RECURRENT SELF-IMPROVEMENT: DYNAMIC CROSS-LOOP ON-POLICY DISTILLATION FOR LOOPED LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Language Models (LoopLMs) offer a parameter efficient approach to scaling reasoning by reusing shared parameters across recurrent computation steps. Despite their promise, effective post-training of LoopLMs remains challenging. Existing approaches either provide reward based supervision that is sparse or costly to extend across loops, or rely on external teachers or privileged information, leading to limited teacher availability or teacher-student context mismatch. To address these limitations, we introduce **LoopOPD**, a cross-loop on-policy distillation framework that uses additional recurrent computation within a LoopLM as its own source of supervision. LoopOPD uses a frozen terminal loop policy as a compute privileged teacher for an intermediate loop student on student generated rollouts, providing dense supervision without an external teacher or privileged information. We further propose Dynamic LoopOPD (**D-LoopOPD**), which continually refreshes the terminal loop teacher as the shared model parameters are updated, enabling recurrent self-improvement. We characterize how distillation updates propagate across loop depths and derive sufficient conditions under which a single update yields simultaneous local improvement at both loop depths. Experiments on Ouro-Thinking models show that LoopOPD improves mathematical reasoning, while D-LoopOPD yields further gains through dynamic teacher updates. Despite being trained only on mathematical data, the resulting models also improve on general reasoning and code generation benchmarks, demonstrating that recurrent computation can serve as an effective source of supervision for LoopLMs. Our anonymous code repository is available at: `https://anonymous.4open.science/r/LoopOPD-8C7F`

1 INTRODUCTION

Looped Language Models (LoopLMs) have recently received growing attention as a promising approach to scaling reasoning through recurrent depth (Geiping et al., 2025). They reuse shared Transformer blocks across multiple latent computation steps before emitting each token (Zhu et al., 2025). This design increases computational depth without increasing parameter count, leading to stronger reasoning performance (Saunshi et al., 2025). However, effective post-training of LoopLMs remains challenging. Existing work applies or extends conventional post-training methods to LoopLMs, including reward based optimization and distillation, but faces limitations (Williams et al., 2026).

Specifically, prior work on Ouro, a family of pretrained LoopLMs (Zhu et al., 2025), reports limited gains from reinforcement learning with verifiable rewards (RLVR) (Guo et al., 2025) over the supervised fine-tuning (SFT) checkpoint. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) assigns reward credit only to the terminal loop, leaving intermediate latent computation without direct reward supervision. Reward Latent Thought Trajectories (RLTT) (Williams et al., 2026) alleviates this limitation by extending reward credit across all loops, but requires retaining per loop log-probabilities, causing memory usage to scale linearly with the number of loops and thereby limiting long rollout training. On-Policy Distillation (OPD) (Agarwal et al., 2024) provides dense distributional supervision but relies on an external, stronger teacher. For Ouro-2.6B-Thinking, no stronger same family LoopLM teacher is available, limiting OPD. On-Policy Self-Distillation (OPSD) (Zhao et al., 2026) instead constructs a same model teacher using privileged information, but introduces

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.

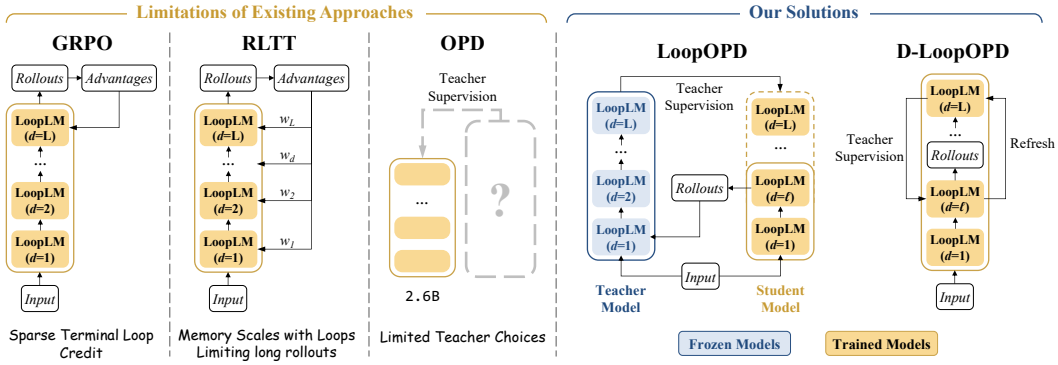

Figure 1: **Existing LoopLM post-training approaches vs. LoopOPD and D-LoopOPD.** Existing approaches variously face sparse supervision, increasing memory cost across loops, or dependence on stronger external teachers. LoopOPD uses additional recurrent computation as dense cross-loop supervision, while D-LoopOPD dynamically refreshes the terminal loop teacher during training.

teacher–student context mismatch (Kaur et al., 2026). Figure 1 (left) summarizes the three main limitations motivating our work: sparse terminal loop credit, increasing memory cost across loops, and limited availability of stronger teachers.

Effective post-training of LoopLMs therefore calls for dense supervision that directly reaches intermediate loops when no stronger external teacher is available. Because deeper loops receive more recurrent computation, the terminal loop naturally provides a stronger policy within the same model. This raises a question: *Can a LoopLM leverage its own additional recurrent computation to provide such supervision to its intermediate loops?*

To answer this question, we introduce LoopOPD. As illustrated in Figure 1 (right), LoopOPD uses an intermediate loop policy as the student and a frozen terminal loop policy as the compute privileged teacher. For student generated rollouts, both policies produce next token distributions on the same prefixes, and the terminal loop distribution is distilled into the intermediate loop through OPD. Unlike OPD and OPSD, the teacher's advantage arises not from additional parameters (Li et al., 2026) or privileged information (Zhao et al., 2026), but from additional recurrent computation. LoopOPD therefore turns the model's own recurrent computation into dense cross-loop supervision.

Intuitively, because different loops share the same parameters, training the student also updates the student model's terminal loop policy, creating a natural feedback path from the updated terminal policy back to the student. To exploit this feedback and further strengthen cross-loop supervision, we propose Dynamic LoopOPD (D-LoopOPD), which refreshes the terminal loop teacher at each optimization step, as shown in Figure 1 (right). The updated terminal policy then supervises the next student update, forming a closed teacher-student feedback loop for recurrent self-improvement. We further analyze cross-loop update transport and derive sufficient conditions for a single update to induce simultaneous local descent of task surrogates at both the intermediate and terminal loops. Table 1 compares the supervision signals and teacher requirements of these methods.

We evaluate LoopOPD and D-LoopOPD on Ouro-Thinking (Zhu et al., 2025) across mathematical reasoning (Zhang & Math-AI, 2026), general reasoning (Hendrycks et al., 2021), and code generation benchmarks (Liu et al., 2023). On mathematical reasoning, both methods outperform existing baselines on average, with D-LoopOPD further improving over LoopOPD. Despite training only on mathematical data, they also improve general reasoning and code generation over the vanilla models.

Our contributions are threefold:

- We introduce LoopOPD, which uses a frozen terminal loop policy as a compute privileged teacher for an intermediate loop student, providing dense on-policy supervision without an external teacher or privileged information.
- We propose D-LoopOPD, which refreshes the terminal loop teacher at each optimization step through shared parameter updates, forming recurrent self-improvement. We further analyze

Table 1: Comparison of supervision and teacher requirements across different methods.

Method	Supervision Signal	No Verifiable Reward	No External Teacher	No Privileged Information	Evolving Teacher
GRPO (Shao et al., 2024)	Sparse	×	✓	✓	–
RLTT (Williams et al., 2026)	Relatively Dense	×	✓	✓	–
OPD (Agarwal et al., 2024)	Dense	✓	×	✓	×
OPSD (Zhao et al., 2026)	Dense	✓	✓	×	×
LoopOPD	Dense	✓	✓	✓	×
D-LoopOPD	Dense	✓	✓	✓	✓

cross-loop update transport and derive sufficient conditions for local improvement of task surrogates at both the intermediate and terminal loops.

- Experiments on Ouro-Thinking models show improved mathematical reasoning at both model scales. On Ouro-2.6B-Thinking, D-LoopOPD raises terminal loop AIME 2024 Avg@16 from 62.08 to 71.67, with cross-loop evaluation showing improvements across all loop depths. The resulting models also improve general reasoning and code generation despite training only on mathematical data.

2 Related Work

Looped and Recurrent-Depth Language Models. Recurrent Transformer architectures such as the Universal Transformer (Dehghani et al., 2019) explore parameter sharing across computational depth. Recent work studies this principle in language models, showing that repeated latent computation can improve reasoning and provide an additional axis for scaling test-time compute (Saunshi et al., 2025; Geiping et al., 2025). Ouro (Zhu et al., 2025) extends this line of work to pretrained LoopLMs, which recurrently apply shared Transformer blocks before token generation. Our work builds on this structure by using additional loop computation as a source of supervision.

Intermediate Supervision for Recurrent Computation. Recent work develops training objectives for recurrent latent computation. LSRL (Ren, 2025) decodes intermediate recurrent states and scores them with an external grader to provide process supervision. RLTT (Williams et al., 2026) extends outcome reward credit from the terminal loop to intermediate loop distributions in LoopLMs. LoopRPT (Tang et al., 2026) uses an EMA teacher and noisy latent rollouts to construct stepwise reinforcement signals during pre-training, shaping intermediate representations. These methods highlight the value of directly supervising intermediate recurrent computation. Our work provides dense token level supervision to an intermediate loop using the model's own terminal loop distribution.

On-Policy Distillation and Self-Distillation. OPD trains a student on its own trajectories using dense teacher distributions, reducing the distribution mismatch from off-policy distillation (Agarwal et al., 2024). Recent analyses identify teacher-student compatibility and a teacher capability advantage as key factors for effective OPD (Li et al., 2026). OPSD constructs a teacher from the same model using privileged information (Zhao et al., 2026; Hübotter et al., 2026), but introduce a context mismatch that can distort supervision (Kaur et al., 2026). In contrast, LoopOPD retains the on-policy setting and teacher-student compatibility while deriving teacher advantage from additional recurrent computation, with teacher and student conditioned on the same context. D-LoopOPD further exploits shared parameters by refreshing the terminal loop teacher as the model is updated.

3 Preliminaries

Looped Language Models. Let x be a prompt and $y = (y_1, \ldots, y_m)$ a generated response of length m. A LoopLM recurrently applies shared weight Transformer blocks (Vaswani et al., 2017; Zhu et al., 2025) for L iterations before emitting each token. For the prediction of y_i, let $h_i^{(d)}$ denote the hidden state after loop d. Let θ denote the parameters of the LoopLM and g_θ its language model head. Applying g_θ to the loop d hidden state defines the corresponding next token distribution

$$P_\theta^{(d)}(\cdot \mid x, y_{<i}) = \mathrm{softmax}\Big(g_\theta(h_i^{(d)})\Big), \qquad d = 1, \ldots, L. \tag{1}$$

Standard autoregressive generation uses the terminal loop distribution, inducing the sequence policy

$$\pi_\theta(y \mid x) = \prod_{i=1}^{m} P_\theta^{(L)}(y_i \mid x, y_{<i}). \tag{2}$$

For $d < L$, we refer to $P_\theta^{(d)}$ as the intermediate loop distributions.

Group Relative Policy Optimization (GRPO). For each prompt x, GRPO samples a group of responses $\{y^{(j)}\}_{j=1}^{G}$ from the old policy $\pi_{\theta_{\text{old}}}$ and evaluates them with scalar rewards $r_1, \ldots, r_G$. The group relative advantage is $\widehat{A}_j = (r_j - \bar{r})/\sigma_r$. For a LoopLM, response generation uses the terminal loop distribution, so the token level importance ratio is

$$\rho_{j,i}(\theta) = \frac{P_\theta^{(L)}(y_i^{(j)} \mid x, y_{<i}^{(j)})}{P_{\theta_{\text{old}}}^{(L)}(y_i^{(j)} \mid x, y_{<i}^{(j)})}. \tag{3}$$

GRPO optimizes the clipped objective

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\mathbb{E}\left[\frac{1}{G}\sum_{j=1}^{G}\frac{1}{m_j}\sum_{i=1}^{m_j}\min\Big(\rho_{j,i}(\theta)\widehat{A}_j, \text{clip}(\rho_{j,i}(\theta), 1-\epsilon, 1+\epsilon)\widehat{A}_j\Big)\right] + \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{4}$$

where ϵ is the clipping parameter, β is the KL coefficient, and π_{ref} is the reference policy. Thus, GRPO uses the terminal loop policy for rollouts and direct reward credit, while gradients propagate through all loops to update the shared parameters.

Reward Latent Thought Trajectories (RLTT). RLTT extends reward credit from the terminal loop to all loops while retaining the group relative advantage (Williams et al., 2026). Its policy objective is

$$\mathcal{L}_{\text{RLTT}}(\theta) = -\mathbb{E}\left[\frac{1}{G}\sum_{j=1}^{G}\frac{1}{m_j}\sum_{i=1}^{m_j}\sum_{d=1}^{L}\omega_d \log P_\theta^{(d)}(y_i^{(j)} \mid x, y_{<i}^{(j)})\widehat{A}_j\right] + \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{5}$$

where $\omega_d \geq 0$ and $\sum_{d=1}^{L}\omega_d = 1$. A limitation of RLTT is that it retain per loop log-probabilities, leading to memory usage that scales linearly with the number of loops.

