

Self-Supervised Rehearsal for Continual RLVR via Feedback Prediction

Research Proposal

Rohan Sikand · Stanford University

March 2026

Sequential RLVR Forgets

Real LLM post-training is multi-stage. Each RLVR stage targets a different capability:

$$\pi_{\text{base}} \xrightarrow{\text{RLVR (math)}} \pi_1 \xrightarrow{\text{RLVR (code)}} \pi_2 \xrightarrow{\text{RLVR (science)}} \pi_3$$

At each stage, prior capabilities degrade:

- After code RLVR: math Pass@1 drops
- After science RLVR: both math and code degrade
- GLM-5 found this severe enough to require a full recovery stage
- Yue et al. showed RLVR narrows the distribution — sequential RLVR narrows it repeatedly

Every team building reasoning models faces this problem. The current solutions are

RLVR Forgetting $\neq$ SFT Forgetting

This distinction is crucial and under-studied:

SFT forgetting:

- Next-token loss directly overwrites representations
- Capabilities are *destroyed*
- The model can't produce task A solutions even with aggressive sampling
- Hard to fix without replay

RLVR forgetting:

- RL reshapes the sampling distribution, not representations directly
- Capabilities are *de-prioritized*
- The model may still produce task A solutions at high temperature / large k
- Fixable by maintaining distributional breadth

Hypothesis: After RLVR on task B, task A's Pass@1 drops but Pass@256 (high temperature) partially or fully recovers. If true, the model hasn't forgotten task A — it

The Diagnostic Experiment (Novel, Contribution on Its Own)

Nobody has measured this. Before proposing solutions, we need to understand the problem:

Protocol: RLVR on math (200 steps) $\rightarrow$ RLVR on code (200 steps). Measure at every checkpoint:

Metric	After math	After code	Interpretation
Math Pass@1	High	Drops	Expected
Math Pass@64 (temp=0.9)	High	?	Key measurement
Math Pass@256 (temp=1.0)	High	?	Capability vs. distribution
Math solution diversity	High	?	Mode collapse extent
Code Pass@1	Low	High	Expected

If Pass@256 recovers: “forgetting” is distribution narrowing $\rightarrow$ fix the distribution

The Key Observation

At every RLVR training step, the environment produces structured execution feedback: stack traces, interpreter errors, assertion failures, partial outputs.

This feedback encodes the **causal relationship between code and execution** — which is the underlying capability the model needs to preserve.

A model that can accurately predict *“this math code will produce `AssertionError` at line 7”* has maintained its internal model of math execution — even if its sampling distribution has shifted toward code.

Feedback prediction is a probe for capability preservation.

If the model can still predict math execution feedback,

it can still reason about math —

The Method: Feedback Prediction as Rehearsal

During RLVR on task B (code), add a self-supervised maintenance loss on task A (math):

$$\mathcal{L}_{\text{pred}}^{(\text{math})}(\theta) = -\mathbb{E}_{s \sim \mathcal{S}_{\text{math}}} \left[\frac{1}{|f_i|} \sum_k \log \pi_{\theta}(f_{i,k} \mid s_i, a_i, f_{i,<k}) \right]$$

where $a_i \sim \pi_{\theta}(\cdot \mid s_i)$ is a fresh rollout on a math problem and f_i is the interpreter's execution feedback.

Full training objective during code RLVR:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \left[\underbrace{\mathcal{L}_{\text{GRPO}}^{(\text{code})}}_{\text{learn code}} + \lambda \underbrace{\mathcal{L}_{\text{pred}}^{(\text{math})}}_{\text{maintain math}} \right]$$

Why This Works (Mechanistic Argument)

Level 1: Representation preservation.

$\mathcal{L}_{\text{pred}}$ requires the model to predict execution outcomes from code. This exercises the *same internal circuits* that math reasoning depends on: understanding variables, control flow, arithmetic, type errors. If these circuits are kept active by feedback prediction, they're available for math problem-solving.

Level 2: Distribution breadth.

