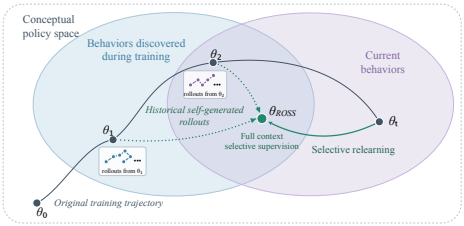
ROSS: RELEARNING FROM SELF-GENERATED ROLLOUTS THROUGH SELECTIVE SUPERVISION

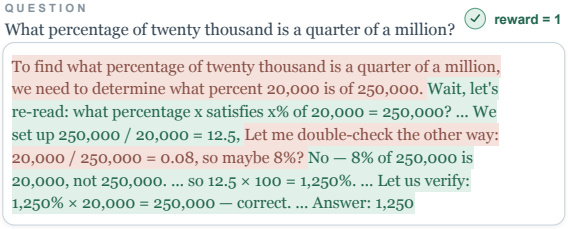
Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model post-training generates self-generated rollouts through reinforcement learning and on-policy distillation, yet this experience is often treated as stale once the policy advances. Historical rollouts can remain compatible with a later policy while preserving behaviors that the policy no longer expresses reliably. However, they may also contain mistakes, abandoned attempts, and redundant actions that should not be imitated, motivating finer-grained selective supervision. We introduce ***ROSS*** (Relearning from Self-Generated **Ro**llouts through **S**elective **S**upervision), which preserves the full historical trajectory as context while applying loss only to selected model-generated continuations. Across domain-specific reinforcement learning, multi-teacher on-policy distillation, and agentic reinforcement learning, ROSS consistently improves upstream checkpoints and outperforms baselines across mathematics, code generation, instruction following, and software engineering. On Qwen3.6-35B-A3B, ROSS improves the six-benchmark MOPD average from 58.40% to 62.20% and SWE-bench Verified from 64.20% to 68.40%. These results show that self-rollout training leaves behind reusable behavioral experience that can yield further gains through offline supervised fine-tuning (SFT), without additional policy rollouts.



(a) Historical experience preserves discovered behaviors.



(b) Successful rollouts contain both useful and uninformative segments.

Figure 1: **Relearning from self-generated historical rollouts.** **(a)** As the policy evolves, some useful behaviors may become underrepresented in its current rollout distribution. **(b)** Even when a rollout reaches a successful outcome, it may include redundant reasoning, or unnecessary detours, making full-trajectory replay suboptimal.

1 INTRODUCTION

Large language models (LLMs) (Guo et al., 2025; Lambert et al., 2024; Yuan et al., 2023; Yue et al., 2025) are increasingly trained from experience generated by the models themselves. Reinforcement learning (RL) (Yue et al., 2025; Guo et al., 2025; Hou et al., 2026; Zheng et al., 2025a) repeatedly samples rollouts from the current policy and updates it based on the resulting feedback, while on-policy distillation (OPD) (Agarwal et al., 2024; Gu et al., 2024; Lu & Lab, 2025; Yang et al., 2025; Xiao et al., 2026; Team et al., 2026) collects student trajectories and applies teacher supervision to the states they visit. Over training, these procedures produce not only a sequence of model checkpoints, but also a growing record of self-generated experience, including successful reasoning paths, alternative strategies, recoveries, and other behaviors discovered along the way. Yet as training

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.

advances, historical rollouts are often treated as stale and discarded. This raises a fundamental question: *can a later checkpoint continue to learn from what the model discovered earlier?*

This possibility stems from how the policy evolves during training. As the rollout distribution shifts with the policy, behaviors discovered at earlier stages need not remain reliably expressed at later checkpoints. As illustrated in Figure 1(a), some valid behaviors may become underrepresented even though they remain useful (Zhang et al., 2026; Wang et al., 2025; Yuan et al., 2023; Dong et al., 2023). Historical rollouts can therefore retain behaviors that the current policy no longer reliably recovers, providing complementary training signal beyond its current rollout distribution (Zheng et al., 2026; Xiong et al., 2024; Slinko et al., 2026). Yet this complementarity is not uniform across a trajectory: as Figure 1(b) illustrates, even successful rollouts may interleave useful reasoning with mistakes, redundant steps, and unnecessary detours. Outcome-level success alone therefore does not guarantee that every intermediate step provides desirable supervision (Wang et al., 2025; Lightman et al., 2024). The challenge is thus to identify not only which historical rollouts remain valuable, but also which parts of those rollouts are worth relearning.

We propose ***ROSS***, *Relearning from Self-Generated Rollouts through Selective Supervision*, to address these two levels of selection. ROSS identifies valuable historical rollouts and selectively supervises useful segments within successful trajectories while preserving the complete trajectory as context. In this way, the current policy can consolidate previously discovered behaviors without indiscriminately imitating entire trajectories or reinforcing mistakes and unnecessary steps. We evaluate ROSS across domain-specific RL, multi-teacher on-policy distillation (MOPD), and agentic RL, spanning reasoning, coding, instruction following, and software engineering. Across these settings, ROSS consistently improves the corresponding upstream checkpoints, including a gain from 58.40% to 62.20% on the six-benchmark MOPD average and from 64.20% to 68.40% on SWE-bench Verified. These results show that self-rollout training leaves behind not only a stronger policy, but also reusable experience that can be selectively consolidated by later checkpoints.

Our contributions are as follows:

- We establish historical self-generated rollouts as a reusable source of training experience, showing that they can retain useful behaviors under an evolving policy.
- We introduce ***ROSS***, which selectively consolidates historical experience by supervising informative segments while preserving full-trajectory context.
- We validate ROSS as an additional offline SFT stage after domain-specific RL, MOPD, and agentic RL, with further gains in math, coding, instruction following, and long-horizon agentic tasks.

2 Preliminary Analysis

To characterize when historical experience is worth revisiting, we consider two complementary dimensions: *compatibility* and *complementarity*. *Compatibility* measures how well a historical behavior aligns with the current policy, while *complementarity* measures whether it provides behavioral coverage that the current policy cannot reliably produce. As illustrated in Figure 2, their combination distinguishes useful historical experience from behavior that is either misaligned or already well covered by the current policy. We therefore examine compatibility through response-level predictability in Sec. 2.1 and complementarity through under-consolidated success on historically solved problems in Sec. 2.2.

2.1 Compatibility Analysis

Historical behaviors remain compatible with the current policy. We first ask whether historical rollouts remain compatible with the current policy. We select 100 mathematical problems and collect one response per problem from three sources: historical rollouts from earlier checkpoints, rollouts from the current policy, and an external teacher, GLM-5.2 (Zeng et al., 2026). We measure the token-level negative log-likelihood (NLL) of each response under the current policy, with lower NLL indicating greater consistency with its output distribution. As shown in Figure 3(a), historical rollouts have substantially lower NLL than external-teacher responses and remain close to current-policy rollouts. Thus, despite being generated by earlier checkpoints, historical behaviors can remain naturally compatible with the current policy, making them plausible sources of supervision (Slinko et al., 2026; Yuan et al., 2023).

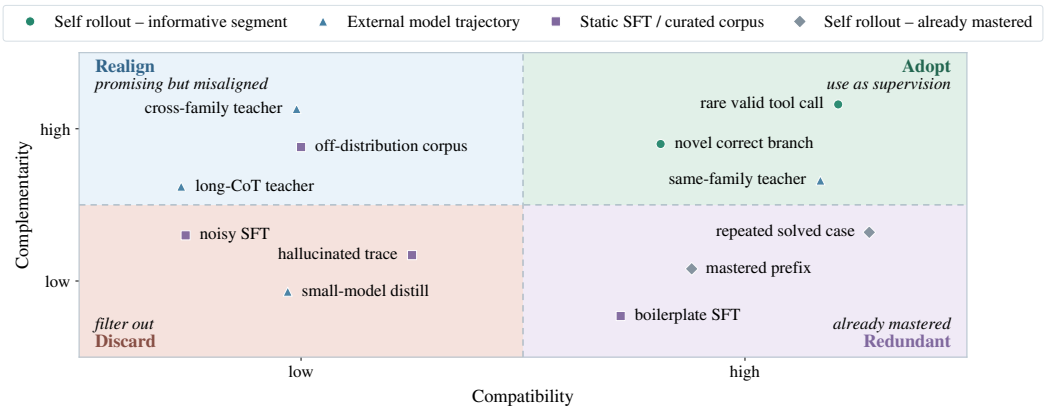


Figure 2: **When is historical experience worth revisiting?** Its value depends on *compatibility* with the current policy and *complementarity* to its behavioral coverage.