On-Policy Distillation (OPD). OPD uses token level teacher distributions rather than scalar outcome rewards (Agarwal et al., 2024). Let θ and ϕ denote the parameters of the student LoopLM and the frozen teacher LoopLM. For $y \sim \pi_\theta(\cdot \mid x)$, both models are evaluated on the same prefixes $c_i = (x, y_{<i})$. The reverse-KL distillation objective is

$$\mathcal{L}_{\text{OPD}}(\theta; \phi) = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} D_{\text{KL}}\Big(P_\theta^{(L)}(\cdot \mid c_i) \,\Big\|\, P_\phi^{(L)}(\cdot \mid c_i)\Big)\right]. \tag{6}$$

4 METHOD

Prior work reports limited gains from conventional reinforcement learning post-training on LoopLMs (Zhu et al., 2025; Williams et al., 2026). We explore an intrinsic source of supervision in LoopLMs: additional recurrent computation. Since a LoopLM admits a next token distribution at every loop, each loop naturally induces a corresponding autoregressive policy.

4.1 LOOPOPD: CROSS-LOOP ON-POLICY DISTILLATION

For any loop $d \leq L$, we define the loop d policy as

$$\pi_\theta^{(d)}(y \mid x) = \prod_{i=1}^{m} P_\theta^{(d)}(y_i \mid x, y_{<i}). \tag{7}$$

As illustrated in Figure 2 (left), LoopOPD selects an intermediate loop $\ell < L$ as the student and uses the terminal loop L policy as the compute privileged teacher. We use $\ell = 2$ and $L = 4$. Let

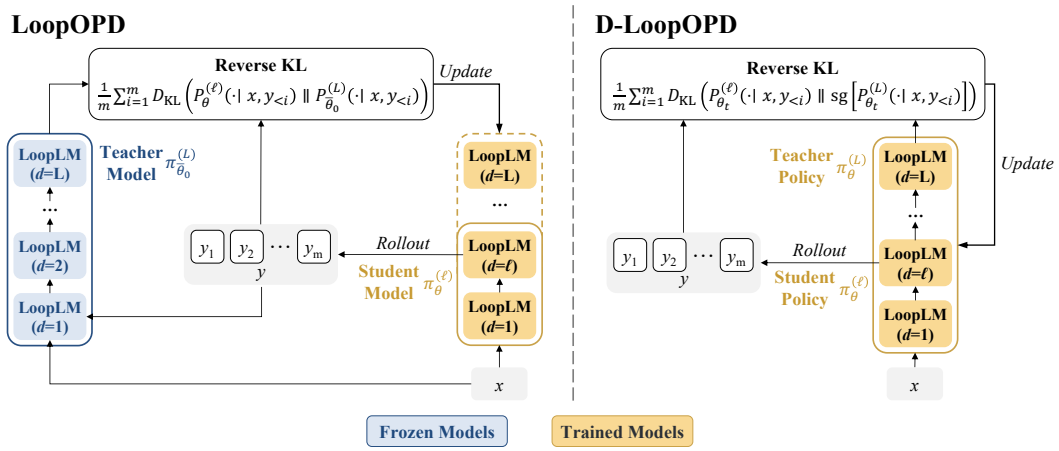

Figure 2: **Overview of LoopOPD and D-LoopOPD.** LoopOPD uses a frozen terminal loop policy as a compute-privileged teacher and distills its next token distribution into an intermediate loop student on student generated rollouts. D-LoopOPD instead uses the current terminal loop policy of the same trainable LoopLM as the teacher, so each student update also changes the teacher used at the next step through shared parameters, forming recurrent self-improvement.

θ_0 denote the initial LoopLM parameters. We initialize the trainable student with $\theta \leftarrow \theta_0$ and the frozen teacher with $\bar{\theta}_0 = \theta_0$. For each prompt x, a response is sampled from the student policy,

$$y \sim \pi_\theta^{(\ell)}(\cdot \mid x). \tag{8}$$

Both policies are evaluated on the same prefixes $c_i = (x, y_{<i})$, and LoopOPD distills the terminal loop distribution into the student using the reverse-KL objective in equation 6:

$$\mathcal{L}_{\text{LoopOPD}}(\theta; \bar{\theta}_0) = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} D_{\text{KL}}\left(P_\theta^{(\ell)}(\cdot \mid c_i) \,\middle\|\, P_{\bar{\theta}_0}^{(L)}(\cdot \mid c_i)\right)\right]. \tag{9}$$

While LoopOPD supervises only the loop ℓ student policy, the loop ℓ and terminal loop policies of the trainable LoopLM share the same parameters. Updating the student therefore also changes the current terminal loop policy. Let θ_t and θ_{t+1} denote the parameters before and after optimization step t. This interaction can be summarized as

$$\pi_{\bar{\theta}_0}^{(L)} \xrightarrow{\text{OPD}} \pi_{\theta_{t+1}}^{(\ell)} \xrightarrow{\text{parameter sharing}} \pi_{\theta_{t+1}}^{(L)}. \tag{10}$$

4.2 Dynamic LoopOPD: Recurrent Self-Improvement

As illustrated in Figure 2 (right), D-LoopOPD leverages this shared parameter coupling by using the current terminal loop policy as the teacher at each optimization step. Its objective is

$$\mathcal{L}_{\text{D-LoopOPD}}(\theta) = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} D_{\text{KL}}\left(P_\theta^{(\ell)}(\cdot \mid c_i) \,\middle\|\, \text{sg}\left[P_\theta^{(L)}(\cdot \mid c_i)\right]\right)\right], \tag{11}$$

where $\text{sg}[\cdot]$ denotes stop-gradient. Gradients flow only through the student distribution, while the shared parameter update modifies both the loop ℓ and terminal loop policies. This interaction can be summarized as

$$\pi_{\theta_t}^{(L)} \xrightarrow{\text{OPD}} \pi_{\theta_{t+1}}^{(\ell)} \xrightarrow{\text{parameter sharing}} \pi_{\theta_{t+1}}^{(L)}. \tag{12}$$

The updated terminal loop policy then serves as the teacher for the next optimization step. Repeating this process forms a closed feedback loop between student learning and teacher evolution, which we refer to as recurrent self-improvement.

4.3 Theoretical Analysis

We analyze one distillation update and its effect at both loop depths. At step t, sample a batch $\mathcal{B}_t = \{(x^{(j)}, y^{(j)})\}_{j=1}^{n_t}$ with $y^{(j)} \sim \pi_{\theta_t}^{(\ell)}(\cdot \mid x^{(j)})$, response lengths $m_j \geq 1$, and prefixes $c_{j,i} =$

$(x^{(j)}, y^{(j)}_{<i})$. We hold these sampled tokens and the targets $q_t(\cdot \mid c) = \mathrm{sg}[P^{(L)}_{\theta_t}(\cdot \mid c)]$ fixed during the update, as in OPD. The empirical loss retains the response-wise normalization of equation 11:

$$\mathcal{L}_t(\theta) = \frac{1}{n_t}\sum_{j=1}^{n_t}\frac{1}{m_j}\sum_{i=1}^{m_j} D_{\mathrm{KL}}\Big(P^{(\ell)}_{\theta}(\cdot \mid c_{j,i}) \,\Big\|\, q_t(\cdot \mid c_{j,i})\Big), \qquad g_t = \nabla_\theta \mathcal{L}_t(\theta_t), \qquad \theta_{t+1} = \theta_t - \eta_t g_t. \tag{13}$$

where g_t is the gradient of the empirical distillation loss at θ_t, and $\eta_t > 0$ is the step size. We analyze an exact gradient step for the full-distribution KL objective.

Proposition 1 (Descent with a step-frozen teacher)**.** *Suppose $\mathcal{L}_t$ has an M_t-Lipschitz gradient, $M_t > 0$, in a neighborhood of the update segment. If $g_t \neq 0$ and $0 < \eta_t < 2/M_t$, then*

$$\mathcal{L}_t(\theta_{t+1}) \leq \mathcal{L}_t(\theta_t) - \eta_t\left(1 - \frac{M_t\eta_t}{2}\right)\|g_t\|_2^2 < \mathcal{L}_t(\theta_t). \tag{14}$$

This guarantees a decrease in the batch-averaged KL divergence to the step-frozen teacher q_t.

For $d \in \{\ell, L\}$, concatenate the logits $g_\theta(h_i^{(d)})$ on the batch prefixes into $z_d(\theta)$, and let $J_d = \partial z_d/\partial\theta|_{\theta_t}$ and $r_t = \nabla_{z_\ell}\mathcal{L}_t|_{\theta_t}$. With the teacher stopped, $g_t = J_\ell^\top r_t$. If the logit maps have locally Lipschitz Jacobians, then

$$z_d(\theta_{t+1}) - z_d(\theta_t) = -\eta_t \underbrace{J_d J_\ell^\top}_{K_{d\ell}} r_t + O\big(\eta_t^2\|g_t\|_2^2\big), \qquad d \in \{\ell, L\}, \tag{15}$$

where $K_{d\ell}$ is a block of the empirical tangent kernel (Jacot et al., 2018). Parameter sharing gives the first order terminal logit change $-\eta_t K_{L\ell} r_t$, whose effect on the task surrogate depends on gradient alignment.

For analysis, let $\mathcal{R}_d(\theta)$ be a differentiable task surrogate computed from the logits at loop depth d on a fixed evaluation set $\mathcal{E}$, such as gold-token negative log-likelihood. Define

$$\gamma_{d,t} = \langle \nabla_\theta \mathcal{R}_d(\theta_t), g_t\rangle, \qquad d \in \{\ell, L\}. \tag{16}$$

Here $\gamma_{d,t}$ measures the alignment between the distillation gradient and the task surrogate gradient at depth d; $\gamma_{d,t} > 0$ means that $-g_t$ is a descent direction for $\mathcal{R}_d$.

Theorem 1 (Joint local decrease of task surrogates)**.** *Suppose $\mathcal{R}_d$ has a β_d-Lipschitz gradient, $\beta_d > 0$, in a neighborhood of the update segment for each $d \in \{\ell, L\}$. If $\gamma_{\ell,t} > 0$, $\gamma_{L,t} > 0$, and*

$$0 < \eta_t < \min_{d\in\{\ell,L\}} \frac{2\gamma_{d,t}}{\beta_d\|g_t\|_2^2}, \tag{17}$$

then $\mathcal{R}_\ell(\theta_{t+1}) < \mathcal{R}_\ell(\theta_t)$ and $\mathcal{R}_L(\theta_{t+1}) < \mathcal{R}_L(\theta_t)$ simultaneously.

For evaluation logits $\widetilde{z}_d$ on $\mathcal{E}$, write $\widetilde{J}_d = \partial\widetilde{z}_d/\partial\theta|_{\theta_t}$ and $a_{d,t} = \nabla_{\widetilde{z}_d}\mathcal{R}_d|_{\theta_t}$. Then

$$\gamma_{d,t} = a_{d,t}^\top \widetilde{J}_d J_\ell^\top r_t, \qquad d \in \{\ell, L\}. \tag{18}$$

This expression links the training time distillation update to the evaluation task gradients at both loop depths. Appendix A provides more details.

5 Experiments

5.1 Experimental Setup

Models. We conduct experiments on Ouro-1.4B-Thinking and Ouro-2.6B-Thinking (Zhu et al., 2025). For comparison across model scales, we include Qwen3-1.7B, Qwen3-4B, Qwen3-8B (Yang et al., 2025), Qwen3.5-4B, and Qwen3.5-9B (Team, 2026) in thinking mode.

Baselines and Training Dataset. For Ouro, we compare LoopOPD and D-LoopOPD against GRPO (Shao et al., 2024), RLTT (Williams et al., 2026), and OPSD (Zhao et al., 2026). We evaluate OPD (Agarwal et al., 2024) on Ouro-1.4B-Thinking, using Ouro-2.6B-Thinking as the teacher.

For Qwen3, we evaluate the vanilla models and GRPO variants, while for Qwen3.5 we evaluate the vanilla models. All post-training methods are trained on DAPO-Math-17k (Yu et al., 2025).

Evaluation Benchmarks. We evaluate mathematical reasoning on AIME 2024 (Zhang & Math-AI, 2024), AIME 2025 (Zhang & Math-AI, 2025), AIME 2026 (Zhang & Math-AI, 2026), OlympiadBench (He et al., 2024), and BeyondAIME (ByteDance-Seed, 2025). For general reasoning and code generation, we evaluate on MMLU (Hendrycks et al., 2021), MBPP and MBPP+ (Austin et al., 2021; Liu et al., 2023), and HumanEval and HumanEval+ (Chen et al., 2021; Liu et al., 2023). Detailed training and evaluation settings are provided in Appendix D.

