**	**CATER CoT**	**Reduction**
Radix divisibility	8/8 both	8,250	3,816	53.75%
Tangential trapezoid	8/8 both	10,287	5,141	50.02%

The radix problem reduces to $b + 7 \mid 56$: since $b > 9$, the valid bases are 21 and 49, whose sum is $\boxed{70}$. For the tangential trapezoid, the radius and area give a height of 6 and $r + s = 24$; tangency gives legs of length 12, so $(r-s)^2 = 4(12^2 - 6^2) = 432$ and $r^2 + s^2 = 504$. Across all 30 AIME25 problems, CATER reduces mean CoT length from 20,201 to 14,467 tokens, a 28.38% reduction. The complete Base and CATER responses to the radix problem are shown below.

D.5.1 Radix Divisibility: Full Responses

Question. Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b.

The two saved responses are typeset in full below. The "Reasoning" and "Final answer" labels mark the original `</think>` boundary; the end-of-message token is omitted from the display.

Base response (7,412 reasoning; 8,255 output tokens)

Reasoning

This is a complex or challenging question, and it is difficult to provide a direct and correct answer. I need to think about it. Well, so the problem says: find the sum of all integer bases b > 9 for which 17 base b divides 97 base b. Hmm, let's start by recalling how to convert numbers from base b to base 10 because that's usually helpful for divisibility problems like this.

First, let's write down what 17_b and 97_b are in base 10. For a number with digits d_n... d_1d_0 in base b, the value is d_n*b^n + ... + d_1*b + d_0. So 17_b is a two-digit number where the left digit is 1 (the b's place) and the right digit is 7 (the units place), right? Wait, hold on: in base b, the digits have to be less than b, right? So that's important later—each digit in base b must be less than b. So for 17_b to be a valid base b number, the digit 7 must be less than b, so b > 7, but the problem says b > 9, so that's already satisfied. Similarly, 97_b has digits 9 and 7, so both digits must be less than b, so since b > 9, 9 < b is given, so 7 < b is automatically true here too, good.

So converting 17_b to base 10: that's 1*b + 7. Let's call that N = b + 7.

Converting 97_b to base 10: that's 9*b + 7. Let's call that M = 9b + 7.

The problem states that 17_b divides 97_b, which means in base 10, N divides M, so (b + 7) divides (9b + 7). Okay, so now we have a divisibility condition in integers: b + 7 | 9b + 7, where b is an integer greater than 9 (and as we thought, b must be at least 10, but let's check if we need to worry about other constraints from digits, but since b > 9, digits 9 and 7 are both less than b, so that's fine—wait, actually, in base b, digits have to be 0 to b-1, so yes, 9 < b implies 9 is a valid digit, 7 < b is also true, so all good for b > 9).

So now, let's formalize the divisibility. If d divides m, then d divides any linear combination of m and d. So here, let d = b + 7, m = 9b + 7. Let's compute m - k*d for some k to eliminate the b term, which should give us a constant, and then d must divide that constant.

Let's see, m = 9b + 7, d = b + 7, so if we multiply d by 9, we get 9b + 63, which is 9 more than m? Wait, let's do it step by step. Let's solve for b in terms of d to see, but maybe better to do the Euclidean algorithm approach for divisibility.

In general, for integers a, b > 0, gcd(a,b) = gcd(b, a mod b). But here we want divisibility, so if d | m, then m mod d = 0. Let's compute (9b + 7) mod (b + 7). To compute this, we can write 9b + 7 = 9*(b + 7) - 9*7 + 7 = 9*(b + 7) - 63 + 7 = 9*(b + 7) - 56. Ah, that's a good trick! So that means 9b + 7 = 9*(b + 7) - 56. Therefore, when we divide 9b + 7 by b + 7, the remainder is -56 (or equivalently 56, but since we're talking about divisibility, the remainder must be 0, so b + 7 must divide 56).

Wait, let's check that algebra again to make sure. Let's let t = b + 7, so b = t - 7. Then substitute into 9b + 7: 9(t - 7) + 7 = 9t - 63 + 7 = 9t - 56. Yes! That's a better substitution. So 9b + 7 = 9t - 56 where t = b + 7. Therefore, t divides 9b + 7 if and only if t divides 9t - 56, which is true if and only if t divides 56 (since t divides 9t, so t divides (9t - (9t - 56)) = 56). Perfect, that's a solid way to see it.

So t = b + 7, and t must be a positive divisor of 56 (since b is an integer base greater than 9, so b 10, so t = b + 7 17, but wait, hold on: first, let's list all positive divisors of 56, then see which ones give b > 9.