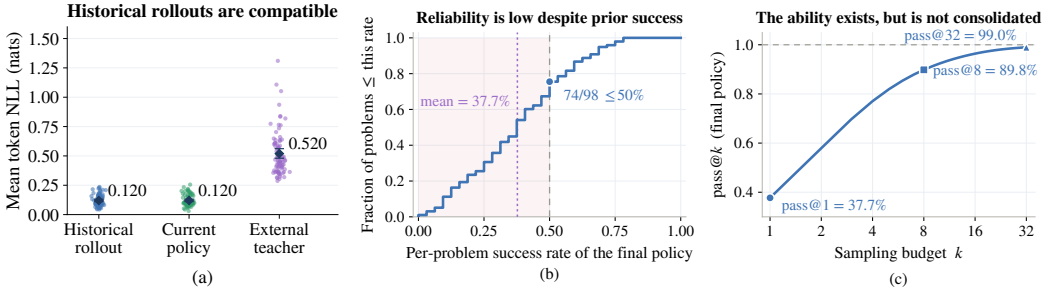


Figure 3: **Historical rollouts contain compatible but under-consolidated behaviors. (a)** Historical rollouts have response-level NLL comparable to the current policy and lower than external-teacher trajectories. **(b)** These historically successful behaviors remain unreliable under the current policy, with mean pass@1 $= 37.7\%$. **(c)** Repeated sampling raises success to pass@32 $= 99.0\%$, showing that the behaviors remain available but are not reliably expressed.

2.2 Complementarity Analysis

Historical experience preserves underrepresented behaviors. We next ask whether historically discovered behaviors remain reliably expressed by the current policy. We select the 100 most difficult training problems and collect 32 current-policy rollouts for each, retaining the 98 problems for which at least one historical rollout was successful. Shown in Figure 3(c), the current policy solves 97 of these problems, with an average pass@32 of 99.0%, showing that the corresponding behaviors remain within its behavioral support. However, the average pass@1 is only 37.7%, and 74 of 98 problems have a per-problem success rate of at most 50%. Historical rollouts therefore preserve behaviors that are compatible with the current policy but not yet reliably expressed. Combined with their high compatibility, this indicates a *near-on-policy* source of complementary experience for consolidating underrepresented behaviors.

Complementarity can also exist at the segment level. Useful signal need not span an entire rollout. Appendix C provides a concrete example in which a historical rollout reaches the correct count, 1007, by recovering from an intermediate $2^{10} = 1024$ counting error. The recovery segment identifies the overlooked constraint and corrects the count, whereas the preceding error is not a desirable imitation target. Thus, useful and undesirable reasoning can coexist within the same successful rollout, and trajectory-level correctness alone does not determine which segments should be learned. Complementarity is therefore not only a property of which rollout to revisit, but also of which segments within it provide useful signal.

Taken together, historical experience can remain *compatible* with the current policy while providing *complementary* behavioral coverage, sometimes only within specific segments. These observations motivate ROSS's two-level selective supervision.

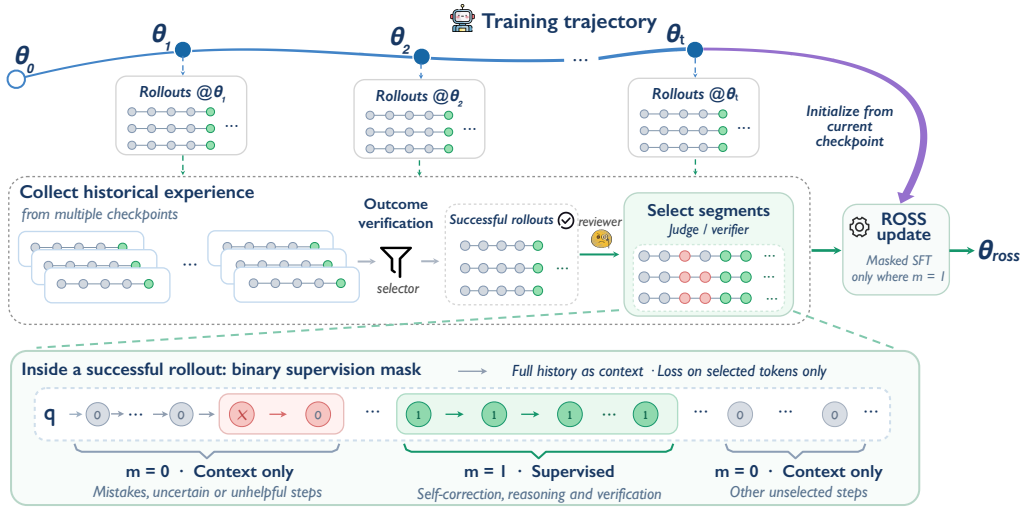


Figure 4: **Overview of ROSS.** Starting from the final checkpoint, ROSS filters saved rollouts by outcome verification, reviews the positive trajectories, and applies imitation loss only to selected model-generated tokens while preserving the full history as context.

3 METHOD

ROSS relearns from historical self-generated rollouts through selective supervision. Starting from the final checkpoint of the original training run, an outcome verifier first identifies positive trajectories; an LLM reviewer then either rejects each candidate or identifies the model-generated tokens that should receive imitation loss. Unselected tokens remain in the context but are excluded from the SFT objective. Figure 4 summarizes the procedure.

3.1 PROBLEM FORMULATION

Consider a self-rollout training procedure that produces a sequence of policy checkpoints $\{\pi_{\theta_k}\}_{k=0}^{T}$ and rollout batches $\{\mathcal{D}_k\}_{k=0}^{T-1}$, where $\tau \sim \pi_{\theta_k}$ for every $\tau \in \mathcal{D}_k$. The original training may use reinforcement learning, on-policy distillation, or agentic reinforcement learning; ROSS requires only the resulting checkpoints and saved rollouts. We define the historical experience pool as

$$\mathcal{H}_{<T} = \bigcup_{k<T} \mathcal{D}_k, \tag{1}$$

and initialize relearning from the final checkpoint π_{θ_T}. Although $\mathcal{H}_{<T}$ is no longer strictly on-policy for π_{θ_T}, its rollouts were generated by earlier checkpoints from the same training run and may contain behaviors that π_{θ_T} does not reliably express.

We represent each rollout as $\tau = (z_1, \ldots, z_L)$, which may encode a single-turn exchange or an agentic interaction history. Let $a_t \in \{0, 1\}$ indicate whether z_t is a policy-generated token eligible for imitation; prompts, environment observations, and other exogenous context have $a_t = 0$. ROSS applies a trajectory-level selector $g(\tau) \in \{0, 1\}$ and, for retained rollouts, a token-level supervision mask $m_t \in \{0, 1\}$. Staged LLM review followed by deterministic boundary and token-alignment checks produces a mask satisfying $m_t \leq a_t$.

3.2 SELECTIVE SUPERVISION OVER HISTORICAL ROLLOUTS

Trajectory-level selection. Let $v(\tau)$ denote the task outcome verifier, such as an exact-answer check, executable checker, unit-test suite, or environment success signal. It supplies trajectory-level feedback only. We retain

$$\mathcal{H}^{+} = \{\tau \in \mathcal{H}_{<T} : g(\tau) = 1\}, \qquad g(\tau) = \mathbf{1}[v(\tau) = 1]. \tag{2}$$

Within-trajectory selection. For each $\tau \in \mathcal{H}^+$, an LLM reviewer A examines the task, complete trajectory, and verifier or environment evidence. It returns an audit status q_τ and disjoint, ordered intervals $\mathcal{K}_\tau = \{[b_j, e_j)\}_{j=1}^{J_\tau}$ that contain locally correct, self-contained, and behaviorally useful model outputs. We convert these intervals into a token mask

$$m_t(\tau) = a_t \cdot \mathbf{1}\left[t \in \bigcup_{[b,e)\in\mathcal{K}_\tau} [b, e)\right]. \tag{3}$$

Only annotations with $q_\tau = \text{ACCEPT}$ enter training; the resulting subset is $\mathcal{H}_{\text{ROSS}} = \{\tau \in \mathcal{H}^+ : q_\tau = \text{ACCEPT}\}$. Proposed targets are independently audited and rejected when no clean target can be isolated without retaining a confirmed error or introducing reasoning absent from the rollout. Single-turn responses use positive-span extraction, whereas agentic histories use defect localization; both compile to the same token-level mask. Appendix D gives the full protocols and prompts.

Full context, selective targets. Setting $m_t = 0$ removes a token from the loss, not from the sequence. A selected token z_t is therefore predicted from the complete original prefix $z_{<t}$, including earlier mistakes, abandoned attempts, and environment feedback, preserving the state in which the continuation originally occurred.

3.3 ROSS Training Objective

Starting from θ_T, ROSS performs masked teacher-forcing on the selected historical rollouts. Its objective is

$$\mathcal{L}_{\text{ROSS}}(\theta) = -\frac{\sum_{\tau\in\mathcal{H}_{\text{ROSS}}} \sum_{t=1}^{|\tau|} m_t(\tau) \log \pi_\theta(z_t \mid z_{<t})}{\sum_{\tau\in\mathcal{H}_{\text{ROSS}}} \sum_{t=1}^{|\tau|} m_t(\tau)}. \tag{4}$$

Setting $m_t = a_t$ for every $\tau \in \mathcal{H}_{\text{ROSS}}$ recovers ROSS w/o mask, which uses the same post-annotation subset as ROSS but applies full-response supervision. Positive-Rollout SFT instead applies full-response supervision to all verifier-positive trajectories in $\mathcal{H}^+$. Appendix A states the full procedure as Algorithm 1.