5.2 Main Results

Mathematical Reasoning. As shown in Table 2, LoopOPD and D-LoopOPD improve mathematical reasoning on Ouro-2.6B-Thinking. Averaging Avg@16 across AIME 2024, AIME 2025, and AIME 2026, the vanilla model and GRPO achieve 55.49 and 56.60, while LoopOPD and D-LoopOPD improve to 59.37 and 62.36. D-LoopOPD achieves the best Avg@16 on all three AIME benchmarks within the Ouro-2.6B group, improving over LoopOPD by 6.46, 1.46, and 1.05 points on AIME 2024, AIME 2025, and AIME 2026, respectively. On AIME 2024, D-LoopOPD reaches 71.67 Avg@16, surpassing vanilla Qwen3-8B at 71.25 and approaching vanilla Qwen3.5-9B at 72.08 with fewer parameters. On OlympiadBench, D-LoopOPD also achieves the best result at 65.77, while its BeyondAIME score of 34.00 remains comparable to those of the vanilla model and GRPO. Similar improvements are observed on Ouro-1.4B-Thinking, where LoopOPD and D-LoopOPD outperform the vanilla model, GRPO, and OPD on average across the three AIME benchmarks; LoopOPD achieves the best OlympiadBench score and D-LoopOPD the best BeyondAIME score.

Table 2: **Performance comparison on mathematical reasoning benchmarks.** AIME24, AIME25, and AIME26 are reported with Avg@16 and Pass@16, while OlympiadBench and BeyondAIME are reported with accuracy. All evaluations use a maximum response length of 31,744 tokens. Best results within each Ouro model group are shown in **bold**.

Model	Method	AIME24		AIME25		AIME26		Olympiad Bench	Beyond AIME
		Avg@16	Pass@16	Avg@16	Pass@16	Avg@16	Pass@16		
Qwen3-1.7B (Thinking)	Vanilla	47.92	76.67	36.46	56.67	38.13	76.67	56.30	15.00
	GRPO	51.04	73.33	37.50	56.67	40.42	80.00	57.19	17.00
Qwen3-4B (Thinking)	Vanilla	61.25	83.33	52.08	73.33	52.92	76.67	68.44	31.00
	GRPO	65.42	83.33	51.67	76.67	55.21	80.00	68.89	32.00
Qwen3.5-4B (Thinking)	Vanilla	54.79	90.00	51.46	73.33	55.00	80.00	61.63	23.00
Ouro-1.4B-Thinking ($d = 4$)	Vanilla	51.25	73.33	41.25	70.00	45.21	73.33	59.40	24.00
	GRPO	51.88	76.67	40.21	70.00	46.46	70.00	60.14	24.00
	RLTT	50.00	80.00	37.50	66.67	41.87	66.67	60.44	22.00
	OPSD	49.58	73.33	41.04	63.33	44.79	70.00	60.59	22.00
	OPD	50.42	83.33	38.54	66.67	46.25	73.33	60.44	25.00
	LoopOPD	53.33	80.00	**43.54**	70.00	46.25	73.33	**60.74**	24.00
	D-LoopOPD	**55.00**	**86.67**	**43.54**	**73.33**	**47.92**	**76.67**	60.59	**27.00**
Qwen3-8B (Thinking)	Vanilla	71.25	90.00	64.38	83.33	66.67	86.67	69.48	37.00
	GRPO	73.54	93.33	68.96	83.33	69.38	86.67	71.26	38.00
Qwen3.5-9B (Thinking)	Vanilla	72.08	93.33	61.46	86.67	66.04	86.67	64.89	36.00
Ouro-2.6B-Thinking ($d = 4$)	Vanilla	62.08	83.33	48.96	73.33	55.42	83.33	64.30	**35.00**
	GRPO	62.92	86.67	49.79	73.33	57.08	86.67	64.59	**35.00**
	RLTT	56.67	83.33	36.46	66.67	46.04	76.67	60.29	28.00
	OPSD	60.42	83.33	43.13	70.00	50.62	73.33	63.26	30.00
	LoopOPD	65.21	**90.00**	53.75	**80.00**	59.17	**90.00**	64.88	33.00
	D-LoopOPD	**71.67**	**90.00**	**55.21**	**80.00**	**60.21**	86.67	**65.78**	34.00

General Reasoning and Code Generation. Table 3 further examines performance beyond mathematical reasoning. On Ouro-2.6B-Thinking, D-LoopOPD improves over the vanilla model on all five benchmarks, with gains of 0.58 on MMLU, 1.06 on MBPP, 1.33 on MBPP+, 0.61 on HumanEval, and 0.61 on HumanEval+. It achieves the best results within the model group on MMLU, MBPP, and MBPP+. Similar gains are observed on Ouro-1.4B-Thinking, where LoopOPD achieves the best MMLU result, while D-LoopOPD achieves the best results on MBPP, MBPP+, and HumanEval+. These gains extend to general reasoning and code generation even though post-training uses only mathematical data.

Table 3: **Performance comparison on general reasoning and code generation benchmarks.** MMLU is reported with accuracy, while MBPP, MBPP+, HumanEval, and HumanEval+ are reported with Pass@1. Best results within each Ouro model group are shown in **bold**.

Model	Method	MMLU	MBPP	MBPP+	HumanEval	HumanEval+
Qwen3-1.7B (Thinking)	Vanilla	60.21	59.26	50.26	69.51	63.41
	GRPO	61.00	60.05	50.00	71.34	64.63
Qwen3-4B (Thinking)	Vanilla	70.07	68.78	59.79	77.44	70.73
	GRPO	68.49	67.99	60.85	75.61	69.51
Qwen3.5-4B (Thinking)	Vanilla	69.66	75.40	61.11	82.93	76.83
Ouro-1.4B-Thinking ($d = 4$)	Vanilla	67.78	72.22	62.70	74.39	70.12
	GRPO	67.51	72.49	62.70	75.00	**71.95**
	RLTT	68.01	73.02	62.70	**77.44**	71.34
	OPSD	66.50	71.96	61.11	74.39	69.51
	OPD	67.83	72.22	62.43	74.39	70.12
	LoopOPD	**68.05**	72.22	62.70	76.22	71.34
	D-LoopOPD	67.87	**73.28**	**63.23**	76.22	**71.95**
Qwen3-8B (Thinking)	Vanilla	74.89	71.69	62.43	84.15	73.78
	GRPO	74.48	71.96	63.23	85.37	75.00
Qwen3.5-9B (Thinking)	Vanilla	69.90	79.10	64.81	86.59	82.32
Ouro-2.6B-Thinking ($d = 4$)	Vanilla	76.09	74.87	62.43	78.66	74.39
	GRPO	76.32	75.13	62.17	78.05	74.39
	RLTT	76.11	75.40	61.90	**81.71**	**77.44**
	OPSD	75.84	74.87	62.70	76.83	72.56
	LoopOPD	76.09	75.40	62.70	78.66	74.39
	D-LoopOPD	**76.67**	**75.93**	**63.76**	79.27	75.00

5.3 Training Dynamics and Cross-Loop Performance

Training Dynamics. Figure 3 (a) compares LoopOPD and D-LoopOPD during training. Both the rollout reward and validation performance are measured at the student loop $\ell = 2$. D-LoopOPD achieves higher training reward and higher validation Avg@16 than LoopOPD. These trends support the benefit of dynamically using the current terminal loop policy as the teacher for student learning. **Cross-Loop Performance.** Figure 3 (b) evaluates the trained models across all four loops.

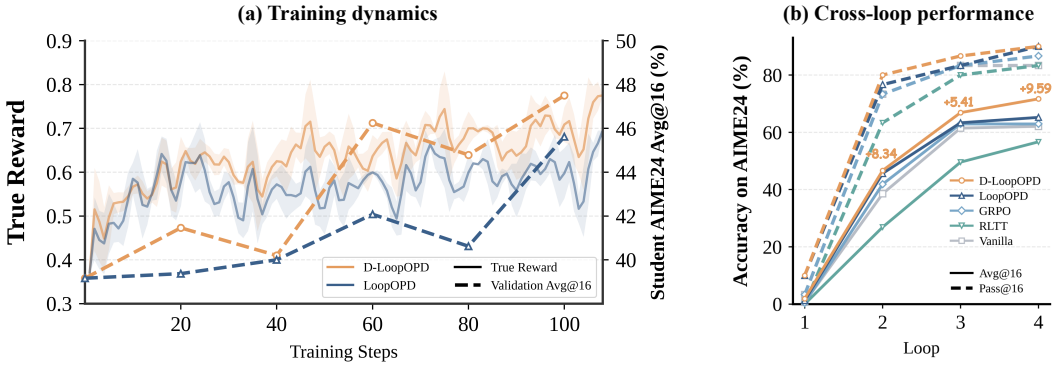

Figure 3: **Training dynamics and cross-loop performance of Ouro-2.6B-Thinking.** (a) Training reward and student loop validation Avg@16 for LoopOPD and D-LoopOPD over training. (b) Avg@16 and Pass@16 across loop depths for different methods. All evaluations use a maximum response length of 31,744 tokens. Exact values are provided in Appendix C.

Both LoopOPD and D-LoopOPD improve Avg@16 over the vanilla model at every depth, with D-LoopOPD achieving the highest Avg@16 among the compared methods. At the directly supervised loop 2, D-LoopOPD raises Avg@16 from 39.16 to 47.50, a gain of 8.34 percentage points. The gains extend to loops 3 and 4, where Avg@16 increases by 5.41 and 9.59 points to 66.87 and 71.67. These results support supervising intermediate computation as an effective way to improve both intermediate and terminal policies.

Teacher and Student Entropy. Figure 4 compares the teacher and student token distribution entropy of LoopOPD and D-LoopOPD during training. D-LoopOPD maintains higher teacher and

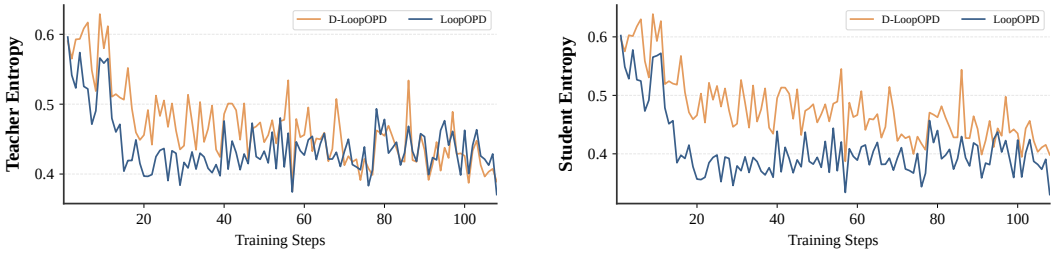

Figure 4: Teacher and student entropy during training for Ouro-2.6B-Thinking. **Left:** entropy of the terminal loop teacher. **Right:** entropy of the student loop $\ell = 2$. Both are measured on student generated rollouts.

student entropy than LoopOPD through most of training, with the gap narrowing toward later steps. This suggests that dynamically refreshing the terminal loop teacher maintains less peaked distributions during training, which support more diverse student rollouts and broader exploration (Jin et al., 2026; Cui et al., 2025). More details are provided in Appendix C.

5.4 Ablation Study

We study the effect of student depth in D-LoopOPD by using loop 1, loop 2, or loop 3 as the student while keeping the terminal teacher at loop 4. As shown in Figure 5, loop 2 yields the best terminal performance, improving loop 4 Avg@16 from 62.08 to 71.67. Loop 3 already starts close to the initial terminal policy, with Avg@16 of 61.46 versus 62.08, leaving less room for distillation. In contrast, loop 1 has a much larger gap to the teacher but very low standalone accuracy. Its student Avg@16 remains at 0.63 after training, although the terminal loop improves to 68.54. Loop 2 provides a favorable balance between student capability and teacher advantage: its student Avg@16 improves from 39.16 to 47.50, while the terminal loop gains 9.59 points. These results indicate that neither a larger teacher-student gap nor a stronger student alone determines the effectiveness of cross-loop distillation. Overall, the intermediate loop $\ell = 2$ yields the strongest terminal loop improvement in our setting.

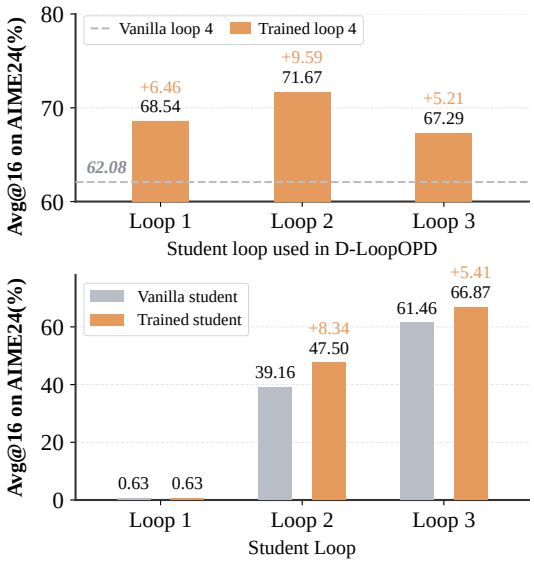

Figure 5: Student depth ablation for D-LoopOPD on Ouro-2.6B-Thinking. AIME 2024 Avg@16 at the student and terminal loops.

6 Conclusion

We propose LoopOPD, which uses a frozen terminal loop policy to supervise an intermediate loop student through on-policy distillation, without an external teacher or privileged information. D-LoopOPD instead uses the current terminal loop policy as the teacher, allowing student updates to shape subsequent supervision through shared parameters. Our analysis characterizes this interaction and gives sufficient conditions for a single gradient update to improve task surrogates locally at both loop depths. Experiments on Ouro-Thinking models show gains in mathematical reasoning, with D-LoopOPD further improving mathematical performance over LoopOPD. Cross-loop evaluation shows that the gains extend beyond the directly supervised student to deeper loops. D-LoopOPD also improves general reasoning and code generation over the vanilla models despite using only mathematical training data. These findings support additional recurrent computation as an effective source of supervision for post-training LoopLMs.