First, factorize 56 to find all positive divisors. 56 = 8*7 = 2^3 * 7^1, so number of positive divisors is (3+1)(1+1)=8 divisors. Let's list them: 1, 2, 4, 7, 8, 14, 28, 56. Let's check: 1*56=56, 2*28=56, 4*14=56, 7*8=56, yes, that's all.

Now, t = b + 7, so b = t - 7. The problem says b > 9, so b 10 (since b is integer base, so integer greater than 9). Therefore, b = t - 7 > 9 t > 16. Wait, is that right? b > 9, so b is integer, so b 10, so t = b + 7 17, so t must be a divisor of 56 greater than 16.

Let's look at the list of positive divisors of 56: 1, 2, 4, 7, 8, 14, 28, 56. Which of these are greater than 16? Let's see: 14 is 14, which is less than 16; 28 is 28 > 16; 56 is 56 > 16. Wait, hold on, is that all? Wait, but hold on a second—when we did the substitution, we assumed that t = b + 7 is a positive divisor, but do we need to consider negative divisors? Wait, base b is an integer greater than 9, so b is at least 10, so t = b + 7 is at least 17, definitely positive, so we can ignore negative divisors here because b is positive integer greater than 9, so t is positive integer greater than 16.

But wait, hold on, let's test with t=28 and t=56 to make sure we didn't make a mistake in the algebra. Let's take t=28 first: b = 28 - 7 = 21. Let's check if 17_21 divides 97_21. Convert 17_21 to base 10: 1*21 + 7 = 28. Convert 97_21 to base 10: 9*21 + 7 = 189 + 7 = 196. Now, does 28 divide 196? 28*7=196, yes! Perfect, that works.

Base response (continued)

Now t=56: b=56-7=49. Check 17_49 = 49 + 7 = 56. 97_49 = 9*49 + 7 = 441 + 7 = 448. Does 56 divide 448? 56*8=448, yes, that works too.

Wait, but hold on, earlier I thought t >16, so t=28 and 56, but let's check if there's a divisor we missed? Wait, 56's divisors: we listed 1,2,4,7,8,14,28,56. Let's check t=14: b=14-7=7. But b>9 is required, so b=7 is too small, and also in base 7, digit 9 is invalid (digits must be less than base), so that's why b>9 is important—it's not just for the inequality, but also to have valid digits. Wait, the problem says "integer bases b>9", so b must be at least 10, so b=7 is out, b=8: t=15, but 15 isn't a divisor of 56, anyway. Wait, but let's check if maybe we made a mistake in the direction of the divisibility?

Wait, another way to think about divisibility: if (b + 7) divides (9b + 7), then (b + 7) divides (9b + 7) - 9*(b + 7) = 9b + 7 - 9b - 63 = -56. So yes, that's exactly what we had before, so (b + 7) divides 56, so b + 7 is a positive divisor of 56 (since b + 7 is positive integer, as b >9), so b + 7 {1,2,4,7,8,14,28,56}, so b {1-7= -6, 2-7=-5, 4-7=-3, 7-7=0, 8-7=1, 14-7=7, 28-7=21, 56-7=49}. Now, from these b's, we need b >9, so b=21 and b=49 are the only ones? Wait, hold on, b=7 is in the list, but b=7 is less than 9, so excluded. b=1 is less than 9, etc. So only 21 and 49?

Wait, but wait, hold on a second—when we have a base b, the digits have to be less than b, right? So in 97_b, the digits are 9 and 7, so we must have 9 < b (since the left digit is 9, which is a digit in base b, so must be less than b) and 7 < b (which is automatically true if 9 < b). So the problem states b >9, so that's exactly ensuring that 9 < b, so digits 9 and 7 are valid in base b (since 7 < 9 < b, so both less than b). So that condition is already satisfied by b >9, so we don't have to worry about extra digit constraints here because b >9 implies all digits in 97_b (which are 9 and 7) are valid.

But let's just make sure we didn't miss any divisors. Wait, is 56 the only number we need to consider? Wait, in the step where we said (b + 7) divides 56, is that all? Let's take an example with a smaller base to see if the logic holds, maybe not necessary, but just to verify.