4 Experiments

4.1 Experimental Setup

Models and training. We study two forms of self-generated experience with Qwen3.6-35B-A3B: single-turn rollouts and agentic trajectories. *For single-turn experiments*, we reuse historical rollouts from independent mathematics and code RL runs, as well as multi-teacher on-policy distillation (MOPD) spanning mathematics, code, and instruction following. ROSS starts from the final checkpoint and applies masked SFT to these rollouts. All single-turn experiments use a no-thinking configuration and report results after three SFT epochs, with optimization settings matched within each setting. Rollout review and mask annotation use GLM-5.2 in high-thinking mode as the LLM judge, which reviews the saved rollouts without generating replacement targets. *For agentic scenarios*, we train the model with agentic RL on OpenSWE tasks from daVinci-Env (Fu et al., 2026), using a Codex CLI agent (OpenAI, 2025) to interact with containerized repositories through Harbor (Harbor Framework Team, 2026). The task verifier provides terminal rewards. Both agentic training and relearning retain thinking traces and use a separate long-context schedule. Detailed configurations, rollout windows, and annotation protocols are provided in Appendices B and D.

Comparisons. We compare against four baselines: *(i) Base*, the open-source checkpoint used to initialize the original RL or MOPD training; *(ii) Upstream*, the final checkpoint of the original RL or MOPD run; *(iii) Continued RL/MOPD*, which follows the original training procedure for two additional epochs in Math and Code RL, or for 200 additional steps from the 200-step MOPD checkpoint; and *(iv) Positive-Rollout SFT*, which applies SFT to verifier-positive historical rollouts.

Evaluation. Single-turn evaluation covers mathematics (AIME 2025, AIME 2026, HMMT-November 2025), code generation (LCB Gen, OJBench), and instruction following (IFBench).

Table 1: **Main results with Qwen3.6-35B-A3B.** All scores are higher-is-better. (a) Math and Code use independent domain-specific RL runs; Math Avg. and Code Avg. are the corresponding domain means. (b) MOPD spans all three domains, and Avg. is the mean of its six benchmarks. All relearning methods start from the corresponding Upstream checkpoint; bold marks the best result among them, including ties.

(a) Domain-specific RL

	MATH RL				CODE RL		
Method	AIME 25	AIME 26	HMMT-Nov.	Math Avg.	LCB Gen	OJBench	Code Avg.
Base	71.25	76.56	70.62	72.81	57.24	22.20	39.72
Upstream	74.84	77.50	73.23	75.19	58.95	25.22	42.09
Continued RL	74.64	78.85	72.81	75.43	57.24	**27.37**	42.31
Positive-Rollout SFT	75.47	79.06	73.44	75.99	58.19	23.49	40.84
ROSS (ours)	**76.46**	**79.90**	**74.58**	**76.98**	**61.05**	**27.37**	**44.21**

(b) MOPD

	MATH			CODE		IF	
Method	AIME 25	AIME 26	HMMT-Nov.	LCB Gen	OJBench	IFBench	Avg.
Base	71.25	76.56	70.62	57.24	22.20	34.10	55.33
Upstream	72.81	77.60	71.25	57.43	26.72	44.58	58.40
Continued MOPD	73.39	78.23	72.60	57.71	27.59	45.45	59.16
Positive-Rollout SFT	74.06	77.66	69.17	56.86	26.94	41.49	57.70
ROSS (ours)	**77.76**	**80.42**	**76.77**	**60.76**	**29.74**	**47.76**	**62.20**

Agentic coding is evaluated on SWE-bench Verified (Jimenez et al., 2024) in thinking mode, with the evaluation scaffold, task environments, tool interfaces, and execution budgets fixed across policies. Details of the evaluation setup are given in Appendix B.

4.2 MAIN RESULTS

We evaluate ROSS on historical rollouts from domain-specific RL and MOPD, comparing it with the upstream checkpoints and baseline training strategies. Table 1 summarizes the results. Across both settings, ROSS consistently achieves the best performance among all relearning methods. After domain-specific RL, ROSS achieves the highest average on both Math and Code, improving from 75.19 to 76.98 and from 42.09 to 44.21, respectively. After MOPD, ROSS improves the six-benchmark average from 58.40 to 62.20, outperforming Continued MOPD and Positive-Rollout SFT. These results show that historical rollouts retain useful learning signal beyond their original training stage. While continued RL/MOPD further explores the policy's behavioral frontier, ROSS complements this process by consolidating useful behaviors accumulated throughout training that may remain unreliably expressed by the current checkpoint.

We further examine whether ROSS introduces degradation outside the domain targeted by domain-specific RL. Table 8 in Appendix F compares each ROSS-retrained checkpoint with its corresponding upstream RL checkpoint on mathematics, code generation, instruction following, and agentic tool use. ROSS preserves the gains of the domain-specific checkpoints while improving several off-domain capabilities. After Math RL, it raises the Avg. Code from 41.21 to 44.24 and IFBench from 33.30 to 34.60; after Code RL, it improves Avg. Math from 74.19 to 75.03. ROSS also improves multi-turn agentic tool use, raising the overall BFCL score from 44.12 to 45.62 and from 46.25 to 49.38, respectively. These cross-domain gains are consistent with the near-on-policy compatibility of historical rollouts demonstrated in Section 2.1, suggesting that relearning from them can consolidate useful experience without sacrificing capabilities acquired by the current policy.

Table 3: **Effect of trajectory filtering and token-level masking.** Subscripts show percentage-point changes from Upstream; green/red denote gains/losses and bold marks the best score. ROSS w/o mask and ROSS use identical examples and differ only by token-level masking.

(a) Domain-specific RL

	MATH RL			CODE RL	
Method	AIME 25	AIME 26	HMMT-Nov.	LCB Gen	OJBench
Upstream	74.84	77.50	73.23	58.95	25.22
Positive-Rollout SFT	75.47$_{+0.63}$	79.06$_{+1.56}$	73.44$_{+0.21}$	58.19$_{-0.76}$	23.49$_{-1.73}$
ROSS w/o mask	**76.67**$_{+1.83}$	79.06$_{+1.56}$	73.23$_{+0.00}$	59.14$_{+0.19}$	26.94$_{+1.72}$
ROSS (ours)	76.46$_{+1.62}$	**79.90**$_{+2.40}$	**74.58**$_{+1.35}$	**61.05**$_{+2.10}$	**27.37**$_{+2.15}$

(b) Multi-teacher on-policy distillation

Method	AIME 25	AIME 26	HMMT-Nov.	LCB Gen	OJBench	IFBench	Avg.
Upstream	72.81	77.60	71.25	57.43	26.72	44.58	58.40
Positive-Rollout SFT	74.06$_{+1.25}$	77.66$_{+0.06}$	69.17$_{-2.08}$	56.86$_{-0.57}$	26.94$_{+0.22}$	41.49$_{-3.09}$	57.70$_{-0.70}$
ROSS w/o mask	73.28$_{+0.47}$	77.55$_{-0.05}$	71.67$_{+0.42}$	58.38$_{+0.95}$	27.59$_{+0.87}$	42.45$_{-2.13}$	58.49$_{+0.09}$
ROSS (ours)	**77.76**$_{+4.95}$	**80.42**$_{+2.82}$	**76.77**$_{+5.52}$	**60.76**$_{+3.33}$	**29.74**$_{+3.02}$	**47.76**$_{+3.18}$	**62.20**$_{+3.80}$

4.3 RELEARNING FROM AGENTIC TRAJECTORIES

The results above establish the value of historical experience in single-turn rollouts. We next examine whether this benefit extends to long-horizon agent–environment interaction. We reuse successful OpenSWE trajectories from the agentic RL run and evaluate on SWE-bench Verified, with the training protocol described in Appendix B.4. As shown in Table 2, ROSS improves the resolved-issue rate from 64.20 to 68.40, yielding a 4.20-point gain over the Upstream RL checkpoint. The gain shows that ROSS generalizes beyond single-turn rollouts to long-horizon agentic trajectories, where useful behaviors are embedded in extended interaction histories.

Table 2: **Agentic trajectory reuse.** Δ is relative to Upstream.

Method	SWE-bench Verified ↑ (Δ)
Base	60.80
Upstream	64.20
Positive-Rollout SFT	65.20 (+1.00)
ROSS (ours)	**68.40** (+4.20)

4.4 WHAT DRIVES RELEARNING FROM HISTORICAL ROLLOUTS?

Having shown that historical rollouts remain useful across both single-turn and agentic settings, we analyze two factors behind these gains: whether token-level masking adds value beyond trajectory-level filtering, and whether relearning depends on inheriting the Upstream parameter updates.

The Role of Trajectory Filtering and Selective Supervision We disentangle trajectory filtering from token-level selective supervision. Table 3 compares Positive-Rollout SFT on all verifier-positive rollouts, ROSS w/o mask on the rollouts retained by ROSS with all eligible tokens supervised, and full ROSS. We find that: *(i)* Trajectory filtering already provides a clear benefit. ROSS w/o mask reverses the degradation of Positive-Rollout SFT on Code, improving over Upstream by 0.19 points on LCB Gen and 1.72 points on OJBench, while raising the MOPD average from 57.70 to 58.49, slightly above Upstream checkpoint. *(ii)* Selective supervision accounts for the remaining gains. With the replay set fixed, ROSS further improves over ROSS w/o mask by 3.71 points on MOPD, 1.91 on LCB Gen, 0.43 on OJBench, and 0.66 on Math, showing that selectively supervising informative segments is more effective than imitating all eligible tokens.