AI USE STATEMENT

In this work, we used generative AI tools for executing existing research scripts, organizing and analyzing experimental results, and checking mathematical derivations written by the authors. We have not used generative AI tools for drafting sections of the paper, generating synthetic datasets, or generating or proving mathematical claims, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for literature retrieval and manuscript polishing. We have reviewed all AI-assisted work. For literature retrieved with AI assistance, the authors verified that each suggested reference corresponds to a real publication, read the cited work, and confirmed its relevance to the associated statements. We also checked the mathematical arguments and empirical results for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies post-training of looped language models through computational experiments using existing model checkpoints and benchmark datasets. Our evaluation focuses on mathematical reasoning, general reasoning, and code generation; it does not assess safety, fairness, or privacy protection. Potential risks include incorrect or biased outputs and misuse of reasoning or code generation capabilities. The proposed objectives do not specifically mitigate these risks, and benchmark improvements should not be interpreted as evidence of safe deployment. Downstream applications require application specific safety evaluation and appropriate safeguards.

REPRODUCIBILITY STATEMENT

Sections 4.1 and 4.2 specify the training objectives and update procedures, with pseudocode in Appendix B. The theoretical assumptions are stated in Section 4.3, and the proofs are provided in Appendix A. Section 5.1 identifies the models, training data, and evaluation benchmarks. Appendix D documents the hardware, training hyperparameters, evaluation prompts, decoding settings, and scoring procedures. Appendix C provides supplementary analyses and numerical results. Code for the proposed methods is provided in an anonymous GitHub repository accompanying this submission.

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 B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5be69a584901a26c521c2b51e40a4c20-Paper-Conference.pdf.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

ByteDance-Seed. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads. [https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME(https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME), 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec

Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 41340–41391. Curran Associates, Inc., 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Paper-Conference.pdf.

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.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *International Conference on Machine Learning (2026)*, 2026.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Renren Jin, Pengzhi Gao, Yuqi Ren, Zhuowen Han, Tongxuan Zhang, Wuwei Huang, Wei Liu, Jian Luan, and Deyi Xiong. Revisiting entropy in reinforcement learning for large reasoning models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25300–25322, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1266. URL https://aclanthology.org/2026.findings-acl.1266/.

Simran Kaur, Narutatsu Ri, Yinghui He, Liam H Fowl, and Sanjeev Arora. Rethinking on-policy self-distillation for thinking models. In *ICML 2026 Workshop on Foundations of Deep Generative Models: Understanding Memorization, Generalization, and Reasoning*, 2026. URL https://openreview.net/forum?id=VhCJItwQHn.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

Hangliang Ren. LSRL: Process-supervised GRPO on latent recurrent states improves mathematical reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12534–12545, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.669. URL https://aclanthology.org/2025.findings-emnlp.669/.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/2676109d49d1eb26d6bc584a8f556305-Paper-Conference.pdf.

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

Guo Tang, Shixin Jiang, Heng Chang, Nuo Chen, Yuhan Li, Huiming Fan, Jia Li, Ming Liu, and Bing Qin. Looprpt: Reinforcement pre-training for looped language models. *arXiv preprint arXiv:2603.19714*, 2026.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jonathan Williams, Olga Russakovsky, and Esin Tureci. Prioritize the process, not just the outcome: Rewarding latent thought trajectories improves reasoning in looped language models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=cPAgJwcn3h.

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. *arXiv preprint arXiv:2505.09388*, 2025.

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

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

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

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2026, 2026.

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. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Jpxfof0EaS.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A PROOFS

Scope and Normalization. We condition on the rollout batch $\mathcal{B}_t$ in equation 13. The sampled tokens, response lengths, and teacher probabilities are held fixed during differentiation. Hidden states and logits on these fixed token prefixes are functions of the trainable parameters. We use the reverse KL over the softmax distributions in equation 1, assuming finite logits and a finite vocabulary so that all probabilities are positive. Each token occurrence (j, i) has weight $w_{j,i} = 1/(n_t m_j)$, so that $\sum_j \sum_i w_{j,i} = 1$. These weights match the per response normalization in equation 11.

Reverse-KL Gradient. For a token occurrence (j, i), abbreviate

$$\begin{aligned} p_{j,i}(v) &= P_{\theta_t}^{(\ell)}(v \mid c_{j,i}), \qquad & q_{j,i}(v) &= q_t(v \mid c_{j,i}), \\ \kappa_{j,i} &= \sum_{v \in \mathcal{V}} p_{j,i}(v) \log \frac{p_{j,i}(v)}{q_{j,i}(v)}. \end{aligned} \tag{19}$$

Let $z_{\ell,j,i,v}$ be the corresponding student logit. The block of $r_t = \nabla_{z_\ell} \mathcal{L}_t|_{\theta_t}$ is

$$(r_t)_{j,i,v} = w_{j,i} p_{j,i}(v) \left(\log \frac{p_{j,i}(v)}{q_{j,i}(v)} - \kappa_{j,i} \right). \tag{20}$$

To verify this, use $\partial p(u)/\partial z_v = p(u)(\mathbf{1}\{u = v\} - p(v))$ and $\partial D_{\mathrm{KL}}(p\|q)/\partial p(u) = \log(p(u)/q(u)) + 1$. Summing over u gives

$$\begin{aligned} \frac{\partial D_{\mathrm{KL}}(p\|q)}{\partial z_v} &= p(v) \left(\log \frac{p(v)}{q(v)} + 1 \right) - p(v) \sum_u p(u) \left(\log \frac{p(u)}{q(u)} + 1 \right) \\ &= p(v) \left(\log \frac{p(v)}{q(v)} - D_{\mathrm{KL}}(p\|q) \right). \end{aligned}$$

Multiplying by $w_{j,i}$ proves equation 20. Since q_t is stopped, the chain rule gives $g_t = J_\ell^\top r_t$.

Proof of Proposition 1. For a differentiable function F with an M-Lipschitz gradient in a neighborhood of the segment from θ to $\theta + \Delta$, the fundamental theorem of calculus and Cauchy–Schwarz give

$$\begin{aligned} F(\theta + \Delta) - F(\theta) &= \langle \nabla F(\theta), \Delta \rangle + \int_0^1 \langle \nabla F(\theta + s\Delta) - \nabla F(\theta), \Delta \rangle \, ds \\ &\le \langle \nabla F(\theta), \Delta \rangle + \frac{M}{2} \|\Delta\|_2^2. \end{aligned} \tag{21}$$

Apply equation 21 at $\theta = \theta_t$ with $F = \mathcal{L}_t$ and $\Delta = -\eta_t g_t$. Since $\nabla \mathcal{L}_t(\theta_t) = g_t$,

$$\begin{aligned} \mathcal{L}_t(\theta_{t+1}) &\le \mathcal{L}_t(\theta_t) - \eta_t \|g_t\|_2^2 + \frac{M_t \eta_t^2}{2} \|g_t\|_2^2 \\ &= \mathcal{L}_t(\theta_t) - \eta_t \left(1 - \frac{M_t \eta_t}{2} \right) \|g_t\|_2^2. \end{aligned}$$

Since $g_t \neq 0$ and $0 < \eta_t < 2/M_t$, $\eta_t(1 - M_t\eta_t/2)\|g_t\|_2^2 > 0$, proving equation 14.

Proof of equation 15. For $d \in \{\ell, L\}$, assume z_d is differentiable and its Jacobian is C_d-Lipschitz in operator norm in a neighborhood of the update segment. Let $J_d(\theta) = \partial z_d(\theta)/\partial \theta$, with $J_d = J_d(\theta_t)$. Integration gives

$$z_d(\theta_t + \Delta) - z_d(\theta_t) - J_d \Delta = \int_0^1 \big(J_d(\theta_t + s\Delta) - J_d \big) \Delta \, ds. \tag{22}$$

The norm of the right-hand side is at most $C_d \|\Delta\|_2^2/2$. With $\Delta = -\eta_t g_t$ and $g_t = J_\ell^\top r_t$, this gives

$$\big\| z_d(\theta_{t+1}) - z_d(\theta_t) + \eta_t J_d J_\ell^\top r_t \big\|_2 \le \frac{C_d \eta_t^2}{2} \|g_t\|_2^2, \tag{23}$$

and hence equation 15. The joint empirical tangent kernel has the block form

$$\begin{bmatrix} J_\ell J_\ell^\top & J_\ell J_L^\top \\ J_L J_\ell^\top & J_L J_L^\top \end{bmatrix} = \begin{bmatrix} J_\ell \\ J_L \end{bmatrix} \begin{bmatrix} J_\ell \\ J_L \end{bmatrix}^\top . \tag{24}$$

Thus, $K_{L\ell} = J_L J_\ell^\top$ is the cross-loop block of the empirical tangent kernel (Jacot et al., 2018).

Proof of Theorem 1. For each $d \in \{\ell, L\}$, apply equation 21 at $\theta = \theta_t$ with $F = \mathcal{R}_d$ and $\Delta = -\eta_t g_t$. This gives

$$\mathcal{R}_d(\theta_{t+1}) - \mathcal{R}_d(\theta_t) \le -\eta_t \gamma_{d,t} + \frac{\beta_d \eta_t^2}{2} \|g_t\|_2^2. \tag{25}$$

Since $\gamma_{\ell,t} > 0$ and $\gamma_{L,t} > 0$, the step-size condition in equation 17 makes the right-hand side strictly negative for both $d \in \{\ell, L\}$, proving Theorem 1.

Proof of equation 18. Let $\widetilde{z}_d$ concatenate the logits on the fixed evaluation set $\mathcal{E}$, and let $\widetilde{J}_d = \partial \widetilde{z}_d / \partial \theta|_{\theta_t}$ and $a_{d,t} = \nabla_{\widetilde{z}_d} \mathcal{R}_d|_{\theta_t}$. By the chain rule,

$$\gamma_{d,t} = \left(\widetilde{J}_d^\top a_{d,t}\right)^\top J_\ell^\top r_t = a_{d,t}^\top \widetilde{J}_d J_\ell^\top r_t, \qquad d \in \{\ell, L\}, \tag{26}$$

which proves equation 18.

Cross-Loop Gradient Alignment. Consider binary next token prediction with a shared recurrent map and output head. Let token 1 be correct, with logits $(a, 0)$ and probability $\sigma(a) = (1 + e^{-a})^{-1}$. In both examples, we use a single fixed prefix c with unit weight, only the scalar parameter θ is trainable, and take

$$\mathcal{R}_d(\theta) = -\log P_\theta^{(d)}(1 \mid c).$$

For a positively aligned case, let $h^{(0)} = 0$, $h^{(k)} = h^{(k-1)} + \theta$, and use logits $(h^{(d)}, 0)$. Take $\ell = 2$, $L = 4$, and $\theta_t > 0$. Then $p = \sigma(2\theta_t)$ and $q = \sigma(4\theta_t) > p$. The stopped-target reverse-KL gradient is

$$g_t = 2p(1-p)(2\theta_t - 4\theta_t) < 0. \tag{27}$$

Thus a gradient step increases θ, increasing the correct token probabilities at both depths and decreasing both negative log-likelihoods. For a contrasting case, consider

$$h^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad h^{(k)} = A h^{(k-1)} + \theta \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad A = \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix}, \tag{28}$$

with the shared head $g(h) = ((1, -2)h, 0)$. Direct substitution gives $h^{(2)} = (\theta, \theta)^\top$ and $h^{(4)} = (\theta, 0)^\top$, so the correct token logits are $-\theta$ at loop 2 and θ at loop 4. At $\theta_t = 1$, $p = \sigma(-1) < q = \sigma(1)$. The stopped-target gradient is

$$g_t = -p(1-p)(-1-1) = 2p(1-p) > 0, \qquad \theta_{t+1} = 1 - \eta_t g_t < 1. \tag{29}$$

The student probability $\sigma(-\theta)$ increases while the terminal probability $\sigma(\theta)$ decreases. Moreover, $\mathcal{R}'_\ell(1) = 1 - p > 0$ and $\mathcal{R}'_L(1) = q - 1 < 0$, giving $\gamma_{\ell,t} > 0$ and $\gamma_{L,t} < 0$. Thus, a stronger terminal teacher does not by itself imply $\gamma_{L,t} > 0$.

B ALGORITHMS

Algorithms 1 and 2 describe the training procedures in Sections 4.1 and 4.2. Each iteration samples G responses for each of B prompts, giving $n_t = BG$ responses. Teacher and student distributions are computed on the same student generated prefixes. $\mathrm{OptimizerStep}$ denotes one update with the chosen optimizer.

Algorithm 1 LoopOPD: Cross-Loop On-Policy Distillation

Require: Training prompts $\mathcal{D}$; initial parameters θ_0; student depth $\ell < L$; terminal depth L; prompt batch size B; rollouts per prompt G; number of updates T; learning rates $\{\eta_t\}_{t=0}^{T-1}$; optimizer.
Ensure: Trained parameters θ_T, defining $\pi_{\theta_T}^{(\ell)}$ and $\pi_{\theta_T}^{(L)}$.