Suppose b=10, base 10: 17_10=17, 97_10=97, 17 doesn't divide 97 (17*5=85, 17*6=102), so no. b=11: 17_11=18, 97_11=9*11+7=106, 106/185.888, not integer. b=12: 19, 9*12+7=127, 127/196.68, nope. b=13: 20, 9*13+7=124, 124/20=6.2, nope. b=14: 21, 9*14+7=133, 133/216.333, nope. b=15:22, 9*15+7=142, 142/226.45, nope. b=16:23, 9*16+7=151, 151/236.56, nope. b=17:24, 9*17+7=160, 160/246.666, nope. b=18:25, 9*18+7=169, 169/25=6.76, nope. b=19:26, 9*19+7=178, 178/266.846, nope. b=20:27, 9*20+7=187, 187/276.925, nope. b=21:28, 9*21+7=196, 196/28=7, yes, that's the one we found. Okay, so that works.

b=22:29, 9*22+7=205, 205/297.068, nope. b=23:30, 9*23+7=214, 214/307.133, nope. … b=48:55, 9*48+7=449, 449/558.16, nope. b=49:56, 9*49+7=448, 448/56=8, yes, that works.

Wait, but hold on, is there a case where maybe (b + 7) is a negative divisor? But base b has to be greater than 9, so b is positive integer greater than 9, so b + 7 is positive, so negative divisors don't give valid bases here, so we can ignore them.

Wait, but let's go back to the divisibility step to make sure we didn't flip anything. The problem says "17_b is a divisor of 97_b". In mathematics, when we say a divides b, we mean there exists an integer k such that b = k*a, with k positive integer (usually in number theory problems like this, unless specified otherwise, we consider positive divisors, and bases are positive integers greater than the digits, so all numbers here are positive). So yes, 97_b = k*17_b for some positive integer k, so our earlier reasoning holds.

Wait, another check: when we did the substitution t = b + 7, so 9b + 7 = 9(t - 7) + 7 = 9t - 56, so 9t - 56 must be divisible by t, so t | 56, that's correct. So t must be a positive divisor of 56, so b = t - 7 must be integer >9, so t >16, as b >9 t = b +7 >16.

Divisors of 56 greater than 16: 28, 56. Wait, is 56 the only one? 28 is 28, 56 is 56, next would be 112, but 112 isn't a divisor of 56. Wait, 56's divisors are only up to 56, so yes, only 28 and 56.

Wait, but hold on, wait a second—wait, in some contexts, when dealing with bases, do we consider the base to be at least the highest digit plus one? Which we already did: highest digit in 97_b is 9, so base must be at least 10, which is exactly b >9, so that's good.

But let me think again—could there be a mistake in the algebra? Let's take the division algorithm: if d | m, then m = q*d for some integer q 1 (since m and d are positive). So 9b + 7 = q*(b + 7), where q is a positive integer, and b >9 integer.

Let's solve for b here to see if we get the same results. So 9b + 7 = q b + 7q 9b - q b = 7q - 7 b(9 - q) = 7(q - 1) b = [7(q - 1)] / (9 - q). Since b must be a positive integer greater than 9, let's see what q can be.

Base response (continued)

First, denominator 9 - q must be positive (because numerator 7(q - 1): if q=1, numerator=0, b=0 invalid; q=2, numerator=7, denominator=7, b=1; q=3, numerator=14, denominator=6, b=14/6=7/3 not integer; q=4, numerator=21, denominator=5, 21/5=4.2; q=5, numerator=28, denominator=4, 28/4=7; q=6, numerator=35, denominator=3, 35/311.666; q=7, numerator=42, denominator=2, 42/2=21; q=8, numerator=49, denominator=1, 49/1=49; q=9, denominator=0, undefined; q=10, denominator negative, numerator positive (q>1), so b negative, which is invalid for base.

Ah! This is a good approach too, solving for b in terms of q, which might help catch if we missed something. Let's list q from 1 upwards:

q=1: b=0, invalid (base must be >9, and base 2 usually, but 0 is not a base)

q=2: b=7(2-1)/(9-2)=7/7=1, base 1 is invalid (base must be at least 2)

q=3: 7(2)/6=14/6=7/3, not integer

q=4:7(3)/5=21/5=4.2, nope

q=5:7(4)/4=28/4=7, base 7, but digits in 97_7: digit 9 is invalid (base 7 digits go up to 6), so even though algebraically b=7 gives 17_7=1*7+7=14, 97_7=9*7+7=70, and 14 divides 70 (70/14=5), but the problem specifies b>9, so we don't care about this, but good to note it's a solution to the divisibility but invalid base due to digits.

q=6:7(5)/3=35/311.666, not integer

q=7:7(6)/2=42/2=21, integer, b=21>9, valid base (digits 9 and 7 <21, good)

q=8:7(7)/1=49, integer, b=49>9, valid base (digits 9,7 <49, good)

q=9: undefined, division by zero

q=10:7(9)/(-1)=-63, negative base, invalid

q=0: 7(-1)/9=-7/9, negative, invalid

So this method also gives us b=21 and b=49 as the only solutions with b>9 (and valid bases, since digits are less than b for b=21,49). That matches what we had before.