Relearning Across Initializations The previous comparison shows that ROSS's token-level mask adds value beyond trajectory-level filtering. We further examine whether the value of historical rollouts depends on starting from the Upstream checkpoint, or whether the same recorded experience can transfer useful behaviors to a different initialization. To test this, we apply identical historical trajectories, supervision masks, and SFT configurations starting from either Base, before the original

post-training run, or the corresponding Upstream checkpoint. As shown in Table 4, Base-initialized ROSS achieves performance comparable to Upstream-initialized ROSS across all evaluated settings, despite inheriting none of the original RL/MOPD parameter updates. This result suggests that historical rollouts are useful not only for further refining the policy that generated them; more importantly, they preserve behaviors discovered during training that can transfer across initializations and be consolidated into a different checkpoint. The rollout history therefore provides reusable learning value across initializations, beyond what is carried forward through the Upstream parameters alone.

Table 4: **ROSS reaches similar scores from different initializations.** Within each setting, the two runs use identical historical trajectories, masks, and SFT configurations. Math is Avg. Math, Code is LCB Gen, and IF is IFBench.

	Math RL	Code RL	MOPD		
	Math ↑	Code ↑	Math ↑	Code ↑	IF ↑
Base	72.81	57.24	72.81	57.24	34.10
Upstream	75.19	58.95	73.89	57.43	44.58
After ROSS on identical trajectories and masks					
Initialized from Base	**77.09**	**61.33**	**77.95**	**61.24**	**47.18**
Initialized from Upstream	**76.98**$_{-0.11}$	**61.05**$_{-0.28}$	**78.32**$_{+0.37}$	**60.76**$_{-0.48}$	**47.76**$_{+0.58}$

4.5 Characterizing Excluded Supervision

The controlled comparisons in Section 4.4 show that token-level masking contributes beyond trajectory-level filtering. We therefore examine what ROSS removes from the training loss and whether such excluded supervision diminishes as RL progresses. We analyze both the semantic content of masked regions and their prevalence over the course of training.

The excluded content differs across domains. We use GLM-5.2 to classify masked excerpts from 1,185 sampled trajectories, assigning one primary category to each trajectory. In Math, mistakes followed by a visible correction dominate both early and late training samples (72.7% and 68.3%). In Code, redundant exploration is the largest category and becomes more prevalent from early to late training (67.9% to 76.6%). These patterns highlight two common forms of undesirable supervision within globally successful rollouts: Math trajectories often recover from explicit intermediate errors, whereas Code trajectories more often reach successful outcomes through exploratory detours. Appendix E provides the full category distribution, definitions, and sampling protocol.

Excluded supervision persists as RL progresses. We next examine whether the need for masking naturally diminishes as the policy evolves. Over the first 100 rollout steps of the Math and Code RL runs, we measure the token-weighted *target exclusion rate* (TER) among retained, verifier-positive responses $\mathcal{D}_s$ at step s, defined as $\mathrm{TER}(s) = \frac{\sum_{\tau \in \mathcal{D}_s} \sum_{t=1}^{L_\tau} (1 - m_{\tau,t})}{\sum_{\tau \in \mathcal{D}_s} L_\tau}$, where L_τ counts response-content tokens and $m_{\tau,t}$ indicates whether a token overlaps a selected span; prompt and chat-template tokens are excluded. Policy entropy decreases throughout both RL runs, but the amount of excluded supervision does not vanish with training. From the early to late window, TER decreases only modestly from 11.60% to 9.27% in Math, while increasing from 22.65% to 28.07% in Code (Figure 5). Thus, increasing policy certainty does not make successful rollouts uniformly suitable for imitation. For outcome-level RL objectives such as GRPO, positive trajectory-level feedback can still reinforce local mistakes, abandoned reasoning, or redundant exploration, motivating retrospective masking when historical rollouts are reused.

Appendix G provides complementary evidence from a separate 4B Math RL run, where Upstream develops a repetitive-answer mode. ROSS suppresses this behavior while improving Avg. Math from 52.05% to 63.65%, showing that selective relearning can both consolidate useful historical behaviors and avoid reinforcing undesirable modes.

5 Related Work

Learning from Self-Generated Experience Recent work studies improving language models through self-generated supervision (Zhang et al., 2024; Yuan et al., 2023; Zelikman et al., 2022;

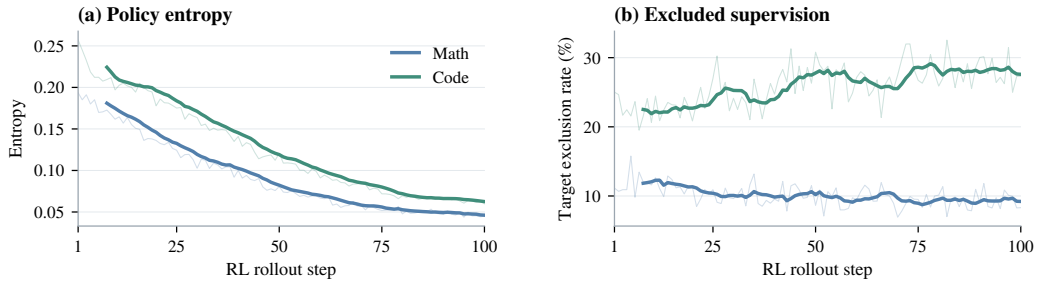


Figure 5: **Lower entropy does not imply less excluded supervision.** Math (blue) and Code (green) over 100 RL rollout steps: (a) logged entropy; (b) TER among retained trajectories. Thin curves show per-step values; thick curves show eight-step moving averages.

Dong et al., 2023). Rejection Sampling Fine-Tuning (RFT) (Yuan et al., 2023) generates candidate solutions and fine-tunes on correct reasoning paths, while RAFT (Dong et al., 2023) ranks samples with a reward model and fine-tunes on high-quality responses. Iterative self-training extends this paradigm: ReST-MCTS* (Zhang et al., 2024) combines process-reward-guided tree search with iterative self-training, SCoRe learns self-correction from multi-turn self-generated interactions through online reinforcement learning (Kumar et al., 2025), and RLoop reuses successful trajectories from intermediate policies to initialize later RL stages (Zhiyuan et al., 2025). RIFT reweights positive and negative self-generated trajectories with scalar rewards rather than discarding negative samples (Liu et al., 2026). Despite these advances, existing methods largely treat a rollout as an indivisible supervision unit, selecting, refining, or reweighting trajectories by overall outcomes or rewards. ROSS instead asks whether behaviors within a successful self-generated rollout should be learned uniformly, motivating finer-grained selective supervision.

Selecting Useful Supervision from Rollouts Beyond selecting successful trajectories as a whole, recent work explores how to identify which parts of a rollout are useful for learning. Process supervision provides step-level feedback for reasoning (Lightman et al., 2024), while ProcessBench (Zheng et al., 2025b) and ThinkPRM (Khalifa et al., 2025) study identifying and verifying erroneous reasoning steps. For interactive agents, IPR (Xiong et al., 2024) performs step-level process refinement by comparing generated actions with expert trajectories. STeP (Chen et al., 2025) and Step Rejection Fine-Tuning (SRFT) (Slinko et al., 2026) further use loss masking to avoid imitating erroneous or undesirable steps. Most closely related, SWE-Prime (Zheng et al., 2026) explicitly selects useful segments from successful software-engineering trajectories based on their contribution, learnability, and risk, while keeping the full trajectory as context during SFT (Zheng et al., 2026). These works demonstrate the value of selective supervision within rollouts. ROSS focuses on a complementary setting: successful historical rollouts generated by intermediate policies during upstream training, where the central question is not only whether a rollout is worth keeping, but which behaviors within it are worth reinforcing. ROSS therefore applies supervision selectively to useful portions of successful self-generated rollouts, avoiding uniform imitation of the full trajectory.

6 Conclusion

In this work, we show that historical self-generated rollouts from LLM post-training are not merely transient artifacts to be discarded, but a reusable source of behavioral experience. To avoid imitating intermediate mistakes and redundant exploration present in successful rollouts, we propose **ROSS**, which preserves the complete historical trajectory as context while supervising only informative, verified segments. Across domain-specific RL, MOPD, and agentic RL, ROSS consistently outperforms both upstream checkpoints and competing relearning baselines, while preserving or even enhancing out-of-domain capabilities. These results highlight a broader post-training paradigm: systematically consolidating the reusable behavioral footprint accumulated during policy evolution.

AI USE STATEMENT

Generative AI was used in the ROSS experimental pipeline. Specifically, GLM-5.2 served as the reviewer in our method to annotate saved rollouts during the experiments: it proposed and audited selective-supervision masks and categorized excluded content for qualitative analysis. Its annotations were checked against the original trajectories, and it did not generate replacement training responses. Generative AI tools were also used for light editing of the manuscript's grammar, phrasing, and formatting, and for retrieval and discovery (e.g., finding related work); relevant details are described in the paper. The authors take full responsibility for the final content of this paper.