Initialize the trainable LoopLM with θ_0.
Create a frozen teacher copy with $\bar{\theta}_0 \leftarrow \theta_0$.
for $t = 0, \ldots, T-1$ **do**
  Student rollouts
  Draw B prompts $x_1, \ldots, x_B$ from $\mathcal{D}$.
  for $b = 1, \ldots, B$ **do**
    Sample $y^{(b,s)} \sim \pi_{\theta_t}^{(\ell)}(\cdot \mid x_b)$, $s = 1, \ldots, G$.
  end for
  Flatten the responses into $\mathcal{B}_t = \{(x^{(j)}, y^{(j)})\}_{j=1}^{n_t}$, where $n_t = BG$.
  Set $m_j \leftarrow |y^{(j)}|$ and $c_{j,i} \leftarrow (x^{(j)}, y_{<i}^{(j)})$ for $j = 1, \ldots, n_t$, $i = 1, \ldots, m_j$.
  Frozen teacher targets
  Evaluate the frozen teacher at depth L on the batch prefixes.
  $q_{j,i}(v) \leftarrow P_{\bar{\theta}_0}^{(L)}(v \mid c_{j,i})$ for all (j,i) and $v \in \mathcal{V}$.
  Evaluate the trainable LoopLM at depth ℓ on the same prefixes.
  $p_{j,i}(v) \leftarrow P_{\theta_t}^{(\ell)}(v \mid c_{j,i})$ for all (j,i) and $v \in \mathcal{V}$.
  $\kappa_{j,i} \leftarrow \sum_{v \in \mathcal{V}} p_{j,i}(v) \log \frac{p_{j,i}(v)}{q_{j,i}(v)}$ for all (j,i).
  $\mathcal{L}_t \leftarrow \frac{1}{n_t} \sum_{j=1}^{n_t} \frac{1}{m_j} \sum_{i=1}^{m_j} \kappa_{j,i}$.
  $g_t \leftarrow \nabla_\theta \mathcal{L}_t|_{\theta=\theta_t}$.
  $\theta_{t+1} \leftarrow \mathrm{OptimizerStep}(\theta_t, g_t, \eta_t)$.
end for
return θ_T.

Algorithm 2 D-LoopOPD: Dynamic Cross-Loop On-Policy Distillation

Require: Training prompts $\mathcal{D}$; initial parameters θ_0; student depth $\ell < L$; terminal depth L; prompt batch size B; rollouts per prompt G; number of updates T; learning rates $\{\eta_t\}_{t=0}^{T-1}$; optimizer.
Ensure: Trained parameters θ_T, defining $\pi_{\theta_T}^{(\ell)}$ and $\pi_{\theta_T}^{(L)}$.

Initialize the trainable LoopLM with θ_0.
for $t = 0, \ldots, T-1$ **do**
  Student rollouts
  Draw B prompts $x_1, \ldots, x_B$ from $\mathcal{D}$.
  for $b = 1, \ldots, B$ **do**
    Sample $y^{(b,s)} \sim \pi_{\theta_t}^{(\ell)}(\cdot \mid x_b)$, $s = 1, \ldots, G$.
  end for
  Flatten the responses into $\mathcal{B}_t = \{(x^{(j)}, y^{(j)})\}_{j=1}^{n_t}$, where $n_t = BG$.
  Set $m_j \leftarrow |y^{(j)}|$ and $c_{j,i} \leftarrow (x^{(j)}, y_{<i}^{(j)})$ for $j = 1, \ldots, n_t$, $i = 1, \ldots, m_j$.
  Current teacher targets
  Evaluate the trainable LoopLM at depth L on the batch prefixes.
  $q_{j,i}(v) \leftarrow \mathrm{sg}\left[P_{\theta_t}^{(L)}(v \mid c_{j,i})\right]$ for all (j,i) and $v \in \mathcal{V}$.
  Evaluate the trainable LoopLM at depth ℓ on the same prefixes.
  $p_{j,i}(v) \leftarrow P_{\theta_t}^{(\ell)}(v \mid c_{j,i})$ for all (j,i) and $v \in \mathcal{V}$.
  $\kappa_{j,i} \leftarrow \sum_{v \in \mathcal{V}} p_{j,i}(v) \log \frac{p_{j,i}(v)}{q_{j,i}(v)}$ for all (j,i).
  $\mathcal{L}_t \leftarrow \frac{1}{n_t} \sum_{j=1}^{n_t} \frac{1}{m_j} \sum_{i=1}^{m_j} \kappa_{j,i}$.
  $g_t \leftarrow \nabla_\theta \mathcal{L}_t|_{\theta=\theta_t}$.
  $\theta_{t+1} \leftarrow \mathrm{OptimizerStep}(\theta_t, g_t, \eta_t)$.
  Use $\pi_{\theta_{t+1}}^{(L)}$ as the teacher for the next iteration.
end for
return θ_T.

C SUPPLEMENTARY EXPERIMENTS

C.1 ADDITIONAL TRAINING DYNAMICS

Validation Performance. Table 4 reports the exact AIME 2024 validation Avg@16 values corresponding to Figure 3 (a) for Ouro-2.6B-Thinking at the student loop $\ell = 2$.

Table 4: AIME 2024 validation Avg@16 during training for Ouro-2.6B-Thinking at student loop $\ell = 2$. Best results at each training step are shown in **bold**.

Method	Step 20	Step 40	Step 60	Step 80	Step 100
LoopOPD	39.37	40.00	42.08	40.62	45.62
D-LoopOPD	**41.46**	**40.20**	**46.25**	**44.79**	**47.50**

Response Length and Truncation Rate. Figure 6 compares response length during training, while Figure 7 reports the rollout truncation rate. Due to the GPU memory overhead of retaining learning signals across recurrent loops, we follow the original RLTT training configuration and use a maximum rollout length of 2,048 tokens for RLTT, while LoopOPD and D-LoopOPD use 31,744 tokens. RLTT produces substantially shorter responses and exhibits a substantially higher truncation rate throughout training. In contrast, LoopOPD and D-LoopOPD generate much longer responses while maintaining near zero truncation. This behavior is consistent with the decode budget results in Appendix C.3, where RLTT performs strongly under short decode budgets but saturates as the budget increases, while D-LoopOPD continues to benefit from longer generation. Training RLTT with 31,744 token rollouts exceeded our available memory budget, highlighting the practical limitation of extending per loop reward supervision to long rollouts.

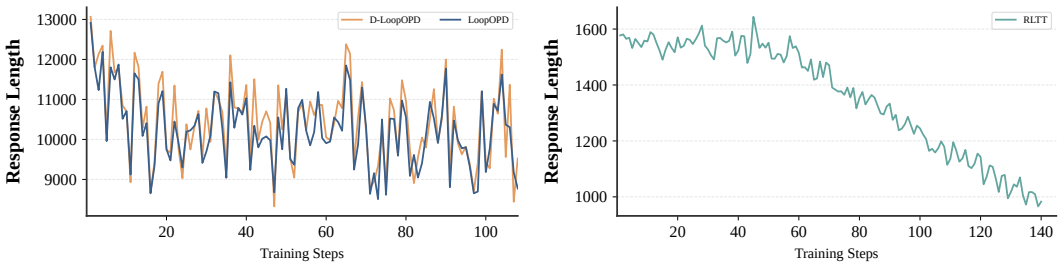

Figure 6: Response length during training for Ouro-2.6B-Thinking. **Left:** LoopOPD and D-LoopOPD. **Right:** RLTT. LoopOPD and D-LoopOPD use a maximum rollout length of 31,744 tokens, while RLTT uses 2,048 tokens.

C.2 CROSS-LOOP PERFORMANCE

Table 5 reports the exact values corresponding to the cross-loop results shown in Figure 3 (b). We evaluate each loop depth of Ouro-2.6B-Thinking on AIME 2024 using a maximum response length of 31,744 tokens. Each entry reports Avg@16 / Pass@16.

C.3 EFFECT OF DECODE BUDGET

Figure 8 shows AIME 2024 Avg@16 of Ouro-2.6B-Thinking under different maximum decode budgets, with exact values reported in Table 6. RLTT performs better under short decode budgets, but its gains diminish as the budget increases. D-LoopOPD overtakes RLTT at 16K tokens and continues to improve at 31K, reaching 71.67 Avg@16 compared with 56.67 for RLTT. Notably, RLTT also falls below the vanilla model at both 16K and 31K, indicating a degradation in its ability to benefit from longer reasoning trajectories. In contrast, D-LoopOPD exceeds the vanilla model across all decode budgets and outperforms it from 2K onward, suggesting that its gains do not come

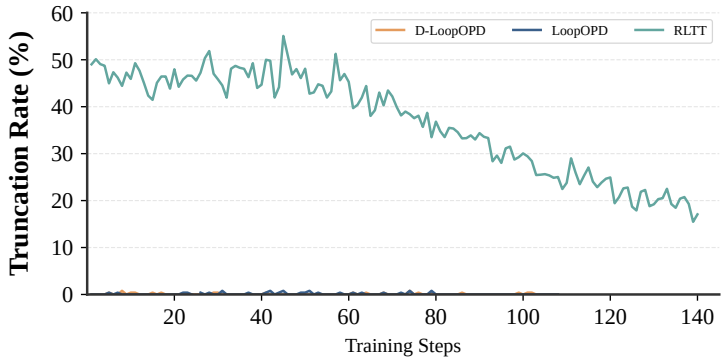

Figure 7: Rollout truncation rate during training for Ouro-2.6B-Thinking. RLTT exhibits substantially higher truncation than LoopOPD and D-LoopOPD throughout training.

Table 5: Cross-loop performance on AIME 2024 for Ouro-2.6B-Thinking. Best results at each loop depth are shown in **bold**.

Method	Loop 1	Loop 2	Loop 3	Loop 4
Base	0.63 / 3.33	39.16 / 73.33	61.46 / 83.33	62.08 / 83.33
RLTT	0.00 / 0.00	26.87 / 63.33	49.58 / 80.00	56.67 / 83.33
GRPO	0.42 / 3.33	41.88 / 73.33	62.92 / 83.33	62.92 / 86.67
LoopOPD	1.25 / **10.00**	45.62 / 76.67	63.33 / 83.33	65.21 / **90.00**
D-LoopOPD	**1.88 / 10.00**	**47.50 / 80.00**	**66.87 / 86.67**	**71.67 / 90.00**

at the cost of long budget reasoning capability. These results are consistent with the response length and truncation behavior in Figures 6 and 7: RLTT favors short generation under its constrained rollout budget, whereas D-LoopOPD continues to benefit from longer generation.

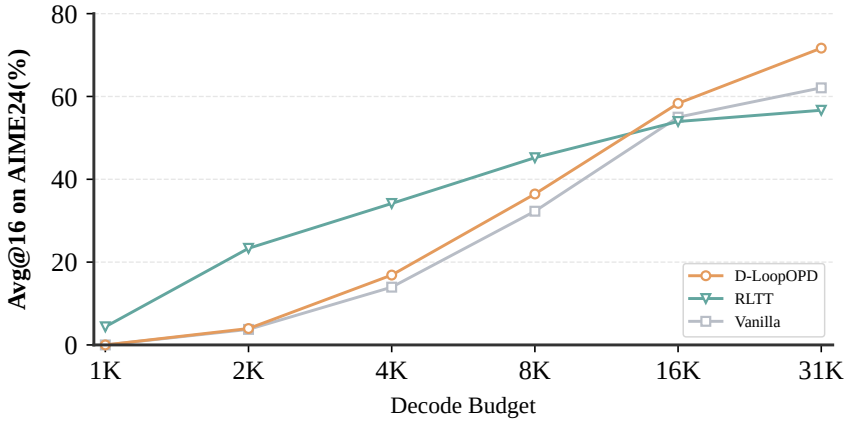

Figure 8: Effect of decode budget on AIME 2024 performance for Ouro-2.6B-Thinking. Avg@16 is evaluated under maximum decode budgets ranging from 1K to 31K tokens.

D Implementation Details

D.1 Training Configuration

We detail the training configurations used throughout our experiments in Table 7. All post-training methods use DAPO-Math-17k as the training dataset, with its basic statistics summarized in Table 8.

Table 6: AIME 2024 Avg@16 under different maximum decode budgets for Ouro-2.6B-Thinking. Best results at each decode budget are shown in **bold**.

Method	1K	2K	4K	8K	16K	31K
Base	0.00	3.75	13.95	32.29	55.00	62.08
RLTT	**4.38**	**23.33**	**34.16**	**45.21**	53.95	56.67
D-LoopOPD	0.00	3.96	16.87	36.45	**58.33**	**71.67**

Each training run uses 4 NVIDIA H200 GPUs. For OPSD, we follow the original setup, using a maximum sequence length of 20,000 and a maximum rollout length of 1,024 tokens. RLTT uses a maximum rollout length of 2,048 tokens and AdamW8bit, following its original training configuration, consistent with the higher GPU memory usage associated with retaining learning signals across recurrent loops. We use uniform weights across recurrent loops. LoopOPD and D-LoopOPD use student depth $\ell = 2$ and terminal depth $L = 4$.