Wait, so earlier when we listed divisors of 56, we had t=b+7=28,56 corresponding to b=21,49, which are exactly q=7,8 here. Let's see: when q=7, b=21, t=b+7=28; q=8, b=49, t=56. So that's consistent.

Is there any other q? We saw q=5 gives b=7, which is a solution but b=79, so excluded. q=2 gives b=1, invalid base, q=3,4,6 not integers, q=1,9,10+ invalid.

So seems like only two bases: 21 and 49. Wait, but the problem says "the sum of all integer bases b>9", so sum is 21+49=70?

Wait, hold on, wait a second—wait, in the first method, we said t divides 56, so t | 56, but in the second method, we have b=7(q-1)/(9-q), so let's see, when we had 9b +7 = q(b +7), so q=(9b +7)/(b +7)=9 - 56/(b +7), right? Because (9b +7)/(b +7)=9 - (63 -7)/(b +7)=9 - 56/(b +7). Yes! That's another way to see it: q must be an integer, so 56/(b +7) must be integer, so b +7 is a positive divisor of 56, which is exactly what we had. So q=9 - (56/t) where t=b+7, so q must be positive integer, so 56/t <9 t >56/96.222, so t7, which is why t=7 gives q=9 -8=1, b=0; t=8 gives q=9-7=2, b=1; t=14 gives q=9-4=5, b=7; t=28 gives q=9-2=7, b=21; t=56 gives q=9-1=8, b=49. So that's all the positive divisors t of 56, and corresponding q's, which are integers as we saw.

So all possible positive integer solutions for b (with b positive integer, base valid digits) are b=0 (invalid), b=1 (invalid), b=7 (digits invalid in base 7), b=21 (valid), b=49 (valid). So only b=21,49 satisfy b>9 and valid base (with valid digits, which is ensured by b>9).

Wait, but let's just confirm once more with b=21: 17 in base 21 is 1*21 +7=28, 97 in base 21 is 9*21 +7=189+7=196, 196 divided by 28 is 7, which is integer, so yes, 17_21 divides 97_21.

b=49: 17_49=49+7=56, 97_49=9*49+7=441+7=448, 448/56=8, integer, perfect.

Is there a base between 10 and 21 that we might have missed? Let's check b=10 to 20 manually to be thorough, since sometimes when doing algebra we might overlook something.

b=10: 17=17, 97=97, 97/175.705, nope

b=11:17=18, 97=9*11+7=106, 106/185.888, nope

b=12:19, 9*12+7=127, 127/196.684, nope

b=13:20, 9*13+7=124, 124/20=6.2, nope

Base response (continued)

b=14:21, 9*14+7=133, 133/216.333, nope (21*6=126, 133-126=7)

b=15:22, 9*15+7=142, 142/22=6.454…, 22*6=132, 142-132=10

b=16:23, 9*16+7=151, 151/236.565, 23*6=138, 151-138=13

b=17:24, 9*17+7=160, 160/246.666, 24*6=144, 160-144=16

b=18:25, 9*18+7=169, 169/25=6.76, 25*6=150, 169-150=19

b=19:26, 9*19+7=178, 178/266.846, 26*6=156, 178-156=22

b=20:27, 9*20+7=187, 187/276.925, 27*6=162, 187-162=25

b=21:28, 196/28=7, works

b=22 to 48: let's not check all, but from the equation b=7(q-1)/(9-q), for q=7,8 we have b=21,49, q=6 gives non-integer, q=5 gives b=7, so no solutions in between. For q=9, undefined, q=10 negative, so no.

Wait, another thought: when we say "17_b is a divisor of 97_b", does the problem consider 17_b as a number (i.e., positive integer) and 97_b as a number, which is standard in these problems, so yes, we're doing it right.

Is there a case where the quotient is zero? No, because 97_b is a larger number than 17_b for b>1 (since 9b+7 > b+7 when b>0), so quotient is at least 1, which we considered.

Wait, let's check if b=56-7=49 is the only other one. Wait, 56's divisors: we had 1,2,4,7,8,14,28,56. Let's take t=14: b=7, which we saw gives 17_7=14, 97_7=70, 70/14=5, so that's a solution but b=79, so excluded. t=8: b=1, 17_1? Wait, base 1 is not a standard base, digits have to be less than base, but base 1 is unary, but the problem says b>9, so irrelevant.