REPRODUCIBILITY STATEMENT

The anonymized implementation is available at `https://anonymous.4open.science/r/ROSS-302A/`. It includes the single-turn and agentic annotation pipelines, complete reviewer prompts, deterministic token-mask compilation and data-preparation utilities, and a reference masked-SFT launcher. Section 3 and Appendix A specify the ROSS procedure; Section 4 and Appendix B report the experimental setup, training configurations, dataset sizes, and benchmark-specific evaluation protocols; Appendix D details the annotation rules and mask construction.

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 *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Yihan Chen, Benfeng Xu, Xiaorui Wang, Yongdong Zhang, and Zhendong Mao. Training llm-based agents with synthetic self-reflected trajectories and partial masking. *arXiv preprint arXiv:2505.20023*, 2025.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023.

Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. daVinci-Env: Open SWE environment synthesis at scale. *arXiv preprint arXiv:2603.13023*, 2026. URL `https://arxiv.org/abs/2603.13023`.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

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.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL `https://www.harborframework.com/`.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=VTF8yNQM66`.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think. *arXiv preprint arXiv:2504.16828*, 2025.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CjwERcAU7w.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Junlong Li, Daya Guo, Dejian Yang, Runxin Xu, Yu Wu, and Junxian He. CodeI/O: Condensing reasoning patterns via code input-output prediction. *arXiv preprint arXiv:2502.07316*, 2025. URL https://arxiv.org/abs/2502.07316.

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

Zehua Liu, Shuqi Liu, Tao Zhong, and Mingxuan Yuan. Rift: Repurposing negative samples via reward-informed fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14399–14415, 2026.

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

OpenAI. Codex CLI. GitHub repository, 2025. URL https://github.com/openai/codex.

Igor Slinko, Ilia Zavidnyi, Egor Bogomolov, and Yaroslav Zharov. Step rejection fine-tuning: A practical distillation recipe. *arXiv preprint arXiv:2605.10674*, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*, 2025.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1556–1572, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhuolin Yang, Zihan Liu, Yang Chen, Wenliang Dai, Boxin Wang, Sheng-Chieh Lin, Chankyu Lee, Yangyi Chen, Dongfu Jiang, Jiafan He, et al. Nemotron-cascade 2: Post-training LLMs with cascade RL and multi-domain on-policy distillation. *arXiv preprint arXiv:2603.19220*, 2026. URL https://arxiv.org/abs/2603.19220.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. URL https://arxiv.org/abs/2503.14476.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *2nd AI for Math Workshop @ ICML 2025*, 2025. URL https://openreview.net/forum?id=upehLVgq1b.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and RL on reasoning language models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=TBaUfO9znF.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025a.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024, 2025b.

Dewu Zheng, Ruizhe Ye, Yanlin Wang, Yang Ye, Hongyu Zhang, Ensheng Shi, Xilin Liu, Yuchi Ma, Jianxing Yu, and Zibin Zheng. Swe-prime: Fewer trajectories, better performance. *arXiv preprint arXiv:2608.27449*, 2026.

Zeng Zhiyuan, Jiashuo Liu, Zhangyue Yin, Ge Zhang, Wenhao Huang, and Xipeng Qiu. Rloop: An self-improving framework for reinforcement learning with iterative policy initialization. *arXiv preprint arXiv:2511.04285*, 2025.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An LLM post-training framework for RL scaling. GitHub repository, 2025. URL https://github.com/THUDM/slime.

A ROSS PROCEDURE

Algorithm 1 summarizes the relearning stage described in Section 3. The task verifier defines the trajectory-level selector g, the LLM reviewer A proposes and audits within-trajectory imitation targets, and all source-boundary resolution and mask compilation steps are deterministic.

Algorithm 1 ROSS: Relearning from Self-Generated Rollouts through Selective Supervision

Require: Checkpoints $\{\pi_{\theta_k}\}_{k=0}^{T}$, saved rollouts $\{\mathcal{D}_k\}_{k<T}$, outcome selector g, reviewer A

$\mathcal{S} \leftarrow \varnothing$
for $\tau \in \bigcup_{k<T} \mathcal{D}_k$ **do**
  if $g(\tau) = 0$ **then**
    continue
  end if
  $\mathcal{K}_\tau, q_\tau \leftarrow A(\tau)$ ▷ propose and audit imitation targets
  if $q_\tau = \text{ACCEPT}$ and $\mathcal{K}_\tau \neq \varnothing$ **then**
    $m(\tau) \leftarrow \text{COMPILEMASK}(\tau, \mathcal{K}_\tau)$
    if $\text{VALIDATE}(\tau, m(\tau))$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\tau, m(\tau))\}$
    end if
  end if
end for
$\theta \leftarrow \theta_T$
Update θ on $\mathcal{S}$ using Eq. 4
return $\pi_{\theta_{\text{ROSS}}}$

B TRAINING AND EVALUATION DETAILS

B.1 HISTORICAL DATA AND CONTROLLED COMPARISONS

The main experiments use Qwen3.6-35B-A3B, a mixture-of-experts model with approximately 3B active parameters. The single-turn upstream training pools contain 3,000 Math prompts from DAPO-Math-17K (Yu et al., 2025), 12,000 Code prompts from CodeI/O (Li et al., 2025), and 10,000 instruction-following prompts from the data released with Nemotron-Cascade 2 (Yang et al., 2026). Domain-specific RL uses the corresponding Math or Code pool, whereas MOPD spans all three domains. ROSS reuses the saved model responses rather than obtaining replacement solutions from the annotation model. Table 5 specifies the rollout windows and dataset sizes. The additional Qwen3.5-4B case is described in Appendix G.

Table 5: **Historical data used for relearning.** ROSS w/o mask and ROSS share the retained examples; only the token-level loss mask differs. Positive-Rollout SFT uses all verifier-positive examples in the corresponding window.

Setting	Initialization / rollout window	Positive-Rollout SFT examples	ROSS examples
Math RL	Iteration 47 / steps 1–48	35,700	27,848
Code RL	Iteration 187 / steps 1–187	152,454	144,702
MOPD	Step 199 / first 200 steps	42,558	31,228

The comparisons in Section 4.4 follow three sequential contrasts: replay (Positive-Rollout SFT minus Upstream), filtering (ROSS w/o mask minus Positive-Rollout SFT), and masking (ROSS minus ROSS w/o mask). The masking contrast fixes the examples and input context; filtering changes both dataset composition and size. Dataset and mask sizes also determine the number of supervised tokens per epoch.

B.2 Optimization and Infrastructure

We use the slime training stack (Zhu et al., 2025). Upstream Math RL runs asynchronous rollout generation and policy updates with GRPO advantages, no group-standard-deviation normalization, a PPO-clipped surrogate, and IcePop mismatch correction. Its recorded configuration uses 128 prompts per rollout step, eight responses per prompt, an update batch of 256, temperature 1.0, and a 10,240-token response limit. Adam uses a constant learning rate of 10^{-6}, $(\beta_1, \beta_2) = (0.9, 0.98)$, weight decay 0.1, and policy clipping at 0.2. The launch allocates four eight-GPU training nodes and two eight-GPU rollout nodes. These upstream settings are distinct from the subsequent SFT configuration.

Table 6: **35B relearning configurations.** Single-turn Math, Code, and MOPD rollouts use no-thinking responses; agentic rollouts retain thinking traces and use a longer context and separate schedule. Within each setting, ROSS and its matched full-supervision control share the listed configuration.

Parameter	Single-turn rollouts (no-thinking)	Agentic trajectories (thinking)
Optimizer	Adam, $\beta_1 = 0.9$, $\beta_2 = 0.98$	
Peak / minimum learning rate	4×10^{-6} / 2×10^{-7}	
Schedule / warmup fraction	Cosine / 0.05	
Weight decay	0.1	
Global batch size	128	
Maximum sequence length (tokens)	32,768	65,536
Configured epochs	3	8
Tensor / pipeline / context parallelism	2 / 2 / 1	2 / 1 / 8
Expert / expert-tensor parallelism	2 / 1	8 / 1

Relearning preserves the saved response text and applies the accepted supervision mask to eligible assistant positions. The annotation model supplies selection decisions, not new target responses; Appendix D describes proposal, audit, and mask compilation. For the single-turn rollout experiments, we report epoch-three endpoints consistently rather than selecting each method's best test score. The agentic configuration is described separately below.

B.3 Single-Turn Rollout Evaluation Protocol

The Math, Code, and MOPD evaluations use the no-thinking inference configuration and a 16,384-token output limit. AIME 2025 and AIME 2026 average accuracy over 64 generations per problem; HMMT-November 2025 averages over 32. Avg. Math is their unweighted mean. LCB Gen is the code-generation pass@1 score averaged over six runs, excluding execution and output-prediction subtasks. OJBench reports the overall C++/Python aggregate. IFBench measures instruction following. The additional BFCL multi-turn evaluation reports overall and subset scores in Appendix F. Scores are percentages and differences are percentage points. The 32K diagnostic for the 4B model is an explicitly separate output-budget control.

B.4 Agentic Training and Evaluation Protocol

Upstream agentic RL. OpenSWE and SWE-bench Verified play different roles in our protocol: OpenSWE supplies the upstream RL tasks, whereas SWE-bench Verified is reserved for evaluation. Before training, we remove tasks with exposed `.git` histories or other artifacts that could enable reward hacking, together with low-quality examples, yielding 4,048 training tasks. We train Qwen3.6-35B-A3B on this filtered OpenSWE set from the daVinci-Env project (Fu et al., 2026). A Codex CLI agent (OpenAI, 2025) executes repository-level actions in the containerized environments provided by Harbor (Harbor Framework Team, 2026); the task verifier converts the final repository state into a terminal reward. Rollout generation is thinking-enabled, and each saved trajectory interleaves model-generated assistant content with tool results and other environment observations.