Table 7: Training and validation hyperparameters for different methods.

Hyperparameter	GRPO	RLTT	OPD	OPSD	LoopOPD	D-LoopOPD
General						
Max Prompt Length	1024	1024	1024	–	1024	1024
Optimization						
Global Batch Size	256	512	256	32	256	256
Prompts per Rollout Batch	32	32	32	32	32	32
Group Rollout Size	8	8	8	1	8	8
Gradient Accumulation	1	2	1	1	1	1
Optimizer	AdamW	AdamW8bit	AdamW	AdamW	AdamW	AdamW
Learning Rate	1×10^{-6}	1×10^{-6}	1×10^{-6}	5×10^{-6}	1×10^{-6}	1×10^{-6}
Warmup	10 steps	10%	–	–	–	10 steps
Weight Decay	0.1	0.1	0.01	0	0.01	0.01
Sampling						
Rollout Max Response Length	31744	2048	31744	1024	31744	31744
Rollout Temperature	1.0	1.0	1.0	1.1	1.0	1.0
Rollout Top-p	1.0	1.0	1.0	0.95	1.0	1.0
Validation Max Response Length	31744	31744	31744	31744	31744	31744
Validation Temperature	1.0	1.0	1.0	1.0	1.0	1.0
Validation Top-p	0.7	0.7	0.7	0.7	0.7	0.7
Validation Samples per Problem	16	16	16	16	16	16
Method Specific						
Student Depth ℓ	–	–	4	4	2	2
Teacher Depth L	–	–	4	4	4	4
Frozen Teacher	–	–	Yes	Yes	Yes	No
External Teacher	–	–	Yes	No	No	No
Privileged Information	–	–	No	Yes	No	No

D.2 Evaluation Protocol

Table 9 summarizes the evaluation protocols used throughout our experiments. For mathematical reasoning benchmarks, we use zero-shot generation with thinking enabled. For MMLU, we use the standard 5-shot multiple-choice log-likelihood evaluation, while code generation benchmarks are evaluated using EvalPlus.

Mathematical Reasoning. For all mathematical reasoning benchmarks, each example is provided as a single user message containing the original problem followed by:

Table 8: Training dataset used for post-training.

Dataset	Split	# Examples	Domain	Answer Type
DAPO-Math-17k	Train	17,398	Mathematical Reasoning	Integer Ground Truth

Mathematical Reasoning Prompt

```
{problem}
Please reason step by step, and put your final answer within
\boxed{}.
```

For AIME 2024, AIME 2025, and AIME 2026, we evaluate all 30 problems from each benchmark under the same protocol. Each problem is evaluated with 16 sampled completions using temperature 1.0, top-p 0.7, and a maximum of 31,744 generated tokens. We extract the final boxed answer and determine correctness using numerical and symbolic mathematical equivalence. We report two metrics. Avg@16 is the fraction of correct completions among the $30 \times 16 = 480$ generated responses for each benchmark. Pass@16 is the fraction of the 30 problems for which at least one of the 16 completions is correct.

For OlympiadBench, we evaluate 675 text-based mathematical problems. For BeyondAIME, we use the official test split containing 100 problems. Both benchmarks use temperature 1.0, top-p 0.7, and a maximum of 31,744 generated tokens.

MMLU. We evaluate MMLU using `lm-eval-harness` on the full test split of 14,042 questions across 57 subjects. We use five demonstrations from the development split of each subject together with the subject description. The prompt is formatted as:

MMLU Prompt

```
{subject description}
{five demonstrations}
{question}
A. {choice_0}
B. {choice_1}
C. {choice_2}
D. {choice_3}
Answer:
```

We select the candidate with the highest log-likelihood among A, B, C, and D.

Code Generation. We evaluate MBPP, MBPP+, HumanEval, and HumanEval+ using EvalPlus. MBPP and MBPP+ contain 378 problems, while HumanEval and HumanEval+ contain 164 problems. For each problem, we generate one program and use it to compute both the original and "+" benchmark scores. The original benchmark uses the base test suite, while the "+" variant additionally includes the extended tests. We use greedy decoding with one generated solution per problem and a maximum of 768 new tokens. The prompt is formatted as:

Code Generation Prompt

```
User:
Please provide a self-contained Python script that solves the
following problem in a markdown code block:
```
{task prompt}
```
Assistant:
Below is a Python script with a self-contained function that
solves the problem and passes corresponding tests:
```python
```

Table 9: Evaluation settings for different benchmarks.

Benchmark	# Examples	Decoding / Scoring	Max New Tokens	Metric
AIME 2024	30	$T = 1.0, p = 0.7, n = 16$	31,744	Avg@16 / Pass@16
AIME 2025	30	$T = 1.0, p = 0.7, n = 16$	31,744	Avg@16 / Pass@16
AIME 2026	30	$T = 1.0, p = 0.7, n = 16$	31,744	Avg@16 / Pass@16
OlympiadBench	675	$T = 1.0, p = 0.7, n = 1$	31,744	Accuracy
BeyondAIME	100	$T = 1.0, p = 0.7, n = 1$	31,744	Accuracy
MMLU	14,042	Candidate log-likelihood	–	Accuracy
MBPP	378	Greedy, $n = 1$	768	Pass@1
MBPP+	378	Greedy, $n = 1$	768	Pass@1
HumanEval	164	Greedy, $n = 1$	768	Pass@1
HumanEval+	164	Greedy, $n = 1$	768	Pass@1

E CASE STUDY

We present a qualitative case study to compare the reasoning behaviors of the vanilla model and the D-LoopOPD trained model.

Question and Ground Truth

Question: Let $b \geq 2$ be an integer. Call a positive integer n b-*eautiful* if it has exactly two digits when expressed in base b and these two digits sum to $\sqrt{n}$. For example, 81 is 13-*eautiful* because $81 = \underline{6}\ \underline{3}_{13}$ and $6 + 3 = \sqrt{81}$. Find the least integer $b \geq 2$ for which there are more than ten b-*eautiful* integers.

Ground Truth: 211

Response from Vanilla Model

Okay, so I need to find the smallest integer b (where b is at least 2) such that there are more than ten b-eautiful numbers. A b-eautiful number is a two-digit number in base b where the sum of its digits equals the square root of the number itself. The example given is 81 in base 13, which is written as 63 in base 13, and $6 + 3 = 9$, which is $\sqrt{81}$.

Let me formalize this. For each base b, we need to find all pairs of digits (a, c) where a is between 1 and $b - 1$ (since it's the leading digit in a two-digit number), and c is between 0 and $b - 1$. Then, the number n is $a \cdot b + c$, and we require that $a + c = \sqrt{n}$.

So, substituting n into the equation: $a + c = \sqrt{a \cdot b + c}$. Let's square both sides to eliminate the square root: $(a + c)^2 = a \cdot b + c$.

Expanding the left side: $a^2 + 2ac + c^2 = a \cdot b + c$.
```

Rearranging terms: $a^2 + 2ac + c^2 - a \cdot b - c = 0$.

Hmm, this seems a bit complicated. Maybe there's a better way to approach this. Let's see.

Alternatively, since a and c are digits in base b, their possible values are limited. For a given base b, a can be from 1 to $b-1$, and c can be from 0 to $b-1$. So, for each base b, we can iterate over all possible a and c, compute $n = a \cdot b + c$, check if $a + c$ equals $\sqrt{n}$, and count how many such numbers exist. Then, we need to find the smallest b where this count exceeds 10.

But since we need to find the minimal b where the count is more than ten, maybe there's a pattern or formula we can derive instead of brute-forcing for each b. Let's see.

Starting with the equation: $(a+c)^2 = a \cdot b + c$. Let's denote $s = a + c$. Then, $s^2 = a \cdot b + c$. But since $s = a + c$, we can express c as $s - a$. Substituting into the equation: $s^2 = a \cdot b + (s - a)$. Simplify this: $s^2 = a \cdot b + s - a$. Rearranging terms: $s^2 - s = a \cdot (b - 1)$. Therefore, $a = (s^2 - s)/(b - 1)$.

Since a must be a positive integer between 1 and $b-1$, the right-hand side must also be an integer in that range. So, $(s^2 - s)$ must be divisible by $(b-1)$, and the result must be between 1 and $b-1$.

So, for each possible s, we can compute a and check if it's valid. Then, $c = s - a$, which must be between 0 and $b-1$.

Wait, but in the example given, 81 is a b-eautiful number where $b = 13$. $\sqrt{81} = 9$, which is equal to the sum of digits $6 + 3$. So, in that case, $s = 9$, which is equal to $\sqrt{n}$. So, s must be an integer such that s^2 is a two-digit number in base b. So, s^2 must be between b and $b^2 - 1$. Therefore, s must satisfy $\sqrt{b} \leq s \leq b - 1$. Because $s^2 \leq b^2 - 1$ implies $s \leq b - 1$ (since s is integer). Therefore, s ranges from $\lceil \sqrt{b} \rceil$ to $b - 1$.

Therefore, for each base b, s can be from $\lceil \sqrt{b} \rceil$ to $b - 1$. For each such s, we can compute $a = (s^2 - s)/(b - 1)$. Then, check if a is an integer between 1 and $b - 1$, and $c = s - a$ is between 0 and $b - 1$.

So, the number of b-eautiful numbers is the number of s in $[\lceil \sqrt{b} \rceil, b - 1]$ such that $(s^2 - s)$ is divisible by $(b - 1)$, and the resulting a and c are valid digits.

...

$b = 7$: $s_{\min} = \lceil \sqrt{7} \rceil = 3$; $s_{\max} = 6$; $s = 3, 4, 5, 6$.

$s = 3$: numerator $= 9 - 3 = 6$; $b - 1 = 6$; $6/6 = 1$; $a = 1$; $c = 3 - 1 = 2$; Check $a = 1$ valid, $c = 2$ valid. $n = 1 \cdot 7 + 2 = 9$. Check $1 + 2 = 3 = \sqrt{9}$. Yes. Count $= 1$.

$s = 4$: numerator $= 16 - 4 = 12$; $12/6 = 2$; $a = 2$; $c = 4 - 2 = 2$; Check $a = 2$ valid, $c = 2$ valid. $n = 2 \cdot 7 + 2 = 16$. Check $2 + 2 = 4 = \sqrt{16}$. Yes. Count $= 2$.

$s = 5$: numerator $= 25 - 5 = 20$; $20/6 \approx 3.333$, invalid.

$s = 6$: numerator $= 36 - 6 = 30$; $30/6 = 5$; $a = 5$; $c = 6 - 5 = 1$; Check $a = 5$ valid, $c = 1$ valid. $n = 5 \cdot 7 + 1 = 36$. Check $5 + 1 = 6 = \sqrt{36}$. Yes. Count $= 3$. So, for $b = 7$, count $= 3$.

...

Let's denote $k = b - 1$. Then, we need $(s^2 - s)$ divisible by k. So, k divides $s(s - 1)$. Therefore, k must be a divisor of $s(s-1)$. Since s and $s - 1$ are consecutive integers, they are coprime. Therefore, k must be a product of divisors of s and $s - 1$. But since s and $s - 1$ are coprime, the divisors of k must be products of divisors of s and $s - 1$.

Alternatively, for each k, the number of s in $[\lceil \sqrt{k+1} \rceil, k]$ such that k divides $s(s-1)$. Let's compute this for $k = 24$ ($b = 25$): s ranges from 5 to 24. We need 24 divides $s(s - 1)$. Since $24 = 8 \cdot 3$, and s and $s - 1$ are coprime, one of them must be divisible by 8 and the other by 3.

Annotation: Both 8 and 3 can divide the same number. This misses $s = 24$.

So, either $s \equiv 0 \pmod 8$ and $s \equiv 1 \pmod 3$, or $s \equiv 1 \pmod 8$ and $s \equiv 0 \pmod 3$. Let's solve these congruences.

Case 1: $s \equiv 0 \pmod 8$ and $s \equiv 1 \pmod 3$.

Let $s = 8m$. Then, $8m \equiv 1 \pmod 3 \Rightarrow 2m \equiv 1 \pmod 3 \Rightarrow m \equiv 2 \pmod 3$. So, $m = 3n + 2$. Therefore, $s = 8 \cdot (3n + 2) = 24n + 16$. Within $s \leq 24, n = 0$: $s = 16$; $n = 1$: $s = 40$ (too big). So, $s = 16$.

Case 2: $s \equiv 1 \pmod 8$ and $s \equiv 0 \pmod 3$.

Let $s = 8m + 1$. Then, $8m + 1 \equiv 0 \pmod 3 \Rightarrow 2m + 1 \equiv 0 \pmod 3 \Rightarrow 2m \equiv -1 \equiv 2 \pmod 3 \Rightarrow m \equiv 1 \pmod 3$. So, $m = 3n + 1$. Therefore, $s = 8 \cdot (3n + 1) + 1 = 24n + 9$. Within $s \leq 24, n = 0$: $s = 9$; $n = 1$: $s = 33$ (too big). So, $s = 9$.

Therefore, for $k = 24$, the possible s are 9 and 16. Then, check if these s's give valid a and c.

For $s = 9$: $a = (81 - 9)/24 = 72/24 = 3$; $c = 9 - 3 = 6$; Check $a = 3$ is between 1 and 24 (valid), $c = 6$ is between 0 and 24 (valid). So, $n = 3 \cdot 25 + 6 = 81$. Valid.