So yeah, seems like only two bases: 21 and 49. Sum is 21+49=70.

Wait, but hold on, wait a second—wait, in the problem statement: "integer bases b>9". Is there a possibility that b could be a non-integer? No, bases are always positive integers greater than the highest digit, so b must be integer >9, which we have.

Wait, let me check an online source or think if there's a standard problem like this. Wait, no, I can't, but let's think again about the divisibility condition.

Alternative approach: Let d = b + 7, so b = d - 7, d >16 (since b>9 d>16). Then 9b +7 = 9(d -7) +7 = 9d -63 +7 = 9d -56. So d divides 9d -56 d divides 56, as before. So d is a positive divisor of 56, d>16, so d=28,56, so b=21,49. That's concise.

I think that's solid. Sometimes when doing these problems, it's easy to forget that the base has to be greater than the digits, but the problem already specifies b>9, which for 97_b ensures digits 9 and 7 are valid (since 9 < b 7 < b automatically). So we don't have to check extra digit validity beyond b>9, which is given.

Wait, just to make 100% sure, let's see if b=21 is a valid base: in base 21, digits go from 0 to 20, so 9 and 7 are valid digits, yes. Base 49: digits 0-48, 9 and 7 are fine.

Is there a divisor of 56 we missed? Let's list all positive divisors of 56 again carefully:

Start from 1: 56 ÷1=56, so 1 and 56

56 ÷2=28, so 2 and 28

56 ÷4=14, so 4 and 14

56 ÷7=8, so 7 and 8

No others, since next would be 56÷8=7, which we already have. So yes, 8 divisors, listed correctly.

So divisors greater than 16: 28,56. Correct.

Therefore, b=28-7=21, 56-7=49. Sum is 21+49=70.

I think that's it. I don't see any mistakes in the reasoning now. Initially, I was worried maybe I missed a divisor, but listing them out and cross-checking with the q substitution method both lead to the same two bases. Manually checking a few bases around the solution also confirmed that there's no solutions in between.

Another way someone might go wrong is by miscalculating the conversion of the base numbers. For example, thinking 17_b is 1*7 + b instead of 1*b +7, but no—leftmost digit is higher place value, so in

Base response (continued)

two-digit number d1 d0 in base b, it's d1*b + d0, so 17_b is 1*b +7, which is what we did. If we had done it backwards, we would have gotten wrong numbers, but we didn't.

Let's see what happens if we did it backwards by mistake: suppose someone thought 17_b=7b +1, then 97_b=7b +9, then (7b +1) divides (7b +9), so 7b +1 divides 8, so divisors of 8:1,2,4,8, so b=(1-1)/7=0, (2-1)/7=1/7, (4-1)/7=3/7, (8-1)/7=1, so b=1, but that's wrong, so we know our conversion was correct because we got valid results with the right conversion.

So yes, conversion is correct: right digit is units, left digit is b's place.

I think we've verified this from multiple angles: algebraic manipulation with substitution, solving for b in terms of quotient, direct computation for nearby bases, checking divisor lists, ensuring digit validity is satisfied by the problem's b>9 condition.

So the only bases b>9 that work are 21 and 49, sum is 70.

Final Answer

$$\boxed{70}$$

Final answer

To solve the problem, we are asked to find the **sum of all integer bases** $b > 9$ such that the number 17_b (in base b) is a **divisor** of 97_b (also in base b).

Step 1: Convert Numbers to Base 10

We begin by converting both numbers from base b to base 10.

- $17_b = 1 \cdot b + 7 = b + 7$
- $97_b = 9 \cdot b + 7 = 9b + 7$

So the condition becomes:

$$b + 7 \mid 9b + 7$$

This means that $b + 7$ divides $9b + 7$. To simplify this divisibility condition, we can use the **Euclidean algorithm** or algebraic manipulation.

Step 2: Use Algebraic Manipulation

Let's define $t = b + 7$, so $b = t - 7$. Substituting into $9b + 7$:

$$9b + 7 = 9(t - 7) + 7 = 9t - 63 + 7 = 9t - 56$$

Thus,

$$t \mid 9t - 56 \quad \Rightarrow \quad t \mid 56$$

So t must be a **positive divisor of 56**, and since $b = t - 7 > 9$, it follows that:

$$t > 16$$

Step 3: List Divisors of 56 Greater Than 16

The positive divisors of 56 are:

$$\{1, 2, 4, 7, 8, 14, 28, 56\}$$

From these, the ones **greater than 16** are:

$$\{28, 56\}$$

Now compute corresponding values of b:

Base response (continued)

- $t = 28 \Rightarrow b = 28 - 7 = 21$
- $t = 56 \Rightarrow b = 56 - 7 = 49$

Both $b = 21$ and $b = 49$ satisfy $b > 9$, and they are valid bases because the digits in 97_b (i.e., 9 and 7) are less than b for both values.