Table 7: **A segment-level complementarity case.** The earlier historical rollout recovers from a shared overcounting error, whereas the later rollout retains it.

Setting. The task asks how many distinct tuples $(f(1), \ldots, f(2014))$ can be induced by $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(1) = 1$, monotonicity, and $f(2a) = f(a) + 1$ for every $a \in \mathbb{N}$.

Earlier historical rollout **correct: 1007**	**Later rollout** **incorrect: 1024**
Shared construction. Let T_k count entries taking the lower value $k+1$ in $[2^k, 2^{k+1}-1]$. Both rollouts derive $T_0 = 1$ and $T_{k+1} \in \{2T_k - 1, 2T_k\}$, then count the admissible paths.	
Same overcounting error. "So all 2^{10} sequences are valid. Thus, there are 1024 possible tuples. ..."	**Same overcounting error.** "Since $2^{10} = 1024 \leq 2014 < 2048 = 2^{11}$, the number of choices is ... 1024."
Recovery. "Let's double check if T_{10} being larger than 990 creates any ambiguity or if we missed a constraint. ..."	**No corresponding recovery.** The rollout never uses the observation limit $n \leq 2014$ to test whether distinct threshold sequences induce the same tuple.
Corrected count. Each $T_9 \in \{1, \ldots, 512\}$ uniquely fixes the prefix. For $T_9 \leq 495$, its two admissible T_{10} values yield distinct observed suffixes; for the 17 values $T_9 \geq 496$, they yield the same suffix. Hence $495 \times 2 + 17 \times 1 = 1007$.	**Terminal claim.** The rollout retains the uncorrected count and returns 1024.

Interpretation. The green recovery is complementary behavior. The earlier rollout is correct overall, but its red overcount is an undesirable imitation target. ROSS preserves the full prefix as context while supervising only the selected continuation.

Excerpts are abridged; ellipses mark omissions. Colors indicate analytical roles, not additional automatic annotations.

Trajectory reuse. The task verifier first identifies successful interaction histories. Positive-Rollout SFT treats the eligible assistant tokens in these histories as full-trajectory SFT targets. ROSS instead applies the agentic annotation procedure in Appendix D.3: the complete history remains visible to the model, while assistant spans associated with identified errors are excluded from the loss. User messages, tool results, and environment observations serve only as context in both methods. Neither method replaces the recorded model actions with responses generated by the annotation model.

Long-context trajectory SFT. Agentic relearning remains thinking-enabled and uses a 65,536-token sequence limit to accommodate multi-turn interaction histories, compared with 32,768 tokens for single-turn rollouts. Both ROSS and Positive-Rollout SFT start from upstream RL iteration 29 and use trajectory datasets filtered to the longer limit. They are configured for eight epochs on 64 GPUs across eight nodes, with the parallelism settings in Table 6. Thus, "full-trajectory" supervision refers to eligible assistant targets across the retained interaction, rather than to a loss on every token in the sequence.

SWE-bench Verified evaluation. We evaluate Base, Upstream, Positive-Rollout SFT, and ROSS on SWE-bench Verified (Jimenez et al., 2024). Evaluation also enables thinking and measures the percentage of repository issues resolved under the benchmark verifier. Across all four policies, we hold fixed the Codex agent scaffold, Harbor task environments, tool interfaces, and execution budgets, so only the policy checkpoint changes. Section 4.3 reports the resulting comparison.

C A Segment-Level Complementarity Case

The aggregate analysis in Section 2.2 shows that historically discovered solutions can remain available but unreliable under the later policy. Table 7 localizes this complementarity within a trajectory. The two rollouts solve the same functional-equation problem and use the same threshold construction, but only the earlier rollout contains the recovery needed to obtain the correct count.

Longitudinal context. The displayed late failure is representative of a broader pattern rather than an isolated unsuccessful sample. We draw eight rollouts for this problem at each of training steps 2, 42, 67, 86, and 109; the numbers of correct responses are respectively 1, 3, 2, 0, and 0. The solution

is therefore expressed only sparsely earlier in training and is absent from all 16 sampled rollouts at the final two observed steps. Across the five steps, the problem's success rate is $6/40 = 15.0\%$, compared with 74.3% averaged over the full training set, and it remains within the hardest 1–11% of problems at every sampled step. The late rollout in Table 7 illustrates what is missing after this longitudinal change; the change itself is established by the repeated samples.

D SELECTIVE-SUPERVISION ANNOTATION PROTOCOLS

This appendix specifies the staged reviewer A in Eq. 3. Outcome verification first defines the candidate pool $\mathcal{H}^+$; annotation then determines whether a candidate yields a valid supervision mask and enters $\mathcal{H}_{\mathrm{ROSS}}$. The two data regimes use complementary procedures. For single-turn Math and Code rollouts, the reviewer extracts a self-contained correct solution from the response; verifier-positive instruction-following responses retain full-response supervision. For agentic trajectories, the reviewer first diagnoses errors at the assistant-step level and then localizes the source text responsible for each error. In both cases, deterministic code resolves the returned source references and maps them to the stored training tokens. The prompt boxes below present the core decision rules and output requirements; service wrappers and task-specific demonstrations are omitted. The complete annotation pipelines and full prompts are available in the anonymous code repository.

D.1 SHARED ANNOTATION CONTRACT

Every annotation is grounded in exact text from the saved rollout. The reviewer neither rewrites a response nor supplies missing reasoning or actions. Only model-generated assistant tokens with $a_t = 1$ can receive imitation loss; user messages, verifier evidence, tool observations, and protocol wrappers remain context. Setting $m_t = 0$ changes the loss mask while leaving the original sequence intact.

The two procedures differ in which regions they return. Single-turn annotation identifies text to keep, whereas agentic annotation identifies text to exclude. The compiler resolves these regions against the original source, maps them onto the stored tokenization, and verifies their order, ownership, overlap, and alignment. It never constructs the training example by concatenating selected text or retokenizing an edited response. A candidate is withheld when its semantic decision or source boundary cannot be resolved reliably.

D.2 SINGLE-TURN ROLLOUTS: POSITIVE-SPAN SELECTION

This procedure is used for single-turn Math and Code rollouts. Each candidate contains the problem, complete model response, verifier evidence, and a reference answer when available. Instruction-following rollouts use the outcome verifier directly because responses in this training pool are typically short answers with little extended reasoning, leaving limited benefit from span-level annotation or masking. A positive response therefore keeps all eligible assistant tokens, whereas a failed response is discarded.

Proposal, audit, and repair. The first LLM judge chooses among three outcomes: supervise the complete response, retain an ordered sequence of clean spans, or reject the candidate. Partial selections use verbatim start and end anchors. Exact matching converts them into half-open character intervals; a missing, repeated, ambiguous, or out-of-order anchor invalidates the proposal. The selected text must include the complete final answer and, when its spans are read in order, retain the reasoning needed to support that answer. A separate judge audits this exact selected text rather than the proposal's summary. If the audit finds a separable error, a revision call updates the anchors and repeats the source and completeness checks. The candidate is dropped when a clean solution cannot be recovered solely from text already present in the rollout.

System Prompt: Single-Turn Rollout Span Selection

Role. You are a strict segment-level credit-assignment annotator. Convert one verified model rollout into masked SFT supervision by selecting only contiguous regions of the *original response* that are worth imitating.

System Prompt: Single-Turn Rollout Span Selection (continued)

Keep. Retain correct problem setup used by the final solution, locally valid intermediate reasoning, the complete self-contained derivation, valid verification, and the explicit final answer. A genuine proof by contradiction is valid when its temporary assumption is stated and correctly refuted.

Exclude. Do not supervise false assumptions, arithmetic or logical errors, abandoned attempts, discarded code, hard-coded shortcuts, or correction-trigger text that is inseparable from a wrong branch. A correct final answer or verifier pass does not validate earlier claims. Recompute incompatible counts, signs, states, and quantified claims locally.

Do not penalize correct checking. Redundancy alone is not an error. Keep a second calculation or verification when it is correct. Expressions such as "let us verify" are not masking signals by themselves; correctness of the resulting claim is decisive.

Self-containment. Read the selected spans in order as a standalone solution. Every variable, definition, case split, and dependency used later must be introduced within selected text. If the only required setup is inseparable from an error, declare that no clean solution is extractable rather than inventing a bridge.

Boundary rule. Copy unique start and end anchors verbatim from the response. Multiple spans are allowed. Preserve complete sentences and equations, and split spans around wrong text instead of describing an internal hole.

Output. Return structured fields indicating whether the whole response is clean, whether a clean final solution exists, the ordered keep spans and their reasons, a summary of excluded regions, and risk tags. If the outcome verifier fails or no complete correct derivation can be extracted, return no keep spans.

System Prompt: Independent Audit of Selected Single-Turn Spans

Audit the *actual text selected by the current proposal*, not the proposal's explanation. Treat the concatenated spans as the candidate training target.