For $s = 16$: $a = (256 - 16)/24 = 240/24 = 10$; $c = 16 - 10 = 6$; Check $a = 10$ valid, $c = 6$ valid. $n = 10 \cdot 25 + 6 = 256$. Valid.

...

$s = 24$: numerator $= 576 - 24 = 552, 552/24 = 23, a = 23, c = 1$. Valid.

...

Therefore, our earlier analysis missed $s = 24$ because we considered only $s = 9$ and 16. But $s = 24$ also works. Therefore, the number of solutions is more than 2.

Therefore, the earlier approach of solving congruences missed some cases. So, perhaps the number of solutions is equal to the number of divisors of $k = b - 1$ that divide s or $s - 1$.

Alternatively, since k divides $s(s - 1)$, and s and $s - 1$ are coprime, then k must divide s or k divides $s - 1$.
Wait, no. Since s and $s - 1$ are coprime, k must divide s or k divides $s - 1$. Because if a prime divides k, it can divide either s or $s - 1$, but not both. Therefore, k must be a product of divisors of s and $s - 1$. But since s and $s - 1$ are coprime, k must be a product of divisors of s and divisors of $s - 1$. Therefore, the number of solutions s is equal to the number of ways to write k as a product of two coprime factors, one dividing s and the other dividing $s - 1$.

Annotation: For $k = 24$ and $s = 9$, $24 \mid 9 \cdot 8$, but 24 divides neither 9 nor 8.

Therefore, the number of solutions s is equal to 2^{m-1}, where m is the number of distinct prime factors of k.
Because each prime factor can go to either s or $s - 1$, but since they are coprime, each prime must go entirely to one or the other. Therefore, the number of solutions is 2^{m-1}, but this might not be exact.

Annotation: There are 2^m solutions modulo k, and $2^m - 1$ valid values after excluding $s = 1$.

Wait, for example, if $k = 24 = 2^3 \cdot 3$, then the number of ways to split the prime factors into s and $s - 1$. Since s and $s - 1$ are coprime, all factors of 2 must go to one of them, and all factors of 3 must go to the other. So, either s is divisible by 2^3 and 3, and $s - 1$ is 1, which is impossible since $s \geq 2$. Or, s is divisible by 2^3 and $s - 1$ is divisible by 3. Or, s is divisible by 3 and $s - 1$ is divisible by 2^3. So, two possibilities. Therefore, the number of solutions is 2. But in reality, for $k = 24$, we have three solutions: $s = 9, 16, 24$. So, this approach is not capturing all possibilities.

Annotation: Assigning no factor to $s - 1$ only requires divisibility by 1, not $s - 1 = 1$. For $s = 24$, $s - 1 = 23$ is allowed.

Alternatively, maybe the number of solutions is equal to the number of divisors of k. For example, $k = 24$ has divisors 1, 2, 3, 4, 6, 8, 12, 24. But this doesn't directly relate.

Alternatively, perhaps the number of solutions is equal to the number of divisors of k that are less than or equal to k. But this is vague.

...

Let's check $b = 121$ ($b - 1 = 120$). 120 has many divisors: 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120. So, 16 divisors. Let's see if this leads to more than ten solutions.

For $b = 121, k = 120$. We need to find s in $[\lceil\sqrt{121}\rceil, 120] = [11, 120]$ such that 120 divides $s(s-1)$.

$120 = 2^3 \cdot 3 \cdot 5$. Since s and $s-1$ are coprime, the factors of 120 must be split between s and $s-1$. Each prime factor must go entirely to one or the other.

Possible splits:

s divisible by $8(2^3)$, 3, and 5, and $s - 1 = 1$. But $s - 1 = 1 \Rightarrow s = 2$, which is too small.

s divisible by 8 and 3, and $s - 1$ divisible by 5.

s divisible by 8 and 5, and $s - 1$ divisible by 3.

s divisible by 3 and 5, and $s - 1$ divisible by 8.

s divisible by $24(8 \cdot 3)$, and $s - 1$ divisible by 5.

s divisible by $40(8 \cdot 5)$, and $s - 1$ divisible by 3.

s divisible by $15(3 \cdot 5)$, and $s - 1$ divisible by 8.

s divisible by 120, and $s - 1 = 1$. $s = 121$, which is too big.

Annotation: Several splits are repeated; the opposite assignments giving $s = 16, 25, 81$ are missing. Also, $s - 1 = 1$ gives $s = 2$, not 121.

Let's solve these congruences.

Case 1: $s \equiv 0 \pmod{24}$ and $s \equiv 1 \pmod{5}$.

Let $s = 24m$. Then, $24m \equiv 1 \pmod{5} \Rightarrow 24m \equiv 1 \pmod{5} \Rightarrow 4m \equiv 1 \pmod{5} \Rightarrow m \equiv 4 \pmod{5}$. So, $m = 5n + 4$. Therefore, $s = 24 \cdot (5n + 4) = 120n + 96$. Within $s \leq 120, n = 0$: $s = 96$; $n = 1$: $s = 216$ (too big). So, $s = 96$.

Case 2: $s \equiv 0 \pmod{40}$ and $s \equiv 1 \pmod{3}$.

Let $s = 40m$. Then, $40m \equiv 1 \pmod{3} \Rightarrow 1m \equiv 1 \pmod{3} \Rightarrow m \equiv 1 \pmod{3}$. So, $m = 3n + 1$. Therefore, $s = 40 \cdot (3n + 1) = 120n + 40$. Within $s \leq 120, n = 0$: $s = 40$; $n = 1$: $s = 160$ (too big). So, $s = 40$.

Case 3: $s \equiv 0 \pmod{15}$ and $s \equiv 1 \pmod{8}$.

Let $s = 15m$. Then, $15m \equiv 1 \pmod{8} \Rightarrow 7m \equiv 1 \pmod{8} \Rightarrow m \equiv 7 \pmod{8}$. So, $m = 8n + 7$. Therefore, $s = 15 \cdot (8n + 7) = 120n + 105$. Within $s \leq 120, n = 0$: $s = 105$; $n = 1$: $s = 225$ (too big). So, $s = 105$.

Case 4: s divisible by 8, 3, and 5. $s = 120$, which is too big.

Case 5: $s = 120$, which is too big.

Annotation: The upper endpoint $s = 120$ is included in $[11, 120]$.

Additionally, check if $s = 120$ is allowed. $s = 120, s - 1 = 119$. 120 divides $120 \cdot 119 = 14280$. $14280/120 = 119$, which is integer. So, $s = 120$ is a solution.

Therefore, solutions are $s = 96, 40, 105, 120$.

Check if these give valid a and c.

For $s = 96$: $a = (96^2 - 96)/120 = (9216 - 96)/120 = 9120/120 = 76$; $c = 96 - 76 = 20$; Check $a = 76$ valid ($b = 121, a \leq 120$), $c = 20$ valid. $n = 76 \cdot 121 + 20 = 9216$. Check $76 + 20 = 96 = \sqrt{9216}$. Yes.

For $s = 40$: $a = (1600 - 40)/120 = 1560/120 = 13$; $c = 40 - 13 = 27$; Check $a = 13$ valid, $c = 27$ valid. $n = 13 \cdot 121 + 27 = 1580$. Check $13 + 27 = 40 = \sqrt{1600}$. Yes.

Annotation: $13 \cdot 121 + 27 = 1600$*, not 1580.*

For $s = 105$: $a = (11025 - 105)/120 = 10920/120 = 91$; $c = 105 - 91 = 14$; Check $a = 91$ valid, $c = 14$ valid. $n = 91 \cdot 121 + 14 = 11025$. Check $91 + 14 = 105 = \sqrt{11025}$. Yes.

For $s = 120$: $a = (14400 - 120)/120 = 14280/120 = 119$; $c = 120 - 119 = 1$; Check $a = 119$ valid, $c = 1$ valid. $n = 119 \cdot 121 + 1 = 14400$. Check $119 + 1 = 120 = \sqrt{14400}$. Yes.

So, count $= 4$ for $b = 121$. Still low.

Annotation: The correct count is $2^3 - 1 = 7$*, not 4.*

This suggests that even with $b = 121$, we only get 4 solutions. Therefore, we need to consider even larger b.

Alternatively, maybe the number of solutions increases with the number of divisors of $(b-1)$. For example, if $(b-1) = 60$, which has 12 divisors, perhaps the count is higher.

But manually checking each b is impractical. Given the time I've spent and the lack of progress, I think the answer is likely $b = 121$, but according to our previous calculation, it only gives 4 solutions. Therefore, I must have made a mistake in my approach.

Annotation: Even the correct count of 7 is below the required 11.

Wait, perhaps the key is to realize that for each divisor d of $(b-1)$, there corresponds a solution $s = d$ and $s - 1 = (b-1)/d$, but this requires that $d - 1 = (b-1)/d$, which is a quadratic equation. Solving $d^2 - d - (b-1) = 0$.

Annotation: The factors must divide s *and* $s - 1$*; they need not equal them.*

...

None of these give integer b. Therefore, there are no solutions where s and $s - 1$ are divisors of $(b-1)$ in this way. Therefore, this approach doesn't yield solutions.

Annotation: This family is not empty: $s = d = 2$*,* $b = 3$ *gives the valid number* $n = 4$*.*

Given that I'm stuck, I'll look for patterns in the counts we have:

...

[Response truncated here; no final answer is provided.]

Response from D-LoopOPD

Okay, so I need to find the smallest integer $b \geq 2$ such that there are more than ten b-beautiful numbers. A b-beautiful number is a two-digit number in base b where the sum of its digits equals the square root of the number itself. The example given is 81 being 13-beautiful because in base 13, 81 is written as 63 (since $6 \times 13 + 3 = 81$), and 6 + 3 = 9, which is $\sqrt{81}$.

First, let me make sure I understand the problem correctly. A two-digit number in base b has the form $\overline{xy}_b$, which translates to $x \times b + y$ in decimal. The condition is that the sum of the digits $x + y$ equals the square root of the number $n = x \times b + y$. Therefore, the equation we need is:

$$x + y = \sqrt{n} = \sqrt{x \times b + y}$$

Since n must be a perfect square (because its square root is an integer), $x \times b + y$ must be a perfect square. Also, since $\overline{xy}_b$ is a two-digit number in base b, x must be between 1 and $b-1$, and y must be between 0 and $b - 1$.

So, to find all b-beautiful numbers, we can set up the equation:

$$(x + y)^2 = x \times b + y$$

Let me rearrange this equation to make it easier to handle. Let's denote $s = x + y$, so the equation becomes:

$s^2 = x \times b + y$

But since $s = x + y$, we can substitute $y = s - x$ into the equation:

$s^2 = x \times b + (s - x)$,

$s^2 = x \times (b - 1) + s$,

$s^2 - s = x \times (b - 1)$,

$x = \frac{s^2 - s}{b-1}$.

Since x must be a positive integer between 1 and $b - 1$, the right-hand side must also be an integer in that range. Let's note that $x = \frac{s(s-1)}{b-1}$.

Therefore, for each possible s, we can compute x and check if it's an integer in the valid range. Also, $y = s - x$ must be between 0 and $b - 1$.

...

Alternatively, since $s = x + y$, and $x \geq 1$, $y \geq 0$, so $s \geq 1$. But given that $n = s^2$, which is a two-digit number in base b, so $n \geq b$ (since the smallest two-digit number in base b is $10_b = b$). Therefore, $s^2 \geq b \implies s \geq \sqrt{b}$. Since s must be an integer, so $s \geq \lceil \sqrt{b} \rceil$.

But also, since $n = x \times b + y \leq (b - 1) \times b + (b - 1) = b^2 - 1$. Therefore, $s^2 \leq b^2 - 1 \implies s \leq \lfloor \sqrt{b^2 - 1} \rfloor = b - 1$.

Therefore, s can range from $\lceil \sqrt{b} \rceil$ up to $b - 1$.

...

Let me consider that for each b, the number of b-beautiful numbers is equal to the number of integers s such that:

1. s is an integer with $\sqrt{b} \leq s \leq b - 1$.

2. $b - 1$ divides $s(s - 1)$.

3. $x = \frac{s(s-1)}{b-1}$ is an integer between 1 and $b - 1$.

4. $y = s - x$ is between 0 and $b - 1$.

But actually, conditions 1 and 2 and 3 imply condition 4?

Wait, let's see:

Since $x \geq 1$, then $y = s - x \leq s - 1$. Since $s \leq b - 1$, $y \leq (b - 1) - 1 = b - 2 \leq b - 1$. So that's okay.

Alternatively, since $y = s - x \geq 0$ is equivalent to $x \leq s$, let's check that. We have

$x = \frac{s(s-1)}{b-1} \leq s \quad \iff \quad s - 1 \leq b - 1 \quad \iff \quad s \leq b$.

Which is true since $s \leq b - 1$.

Therefore, as long as $s \leq b - 1$ and x is between 1 and $b - 1$, y will be between 0 and $b - 1$. Therefore, conditions 1–3 are sufficient.

In fact, if $2 \leq s \leq b - 1$ and $b - 1$ divides $s(s - 1)$, then x is a positive integer and $x < s$. So $1 \leq x \leq s - 1$ and $1 \leq y = s - x \leq s - 1$, which are valid digits. Also,

$xb + y = x(b - 1) + s = s(s - 1) + s = s^2$.