Step 4: Verify the Divisibility Condition

Let's confirm that $17_b \mid 97_b$ for each value:

- For $b = 21$:
  - $17_{21} = 21 + 7 = 28$
  - $97_{21} = 9 \cdot 21 + 7 = 189 + 7 = 196$
  - $196 \div 28 = 7 \rightarrow$
- For $b = 49$:
  - $17_{49} = 49 + 7 = 56$
  - $97_{49} = 9 \cdot 49 + 7 = 441 + 7 = 448$
  - $448 \div 56 = 8 \rightarrow$

Both satisfy the required condition.

Final Step: Compute the Sum

The only valid bases $b > 9$ satisfying the condition are 21 and 49. Their sum is:

$$21 + 49 = \boxed{70}$$

Final Answer

$$\boxed{70}$$

CATER response (3,703 reasoning; 4,237 output tokens)

Reasoning

This is a complex or challenging question, and it is difficult to provide a direct and correct answer. I need to think about it. Well, so the problem says: find the sum of all integer bases b > 9 where 17 base b divides 97 base b. Hmm, let's start by recalling how to convert numbers from base b to base 10 because that might make the divisibility condition easier to handle with algebra.

First, let's write down what 17_b and 97_b are in base 10. For a number with digits d_n... d_1d_0 in base b, it's d_n*bˆn + ... + d_1*b + d_0. So 17_b is a two-digit number, so that's 1*b + 7, right? Wait, hold on: the leftmost digit is the higher place value, so yes, 17_b = 1*bˆ1 + 7*bˆ0 = b + 7. Similarly, 97_b = 9*bˆ1 + 7*bˆ0 = 9b + 7. Got that.

So the problem states that 17_b divides 97_b, which translates to (b + 7) divides (9b + 7) in integers, where b is an integer greater than 9 (since bases have to be greater than the digits they use; wait, hold on! Important point: in base b, all digits must be less than b. So 17_b has digits 1 and 7, so b must be

CATER response (continued)

greater than 7, but the problem says b > 9, so that's already covered, but just good to remember for later, maybe we can check later if we get b <=7, but problem says b>9 so we can focus on that).

So divisibility condition: (b + 7) | (9b + 7). When we have a divisibility condition like this, a common technique is to use the division algorithm or to express the larger number in terms of the smaller one to reduce the degree. Let's try that.

Let me let d = b + 7, so maybe that's a substitution to make it simpler? Wait, or just do polynomial division of 9b + 7 by b + 7. Let's do that.

Divide 9b + 7 by b + 7. How many times does b go into 9b? 9 times. Multiply (b + 7) by 9: 9b + 63. Subtract that from 9b + 7: (9b + 7) - (9b + 63) = -56. So the remainder is -56. Therefore, by the division algorithm, 9b + 7 = 9*(b + 7) - 56. Which means that (b + 7) divides (9b + 7) if and only if (b + 7) divides the remainder, which is -56 (and divisors can be positive or negative, but since b > 9, b + 7 is positive and greater than 16, so we can consider positive divisors).

Wait, let's recall the theorem: if a | b, then a | (b - k*a) for any integer k. So here, a = b + 7, b = 9b + 7 (wait, maybe bad notation with b as the variable, let's rename the remainder step to avoid confusion). Let's say for integers m, n, n > 0, m | n iff m | (n - q*m) where q is quotient. So here, n = 9b + 7, m = b + 7. Then n - 9m = (9b + 7) - 9(b + 7) = 9b + 7 - 9b - 63 = -56. Therefore, m | n iff m | (-56) iff m | 56 (since divisors come in positive/negative pairs, but m = b + 7 > 9 + 7 = 16, so m is positive integer greater than 16, so we can list positive divisors of 56 greater than 16, then set m = b + 7, solve for b, check if b > 9 and digits are valid? Wait, but we already have b > 9, and digits in 97_b: digits are 9 and 7, so base b must be greater than 9 (since digit 9 is present), which matches the problem's b > 9, so that's good, so we don't have to check digits beyond b > 9 here.