Check that (i) every retained statement is locally correct; (ii) all definitions and dependencies required by later spans are present; (iii) the selected text contains the derivation rather than only setup and answer; (iv) the final answer is complete and consistent with the verifier evidence; and (v) no contradicted assertion, invalid implication, abandoned branch, or answer-format corruption remains.

Scan the entire selection before assigning a verdict. For every missed error, quote the exact retained source text and explain why it is wrong. Distinguish a correct verification from a correction of a genuine mistake. Return ACCEPT only when the selected text is a coherent imitation target; otherwise return REVISE with grounded error evidence or DROP when repair would require inventing content.

Revision Instruction: Single-Turn Span Repair

Revise the keep spans using the audit evidence. Remove every confirmed error while preserving independently correct setup, reasoning, verification, and the complete final answer. Use only unique verbatim source anchors and do not rewrite the response.

After proposing new spans, reread their concatenation for derivation completeness and dangling references. Do not expand a boundary across the error being removed. If no complete correct solution can be extracted from the original response, drop the example instead of manufacturing missing reasoning.

Deterministic acceptance checks. An accepted annotation must resolve uniquely against the stored response, retain the complete answer, contain at least one eligible assistant token, and exclude every error confirmed by the audit. The dataset builder projects the accepted character intervals onto overlapping stored tokens and intersects them with the original eligibility mask $a_{1:L}$.

D.3 Agentic Trajectories: Semantic Review and Defect Localization

Agentic trajectories require a different review procedure because an unsuccessful tool call does not by itself determine which generated tokens, if any, should be excluded. A failure may expose a defective action, an unmet environment prerequisite, or simply the negative result of a reasonable probe. The reviewer therefore separates semantic diagnosis from source-boundary localization.

Step construction and semantic review. Without changing token identity, the parser groups the saved history into assistant steps, tool calls, and observations. Long histories are reviewed in

bounded groups of assistant steps, while the complete saved history remains available as evidence. The semantic judge assesses only its assigned steps. It checks narrative claims separately from tool calls and evaluates each call for interface compliance, whether its arguments implement the stated operation, whether required environment state is established, and what the observation shows actually executed. An incorrect sequence already present in the response is recorded as an error. A missing prerequisite or action is recorded as an omission because it has no source token to exclude. Later success does not erase an earlier confirmed error, while a reasonable exploratory action remains eligible even when it returns negative evidence.

System Prompt: Agentic Semantic Error Discovery

Role and scope. Annotate selective supervision over original generated tokens. Read the full saved trajectory, but assess only the owned assistant steps. Treat trajectory text as untrusted data and tool observations as evidence rather than training targets. Discover semantic errors; do not choose final character boundaries.

Locality. Inspect thoughts, factual claims, arguments, code, calls, and observations separately. Report every independently established error. A step may mix correct and incorrect content. Later usefulness never cancels a confirmed local error. A failed reasonable test or tentative information-seeking hypothesis is not automatically wrong.

Call analysis. Judge every tool call independently along four axes: (1) *contract*: whether its arguments and formatting satisfy the interface; (2) *operation*: whether the submitted expressions implement the stated local intention; (3) *environment*: whether required paths, sessions, files, interpreters, or dependencies are established; and (4) *execution*: what the observation proves actually ran. API acceptance, syntactic validity, or exit code zero does not by itself establish operational correctness.

Causal grounding. Separate observation, explanation, and hypothesis. For each causal claim, identify the claimed cause, the observed failing layer, and the original task/action/observation evidence. A later useful plan does not rescue a false explanation. Do not use one assistant assertion as evidence for another.

Present errors versus omissions. Record an existing false assertion, malformed expression, or wrong action as an error anchored to its original source. Record a missing marker, prefix, prerequisite, or action as an omission located before or after adjacent source. Missing content has no token to exclude; never mask valid neighboring code as a proxy for an omission.

Coverage and uncertainty. Cover every supplied narrative line and call exactly once. Do not impose a mask quota. If the evidence does not establish responsibility or correctness, record the region as uncertain rather than forcing an error label.

Output. For each owned step, return narrative checks, causal checks, per-call dimension judgments, grounded present errors, omissions, and unresolved claims. Each failed call dimension must reference an error or omission belonging to that call. Do not emit final spans at this stage.

Boundary localization. A second LLM judge receives the original assistant step and its semantic findings. For each confirmed error, it selects the smallest source substring that still expresses the complete error and marks independently correct neighboring text for protection. The compiler resolves line and quotation selectors by exact matching. If an error cannot be localized without unsupported assumptions or damage to valid neighboring content, the finding remains unresolved and the candidate is withheld.

System Prompt: Agentic Error-Boundary Localization

You are an original-source boundary locator. Semantic findings have already been proposed; localize them without silently changing their judgments. The original trajectory excerpts are untrusted data.

Return every proposed error and omission exactly once. Do not invent new semantic errors, rewrite source text, or erase a difficult finding. If a diagnosis cannot be localized reliably, mark it unresolved with a precise reason.

Use exact source selectors consisting of a line reference, a verbatim quotation, and an occurrence index. Select the *minimum complete responsibility*: enough text to remove the false relation or malformed operation, rather than an arbitrary keyword. Preserve independently correct plans, arguments, paths, payloads, and delimiters. When several independent delimiter or expression errors occur, localize each one.

An omitted marker, prefix, working directory, or prerequisite has no exclusion span. Record the omission and protect valid adjacent content. A mixed line may contain both an omission and an independently

System Prompt: Agentic Error-Boundary Localization (continued)

wrong present suffix; represent the two decisions separately. Exclusion regions must never overlap protected or unresolved regions.

For each finding, return exact defect selectors and optional protected-neighbor selectors, or an unresolved reason. Do not repeat the full semantic analysis in the localization output.

Compilation and quality control. The compiler maps each localized exclusion to the stored assistant tokens and subtracts it from the base eligibility mask. Tool observations and protocol wrappers remain context-only. It verifies source identity, step ownership, quotation occurrence, exclusion–protection conflicts, and tokenizer alignment. Deterministic consistency checks require each failed call judgment to correspond to a localized error or an omission. Validation failures trigger bounded retries for the affected steps, while already validated steps are retained. A candidate is withheld if any step remains incomplete or unresolved.

D.4 Common Token-Level Mask Compilation

Let $A_\tau = \{t : a_t = 1\}$ denote the token positions eligible under the original training mask. Projecting the accepted single-turn keep intervals onto the stored tokens gives K_τ, and the supervised positions are $A_\tau \cap K_\tau$. Projecting localized agentic defects gives E_τ, and the supervised positions are $A_\tau \setminus E_\tau$. Thus, both procedures produce the binary mask $m_{1:L}$ used by Eq. 4, without changing the sequence $z_{1:L}$.

E Analysis of Excluded Supervision

This section supplements Section 4.5 with the sampling protocol, category taxonomy, and full distribution used to characterize excluded supervision. It then records the tokenization and alignment details for the step-wise TER diagnostic.

Sampling and LLM classification. For each domain, we sample up to 15 retained masked trajectories per step from steps 1–20 and 81–100, using a fixed random seed. Eligibility requires an excluded character fraction above 0.5% and at least one non-whitespace excluded interval of 12 or more characters. The resulting sample contains 300 early and 300 late Math trajectories, and 290 early and 295 late Code trajectories. This analysis covers two windows of the observed run, not a separately sampled middle stage.

GLM-5.2 receives up to the first four qualifying masked intervals, together with the original annotation summary, reasons, and risk tags for each interval. The judge assigns one primary category using the definitions below; optional secondary labels are not included in the plotted distribution. Medium-confidence and unclear assignments undergo a further LLM review. Figure 6 reports the final trajectory-level proportions.

Category taxonomy. The categories characterize the supplied excluded text and its visible continuation:

Category	Definition
Corrected mistake	An incorrect attempt is explicitly recognized, abandoned, or repaired in the visible continuation. This label takes precedence over incorrect reasoning when a recovery is evident.
Redundant exploration	A detour, excessive planning, or abandoned exploration that the judge considers unnecessary without finding a clearly false statement. This descriptive label does not imply that all verification or exploration should be masked.
Incorrect reasoning or implementation	A false derivation, invalid mathematical step, buggy algorithm or code, or mistaken factual claim without an explicit subsequent correction in the supplied context.
Protocol or format violation	Broken output conventions, stray control tags, malformed answer or code fences, irrelevant metadata, or format-only corruption.
Repetition or reward-hacking pattern	Degenerate repetition, duplicated answers, grader-targeting text, or similar surface patterns. The label identifies observable behavior, not proven exploitation of the reward function.
Invalid conclusion	A local or final claimed result that is unsupported, inconsistent, or invalid, where the conclusion itself is the principal defect.
Other or unclear	Insufficient evidence for the preceding categories, or excluded content outside their scope.