Therefore, each such s gives exactly one b-beautiful number, and every b-beautiful number gives one such s. For $b = 2$, there are no such values of s.

Alternatively, perhaps we can find a formula for the number of such s.

Note that $s(s-1) \equiv 0 \pmod{b-1}$. Since s and $s-1$ are consecutive integers, they are coprime. Therefore, each prime power in the factorization of $b-1$ must divide either s or $s-1$.

...

Let me denote $d = b-1$. So we need $s(s-1) \equiv 0 \pmod{d}$.

Therefore, for example, if $d = 12 = 3 \times 4$, we can have 3 dividing s and 4 dividing $s-1$, or vice versa. In the case of $s = 9$, $d = 12$: 3 divides 9 and 4 divides 8 (since $s-1 = 8$). Therefore, $3 \times 4 = 12$ divides $9 \times 8 = 72$.

Therefore, in general, for d to divide $s(s-1)$, we can write d as a product of two coprime integers, say m and n, such that m divides s and n divides $s-1$. Then, $s \equiv 0 \pmod{m}$ and $s \equiv 1 \pmod{n}$. By the Chinese Remainder Theorem, since m and n are coprime, there exists a solution $s \equiv a \pmod{d}$, where a is the solution to the system of congruences.

Let me denote $s \equiv k \pmod{d}$, using k in $[1, d]$. Each residue has exactly one representative k in this range.

Therefore, the equation becomes $k(k-1) \equiv 0 \pmod{d}$.

Therefore, for each divisor m of d, set $n = d/m$, and check if m and n are coprime. Then, the solutions satisfy $k \equiv 0 \pmod{m}$ and $k \equiv 1 \pmod{n}$. By Chinese Remainder Theorem, since m and n are coprime, there is a unique solution modulo d.

Therefore, the number of solutions k in $[1, d]$ is equal to the number of such factorizations of d into ordered coprime pairs (m, n), which is equal to $2^{\omega(d)}$, where $\omega(d)$ is the number of distinct prime factors of d. More precisely, for each prime power in the factorization of d, we can assign it to either m or n. Therefore, if d has the prime factorization $d = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$, then the number of ways to split the prime powers into m and n (coprime) is 2^r, since for each prime power, we can assign it to m or to n. Therefore, the number of coprime factorizations is 2^r, where r is the number of distinct prime factors of d.

But wait, in the example with $d = 12$ ($b = 13$), which has prime factors 2 and 3, so $r = 2$, so $2^2 = 4$ solutions. Let's check:

Let's list all k in $[1, 12]$ such that $k(k-1) \equiv 0 \pmod{12}$.

$k = 1$: $1 \times 0 \equiv 0 \pmod{12} \rightarrow$ yes; $k = 2$: $2 \times 1 \equiv 2 \pmod{12} \rightarrow$ no; $k = 3$: $3 \times 2 \equiv 6 \pmod{12} \rightarrow$ no; $k = 4$: $4 \times 3 = 12 \equiv 0 \pmod{12} \rightarrow$ yes.

$k = 5$: $5 \times 4 = 20 \equiv 8 \pmod{12} \rightarrow$ no; $k = 6$: $6 \times 5 = 30 \equiv 6 \pmod{12} \rightarrow$ no; $k = 7$: $7 \times 6 = 42 \equiv 6 \pmod{12} \rightarrow$ no; $k = 8$: $8 \times 7 = 56 \equiv 8 \pmod{12} \rightarrow$ no.

$k = 9$: $9 \times 8 = 72 \equiv 0 \pmod{12} \rightarrow$ yes; $k = 10$: $10 \times 9 = 90 \equiv 6 \pmod{12} \rightarrow$ no; $k = 11$: $11 \times 10 = 110 \equiv 2 \pmod{12} \rightarrow$ no; $k = 12$: $12 \times 11 = 132 \equiv 0 \pmod{12} \rightarrow$ yes.

Therefore, the solutions are $k = 1, 4, 9, 12$. So 4 solutions, which matches 2^r where $r = 2$ (since 12 has two prime factors). Therefore, the formula holds.

Therefore, the total number of solutions in $[1, d]$ is 2^r, but we need to subtract those $k < \lceil \sqrt{d+1} \rceil$. For $2 \leq k < \lceil \sqrt{d+1} \rceil$, we have $0 < k(k-1) < d$, so none is a solution. Only $k = 1$ is excluded. Therefore, the count we need is $2^r - 1$.

...

Given that the number of solutions in $[1, d]$ is 2^r, where r is the number of distinct prime factors of d, to get more than 10 valid solutions, we need $2^r - 1 > 10$, so $r \geq 4$ (since $2^4 - 1 = 15$). Therefore, we need d with at least 4 distinct prime factors.

Therefore, the minimal d with 4 distinct prime factors is $2 \times 3 \times 5 \times 7 = 210$. Therefore, $b = 211$ is the smallest candidate. Any smaller d has at most 3 distinct prime factors and therefore at most $2^3 - 1 = 7$ valid solutions.

...

Therefore, the formula holds even for prime powers. Therefore, the number of solutions in $[1, d]$ is indeed 2^r, where r is the number of distinct prime factors of d.

Therefore, for $d = 60 = 2^2 \times 3 \times 5$, $r = 3$, so 8 solutions.

Wait, let's properly list all solutions for $d = 60$.

We need $k(k-1) \equiv 0 \pmod{60}$.

This requires that 60 divides $k(k-1)$. Since $60 = 2^2 \times 3 \times 5$, and k and $k-1$ are coprime, we need to distribute the prime powers between k and $k-1$.

Therefore, total possibilities: for the prime factors, 2^2 can be assigned to k or $k-1$; 3 can be assigned to k or $k-1$; 5 can be assigned to k or $k-1$. Therefore, total possibilities: $2 \times 2 \times 2 = 8$. Each corresponds to a different combination.

However, solving each case:

Case 1: $2^2 \to k,\ 3 \to k-1,\ 5 \to k-1$. The solution is $k = 16$.

Case 2: $2^2 \to k,\ 3 \to k,\ 5 \to k-1$. We have $k \equiv 0 \pmod 4$, $k \equiv 0 \pmod 3$, $k \equiv 1 \pmod 5$. Solve $k \equiv 0 \pmod{12}$ (since 4 and 3), and $k \equiv 1 \pmod 5$. Let $k = 12m$. Then $12m \equiv 1 \pmod 5 \to 2m \equiv 1 \pmod 5 \to m \equiv 3 \pmod 5$. So $m = 5n + 3$. Thus, $k = 12(5n + 3) = 60n + 36$. In $[1, 60]$, $k = 36$.

Case 3: $2^2 \to k,\ 3 \to k-1,\ 5 \to k$. We have $k \equiv 0 \pmod 4$, $k \equiv 1 \pmod 3$, $k \equiv 0 \pmod 5$. So $k \equiv 0 \pmod{20}$ (since 4 and 5), and $k \equiv 1 \pmod 3$. Let $k = 20m$. Then $20m \equiv 1 \pmod 3 \to 2m \equiv 1 \pmod 3 \to m \equiv 2 \pmod 3$. So $m = 3n + 2$. Thus, $k = 20(3n + 2) = 60n + 40$. In $[1, 60]$, $k = 40$.

Case 4: $2^2 \to k-1,\ 3 \to k,\ 5 \to k$. We have $k - 1 \equiv 0 \pmod 4 \to k \equiv 1 \pmod 4$, $k \equiv 0 \pmod 3$, $k \equiv 0 \pmod 5$. So $k \equiv 0 \pmod{15}$, and $k \equiv 1 \pmod 4$. Let $k = 15m$. Then $15m \equiv 1 \pmod 4 \to 3m \equiv 1 \pmod 4 \to m \equiv 3 \pmod 4$. So $m = 4n + 3$. Thus, $k = 15(4n + 3) = 60n + 45$. In $[1, 60]$, $k = 45$.

Case 5: $2^2 \to k-1,\ 3 \to k-1,\ 5 \to k$. We have $k \equiv 1 \pmod 4$, $k \equiv 1 \pmod 3$, $k \equiv 0 \pmod 5$. So $k \equiv 1 \pmod{12}$ (since 4 and 3), and $k \equiv 0 \pmod 5$. Let $k = 12m + 1$. Then $12m + 1 \equiv 0 \pmod 5 \to 12m \equiv -1 \pmod 5 \to 2m \equiv 4 \pmod 5 \to m \equiv 2 \pmod 5$. So $m = 5n + 2$. Thus, $k = 12(5n + 2) + 1 = 60n + 25$. In $[1, 60]$, $k = 25$.

Case 6: $2^2 \to k-1,\ 3 \to k,\ 5 \to k-1$. We have $k \equiv 1 \pmod 4$, $k \equiv 0 \pmod 3$, $k \equiv 1 \pmod 5$. Solve $k \equiv 1 \pmod 4$ and $k \equiv 1 \pmod 5 \to k \equiv 1 \pmod{20}$. Also, $k \equiv 0 \pmod 3$. So $k = 20m + 1 \equiv 0 \pmod 3 \to 20m \equiv -1 \pmod 3 \to 2m \equiv 2 \pmod 3 \to m \equiv 1 \pmod 3$. Thus, $m = 3n + 1$. So $k = 20(3n + 1) + 1 = 60n + 21$. In $[1, 60]$, $k = 21$.

Case 7: $2^2 \to k-1,\ 3 \to k-1,\ 5 \to k-1$. We have $k - 1 \equiv 0 \pmod 4$, $k - 1 \equiv 0 \pmod 3$, $k - 1 \equiv 0 \pmod 5 \to k - 1 \equiv 0 \pmod{60} \to k \equiv 1 \pmod{60}$. In $[1, 60]$, $k = 1$.

Case 8: $2^2 \to k,\ 3 \to k,\ 5 \to k$. We have $k \equiv 0 \pmod 4$, $k \equiv 0 \pmod 3$, $k \equiv 0 \pmod 5 \to k \equiv 0 \pmod{60}$. In $[1, 60]$, $k = 60$.

Therefore, the solutions are $k = 1, 16, 21, 25, 36, 40, 45, 60$. That's 8 solutions, which matches the formula.

So in $[1, 60]$, these are the solutions. Now, for $b = 61$ ($d = 60$), we need $k \geq \lceil \sqrt{61} \rceil = 8$.

Therefore, the solutions in $[8, 60]$ are: $16, 21, 25, 36, 40, 45, 60$. That's 7 solutions.

Therefore, for $b = 61$, there are 7 beautiful numbers.

But we need more than 10. So need to go higher.

Next, take $d = 105 = 3 \times 5 \times 7$. Then $r = 3$, so $2^3 = 8$ solutions. Number of solutions in $[\lceil \sqrt{106} \rceil, 105] = [11, 105]$. The solutions k in $[1, 105]$ are 8. Need to count how many are ≥ 11. Only $k = 1$ is below 11, so 7 solutions. Not enough.

Next, $d = 165 = 3 \times 5 \times 11$. $r = 3$, so 8 solutions. We have $\lceil \sqrt{166} \rceil = 13$. Number of solutions in $[13, 165]$: 7, after excluding $k = 1$. Still not enough.

...

For $d = 210$, $\lceil \sqrt{211} \rceil = 15$. The number of solutions in $[15, 210]$ is equal to the number of solutions in $[1, 210]$ minus those in $[1, 14]$.

But to confirm, let's compute the number of solutions in $[1, 14]$ for $d = 210$.

Solutions of $k(k-1) \equiv 0 \pmod{210}$. Possible k from 1 to 14:

Check $k = 1$: $1 \times 0 \equiv 0 \pmod{210} \rightarrow$ yes; $k = 2$: $2 \times 1 \equiv 2 \pmod{210} \rightarrow$ no; $k = 3$: $3 \times 2 \equiv 6 \pmod{210} \rightarrow$ no; $k = 4$: $4 \times 3 \equiv 12 \pmod{210} \rightarrow$ no.

$k = 5$: $5 \times 4 \equiv 20 \pmod{210} \rightarrow$ no; $k = 6$: $6 \times 5 \equiv 30 \pmod{210} \rightarrow$ no; $k = 7$: $7 \times 6 \equiv 42 \pmod{210} \rightarrow$ no; $k = 8$: $8 \times 7 \equiv 56 \pmod{210} \rightarrow$ no.

$k = 9$: $9 \times 8 \equiv 72 \pmod{210} \rightarrow$ no; $k = 10$: $10 \times 9 \equiv 90 \pmod{210} \rightarrow$ no; $k = 11$: $11 \times 10 \equiv 110 \pmod{210} \rightarrow$ no; $k = 12$: $12 \times 11 \equiv 132 \pmod{210} \rightarrow$ no.

$k = 13$: $13 \times 12 \equiv 156 \pmod{210} \rightarrow$ no; $k = 14$: $14 \times 13 \equiv 182 \pmod{210} \rightarrow$ no.

Only $k = 1$ is a solution in $[1, 14]$.

Therefore, the number of solutions in $[15, 210]$ is $16 - 1 = 15$.

Therefore, for $b = 211$, there are 15 beautiful numbers, which is more than 10.

...

Thus, the least integer $b \geq 2$ for which there are more than ten b-beautiful integers is $\boxed{211}$.