First, let's list all positive divisors of 56. 56 factors: 1, 2, 4, 7, 8, 14, 28, 56. Let's list them in order: 1,2,4,7,8,14,28,56. Now, m = b + 7 must divide 56 (from above, since m | 56), and m > 16 (since b > 9 m = b + 7 > 16). Let's see which divisors of 56 are greater than 16: 28 and 56. Wait, 14 is less than 16, next is 28, then 56. Let's check:

If m = 28, then b + 7 = 28 b = 21. Check if b > 9: yes, 21 > 9. Good.

If m = 56, then b + 7 = 56 b = 49. Also >9, good.

Wait, hold on, is that all? Wait, did we miss any divisors? Let's make sure. Divisors of 56: prime factorization is 2^3 * 7^1, so number of positive divisors is (3+1)(1+1)=8, which we listed: 1,2,4,7,8,14,28,56. Correct. So divisors greater than 16: 28,56. Wait, but hold on a second, let's test with an example to make sure we didn't make a mistake in the algebra.

Take b=21: 17_21 is 21 + 7 = 28; 97_21 is 9*21 + 7 = 189 + 7 = 196. Does 28 divide 196? 28*7=196, yes, perfect.

b=49: 17_49 = 49 + 7 = 56; 97_49 = 9*49 + 7 = 441 + 7 = 448. 56*8=448, yes, that works too.

Wait, but hold on, is there a case where m is a negative divisor? But m = b + 7, b is an integer base >9, so m is at least 17, positive, so no need to consider negative divisors here.

Wait, but wait a second, let's think again: when we said m | n iff m | (n - 9m), which is -56, so m | 56, that's correct because if m divides a number, it divides any multiple, so m divides 56 (since m divides -56 iff m divides 56). So that step is solid.

But let me try another approach to verify, maybe by setting up the fraction: (9b + 7)/(b + 7) must be an integer. Let's call that integer k, so (9b + 7) = k(b + 7). Then solve for b: 9b + 7 = kb + 7k 9b - kb = 7k - 7 b(9 - k) = 7(k - 1) b = 7(k - 1)/(9 - k). Since b must be a positive integer greater than 9, let's see what k can be.

First, denominator 9 - k must divide numerator 7(k - 1), and b > 0, so numerator and denominator must have same sign. Let's see: if k < 9, denominator positive; if k > 9, denominator negative. Let's check k > 9 first: denominator negative, so numerator must be negative for b positive. Numerator 7(k - 1) negative k - 1 < 0 k < 1, but k is integer (quotient of positive integers), so k < 1, but 9b + 7 > b + 7 for b > 0, so k 1, so k=1? Wait, k=1: b=7(0)/(8)=0, invalid. So k must be less than 9, so denominator positive, numerator positive (since k integer, k 1). Let's let k be integer with 1 k 8 (since k <9).

Compute b for k=1 to 8:

k=1: b=7(0)/8=0 invalid

k=2: 7(1)/7=1, b=1 <9 invalid

k=3: 7(2)/6=14/6=7/3 2.333 not integer

CATER response (continued)

k=4: 7(3)/5=21/5=4.2 nope

k=5: 7(4)/4=28/4=7, b=7 <9 invalid (also, base 7 can't have digit 9, but problem says b>9 anyway)

k=6: 7(5)/3=35/311.666 nope

k=7: 7(6)/2=42/2=21, b=21 >9, valid! That's the first one we had.

k=8: 7(7)/1=49, b=49 >9, valid! Second one.

k=9: denominator zero, undefined

k=10: as above, negative denominator, numerator positive (k=10, numerator=7*9=63), so b negative, invalid.

So this method also gives b=21 and b=49, same as before. Good, so that confirms it.

Wait, earlier when I listed divisors of 56 greater than 16, got 28 and 56, leading to b=21 and 49, which matches. So seems like those are the only two.

Wait, but hold on, let's make sure we didn't miss a divisor. Let's think: is there a case where maybe the remainder is positive? Wait, in the first method, remainder was -56, but divisibility is about absolute value, so yes, m divides 56. Let's check if b=16: m=23, 23 divides 9*16+7=151? 23*6=138, 23*7=161>151, no. b=17: m=24, 9*17+7=160, 160/246.666 no. b=20: m=27, 9*20+7=187, 187/276.925 no. b=21: works, as we saw. b=22: m=29, 9*22+7=205, 205/297.068 no. b=48: m=55, 9*48+7=439, 439/557.98 no. b=49: works. b=50: m=57, 9*50+7=457, 457/578.017, 57*8=456, so 457-456=1, remainder 1, no.

So seems like only 21 and 49 work. Wait, but let's check the problem statement again: "sum of all integer bases b>9". So sum is 21 + 49 = 70? Wait, is that right?