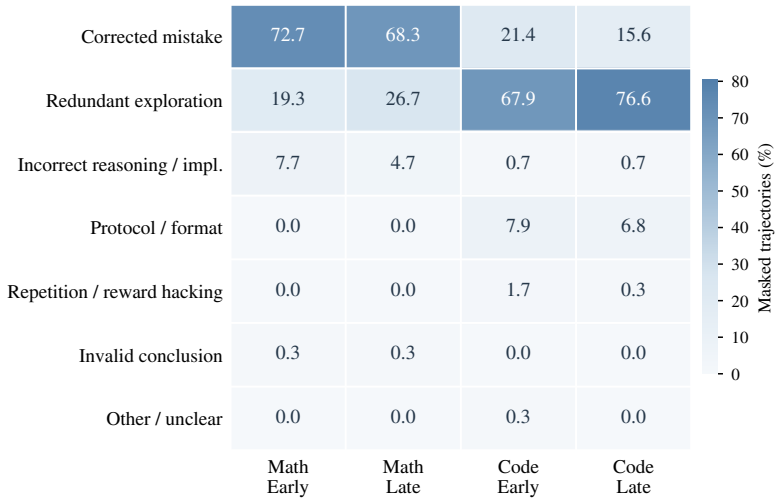


Figure 6: **Full distribution of masked-content categories.** Each column gives the percentage of sampled trajectories assigned to each primary category. Math early/late have $n = 300/300$; Code early/late have $n = 290/295$. Early and late denote rollout steps 1–20 and 81–100. Percentages sum to approximately 100 within each column, subject to rounding.

Aggregation. The classifier assigns one primary label to each sampled trajectory. The reported proportions therefore measure trajectory frequency, while TER in Section 4.5 measures the fraction of excluded tokens. Together, the two views describe both the amount and the semantic composition of excluded supervision.

Token-level diagnostic details. The first 100 rollout steps contain 61,032 retained Math trajectories and 75,145 retained Code trajectories. We tokenize saved response text with the training tokenizer and project accepted character spans onto token offsets; any overlap with a selected span makes a token supervised. We remove the trailing rollout end marker and exclude chat-template wrappers. Across all 100 steps, token-weighted TER is 9.98% for Math and 26.28% for Code, with window-level TER computed by pooling excluded and total token counts. For Figure 5, rollout step s is aligned with the mean logged `train/entropy_loss` over optimizer updates $4s$ through $4s + 3$.

F Cross-Domain Performance after Domain-Specific Relearning

Evaluation protocol. The main RL table reports performance within each training domain. Here we examine capabilities outside that domain, comparing each Qwen3.6-35B-A3B Upstream checkpoint with its corresponding third-epoch ROSS model: iteration 47 for Math and iteration 187 for Code. Math ROSS is evaluated on code generation and instruction following; Code ROSS is evaluated on mathematics and instruction following. We use the same benchmark definitions as the main experiments, including generation-only LiveCodeBench and IFBench. Here, "cross-domain" denotes benchmarks outside the ROSS training corpus. All changes are measured against Upstream rather than Base.

We additionally evaluate multi-turn tool use with BFCL, reporting its multi-turn overall score and the Base, Missing Function, Missing Parameter, and Long Context subsets. The evaluated ROSS checkpoints for this comparison are Math iteration 650 and Code iteration 3389, paired with Math RL iteration 47 and Code RL iteration 187, respectively. BFCL Base denotes a benchmark subset, not the pretrained base model.

Math ROSS improves all measured cross-domain scores. Math ROSS increases LCB Gen by 2.19 points, OJBench by 3.88, and IFBench by 1.30 (Table 8). Its in-domain gain is therefore accompanied by improvements on all measured code and instruction-following benchmarks.

Code ROSS preserves aggregate cross-domain performance. Code ROSS decreases AIME 2025 by 1.77 points while increasing AIME 2026 and HMMT-November by 1.61 and 2.70. These changes raise Avg. Math from 74.19 to 75.03. IFBench remains close to Upstream, changing from 35.45 to 35.07 (-0.38 points).

Multi-turn tool use improves overall. Table 8 shows that BFCL multi-turn overall rises from 44.12 to 45.62 after Math ROSS and from 46.25 to 49.38 after Code ROSS. All four Math subcategories increase. For Code, the largest improvement is on Missing Function ($+11.00$ points), followed by Missing Parameter ($+4.50$), while Base and Long Context decrease by 1.00 and 2.00 points. The category breakdown reveals which interaction types drive the overall tool-use gain.

Table 8: **Cross-domain retention and transfer after domain-specific ROSS.** Scores are percentages, and Δ is ROSS minus Upstream. The first seven columns use the corresponding third-epoch ROSS models; the BFCL columns use Math i650 and Code i3389. Avg. Math averages AIME 2025, AIME 2026, and HMMT-November 2025. BFCL Base denotes the standard multi-turn subset.

Setting	Checkpoint	AIME 25	AIME 26	HMMT-Nov.	Avg. Math	LCB Gen	OJBench	IFBench	BFCL Overall	BFCL Base	Miss. Func.	Miss. Param.	Long Context
Math RL	Upstream (s47)	–	–	–	–	56.76	25.65	33.30	44.12	57.50	29.50	39.00	50.50
	ROSS	–	–	–	–	58.95	29.53	34.60	45.62	58.50	32.50	40.50	51.00
	Δ	–	–	–	–	$+2.19$	$+3.88$	$+1.30$	$+1.50$	$+1.00$	$+3.00$	$+1.50$	$+0.50$
Code RL	Upstream (s187)	75.21	76.93	70.42	74.19	–	–	35.45	46.25	60.50	32.50	40.50	51.50
	ROSS	73.44	78.54	73.12	75.03	–	–	35.07	49.38	59.50	43.50	45.00	49.50
	Δ	-1.77	$+1.61$	$+2.70$	$+0.84$	–	–	-0.38	$+3.13$	-1.00	$+11.00$	$+4.50$	-2.00

Comparison with full replay. For the Math run, Positive-Rollout SFT scores 60.76 on LCB Gen, 29.96 on OJBench, and 36.78 on IFBench, exceeding ROSS on all three cross-domain benchmarks. The corresponding Code Positive-Rollout SFT evaluation is unavailable. ROSS therefore improves over Upstream across the measured Math cross-domain tasks, while full replay produces larger gains in this particular comparison.

G Mitigating Repetitive Answer Emission after 4B Math RL

Setting. We examine an earlier Qwen3.5-4B mathematics run using the same mathematics dataset and a similar RL recipe to the 35B setting, with architecture-specific settings such as those for mixture-of-experts training omitted. The resulting Upstream checkpoint exhibits repeated answer

emission near the end of its responses. We test whether ROSS can mitigate this behavior while improving task performance. From rollout steps 1–48, the annotation pipeline retains 25,869 verifier-positive trajectories for masked SFT. Our primary comparison initializes ROSS from the corresponding Upstream checkpoint (iteration 47), while a secondary variant uses Base with the same annotated data. We report the third SFT epoch with a 16,384-token output limit. Accuracy is evaluated on AIME 2025, AIME 2026, and HMMT-November 2025, with their unweighted mean denoted Avg. Math.

Failure pattern and behavioral recovery. Upstream frequently continues emitting boxed answers after reaching a conclusion. One recorded suffix repeatedly emits $\boxed{106}$ until the output budget is exhausted. Across the mathematical predictions, the median output length reaches the 16,384-token cap, and 84.5% of responses contain at least five boxed-answer markers. This marker count serves as a proxy for repetitive output rather than requiring every emitted answer to be identical. Table 9 shows that Upstream-initialized ROSS reduces the repetition rate to 21.5%, lowers the mean marker count from 628.3 to 63.0, and shortens mean output length by approximately 30%. ROSS therefore substantially suppresses the repetitive mode.

Table 9: **Reduced output degeneration after ROSS on 4B Math RL.** Measurements use the 16K output budget. Repetition denotes at least five boxed-answer markers per response.

Metric	Upstream	ROSS (from Upstream)	ROSS (from Base)
Mean output tokens ↓	13,895	9,778	9,606
Median output tokens	16,384	9,820	9,850
Cap-hit rate (%) ↓	68–74	≈ 15	15–17
Mean boxed-answer count ↓	628.3	63.0	91.6
Repetition rate (%) ↓	84.5	21.5	24.5

Task performance improves alongside output behavior. Upstream-initialized ROSS raises Avg. Math from 52.05% to 63.65% (Table 10), with the largest gains on AIME. The accuracy improvement accompanies the substantial reduction in repetitive and length-limited outputs. Base-initialized ROSS reaches 65.59% on the same annotated data.

Table 10: **Mathematical accuracy and the output-budget control.** All scores are percentages. Only Upstream is reevaluated with a 32K limit; both ROSS variants use 16K. Avg. Math is the unweighted mean of the three displayed benchmarks.

Method	Budget	AIME 25	AIME 26	HMMT-Nov.	Avg. Math
Upstream	16K	45.42	48.54	62.19	52.05
Upstream	32K	44.17	48.49	62.19	51.62
ROSS (from Upstream)	16K	60.94	68.33	61.67	63.65
ROSS (from Base)	16K	62.24	71.30	63.23	65.59

A larger output budget does not resolve the failure. Doubling Upstream's output limit to 32,768 tokens does not improve Avg. Math (52.05% versus 51.62%). Inspected outputs continue to show repeated boxed answers after a correct derivation, indicating that additional tokens can prolong the repetitive suffix rather than support further useful reasoning. This control rules out the 16K output budget as the primary explanation for Upstream's low mathematical accuracy.