Wait, hold on, wait a second! Wait, when we did the substitution with m = b + 7 dividing 56, we said divisors greater than 16, but let's check m=14: m=14, b=7, but b=7 is less than 9, problem says b>9, so discard. m=8: b=1, too small. m=7: b=0, invalid. So yes, only m=28,56 give b>9.

Wait, but another check: let's compute (9b + 7)/(b + 7) for b=21: 196/28=7, integer. For b=49: 448/56=8, integer. Any others? Let's see, suppose someone thinks maybe negative bases? But no, bases are positive integers greater than the digits, so b must be at least 10 (since digits go up to 9 in 97_b), so b>9, positive integer, so we're safe.

Is there a possibility that when we did the remainder, we need to consider that in some contexts, but no, the algebra checks out with two different methods: polynomial remainder theorem (which is essentially what we did with the division algorithm) and setting the quotient as an integer and solving for b. Both lead to b=21 and 49.

Wait, let's make sure about the digit condition: in base b, digits must be less than b. For b=21, digits in 97_b are 9 and 7, both less than 21, good. For b=49, digits 9 and 7, both less than 49, perfect. So no issues there, which is why b>9 is sufficient (since digit 9 requires b>9, as base must be higher than any digit).

I think that's thorough. Sometimes when doing these problems, it's easy to forget the digit constraints, but here the problem specifies b>9, so we already have that covered, and the solutions satisfy b>9 with valid digits.

So the two bases are 21 and 49, sum is 21 + 49 = 70. Wait, 21+49 is 70, yes.

Wait, just to make 100% sure, let's see if there's a divisor we missed. Divisors of 56: 1,2,4,7,8,14,28,56. Let's compute b for each:

m=1: b=1-7=-6 invalid

m=2: b=2-7=-5 invalid

m=4: b=4-7=-3 invalid

m=7: b=7-7=0 invalid

m=8: b=8-7=1 invalid

m=14: b=14-7=7 invalid (b=7 <9, and also base 7 can't have digit 9)

m=28: b=21 valid

m=56: b=49 valid

CATER response (continued)

Yep, that's all. So only two valid bases, sum to 70.

I think that's solid. I don't see any mistakes in the reasoning now. Initially, I was a bit worried if I missed a divisor or messed up the remainder, but cross-verifying with the quotient method helped confirm. Sometimes when doing the division algorithm step, it's easy to miscalculate the remainder, but here (9b +7) - 9*(b +7) = -56, that's correct. So the key was recognizing that the divisor must divide 56, then finding which divisors give b>9.

Another way to think about it: (9b +7) mod (b +7) = (9*(b +7) - 63 +7) mod (b +7) = (-56) mod (b +7), so for the remainder to be zero, b +7 divides 56. Exactly, that's another way to see the remainder is -56, so modulo b+7 is -56, so must be 0, hence b+7 |56.

All paths lead to the same two bases. So sum is 70. I think that's the answer.

Final Answer

$$\boxed{70}$$

Final answer

To solve the problem, we are asked to find the **sum of all integer bases** $b > 9$ such that 17_b divides 97_b. Let's proceed step by step.

Step 1: Convert Numbers from Base b to Decimal

In base b:

- $17_b = 1 \cdot b + 7 = b + 7$
- $97_b = 9 \cdot b + 7 = 9b + 7$

We want $b + 7 \mid 9b + 7$, i.e., $b + 7$ divides $9b + 7$.

Step 2: Use Division Algorithm for Divisibility

Let's compute the remainder when $9b + 7$ is divided by $b + 7$:

$$9b + 7 = 9(b + 7) - 56$$

So the **remainder is** -56. Therefore, $b + 7 \mid 56$.

This means $b + 7$ must be a **positive divisor of 56**, and since $b > 9$, we require $b + 7 > 16$.

Step 3: List Positive Divisors of 56 Greater Than 16

The positive divisors of 56 are:

$$\{1, 2, 4, 7, 8, 14, 28, 56\}$$

From these, the divisors **greater than 16** are:

$$28, 56$$

Now compute corresponding values of b:

- If $b + 7 = 28$, then $b = 21$
- If $b + 7 = 56$, then $b = 49$

CATER response (continued)

Both $b = 21$ and $b = 49$ satisfy $b > 9$, and both bases are valid (since digits in 97_b are less than b).

Step 4: Final Check and Sum

We have two valid bases: $b = 21$ and $b = 49$.

Their sum is:

$$21 + 49 = \boxed{70}$$

Final Answer

$$\boxed{70}$$