The math rollouts in $\mathcal{L}_{\text{pred}}$ are generated on-policy ($a_i \sim \pi_\theta$). Even if the rollouts are mostly wrong (the distribution has shifted toward code), the model must still *engage with math* at every training step. This prevents the complete abandonment of math-relevant generation patterns.

Level 3: Implicit calibration.

What This Is Not

Method	What it preserves	Cost
Replay buffer	Surface behavior (correct solutions)	Storage + staleness
KL regularization	Total distribution (blunt)	Prevents useful learning
EWC / Fisher	Important weight directions	Expensive computation
GLM-5 recovery	All prior capabilities (retroactive)	Extra training stage
Feedback prediction	Internal execution model (the underlying capability)	One fwd + bwd pass per batch, no storage

The key distinction: replay preserves *what the model outputs*. KL preserves *the output distribution*. Feedback prediction preserves *the internal reasoning capability*. The first two are behavioral; the third is mechanistic.

A model that maintains its internal execution model can regenerate math solutions once prompted — even if it stopped spontaneously producing them during code RLVR.

The Training Loop

For each training step during code RLVR:

1. Code RLVR (primary):

- Generate G code rollouts per prompt
- Execute, collect feedback and R_{env}
- Compute GRPO advantages, backward pass

2. Math rehearsal (maintenance):

- Sample a mini-batch of math problems
- Generate one rollout per problem (on-policy, may be wrong — that's fine)
- Execute, collect execution feedback f_i
- Backward pass on $\mathcal{L}_{\text{pred}}^{(\text{math})}$: predict f_i from (s_i, a_i)

Extending to N Prior Tasks

The method scales naturally to maintaining N prior tasks during task $N+1$ RLVR:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \left[\mathcal{L}_{\text{GRPO}}^{(\text{new task})} + \sum_{j=1}^N \lambda_j \mathcal{L}_{\text{pred}}^{(\text{task } j)} \right]$$

Each prior task contributes a feedback prediction loss. The cost scales linearly in N but with a small constant per task (one mini-batch of rollouts + execution + prediction).

No replay buffer grows. Classical replay stores examples from all prior tasks, growing without bound. Feedback prediction generates fresh on-policy rollouts at each step — no storage, no staleness, always on-distribution.

Weighting λ_j : Can weight by task importance, by measured degradation (increase λ_j if

Verifier Calibration as a Complementary Signal

Feedback prediction maintains the execution model. Add verifier calibration to maintain judgment quality:

$$\mathcal{L}_{\text{cal}}^{(\text{math})}(\theta) = \mathbb{E} \left[(R_{\text{self}}^{(\text{math})}(s, a) - R_{\text{env}}^{(\text{math})}(s, a))^2 \right]$$

During code RLVR, the model judges its own math rollouts and the calibration loss ensures those judgments stay anchored to ground truth.

Full continual objective:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \left[\mathcal{L}_{\text{GRPO}}^{(\text{code})} + \lambda_1 \mathcal{L}_{\text{pred}}^{(\text{math})} + \lambda_2 \mathcal{L}_{\text{cal}}^{(\text{math})} \right]$$

Experimental Plan

Phase 1: Diagnostic (1 week)

- RLVR Qwen3-4B on math (200 steps) → RLVR on code (200 steps)
- Measure math Pass@1, Pass@64, Pass@256 at checkpoints throughout code training
- Establish: is this distribution narrowing or capability loss?

Phase 2: Core comparison (3 weeks)

Same sequential setup with five conditions:

Condition	Rehearsal method	Labels?	Storage?
(a) No rehearsal	—	No	No
(b) Replay buffer	Stored math solutions	Yes	Yes
(c) KL regularization	Penalize drift from math checkpoint	No	1 ckpt
(d) $\mathcal{L}_{\text{pred}}$ only	Feedback prediction on math	No	No
(e) $\mathcal{L}_{\text{pred}} + \mathcal{L}_{\text{cal}}$	Prediction + calibration	No	No

Experimental Plan (continued)

Metrics for Phase 2:

- Math Pass@1 retention (primary): how much math capability survives?
- Code Pass@1 improvement (must not degrade): does rehearsal slow code learning?
- Math Pass@k (k=64, 256): distributional breadth preservation
- Math solution diversity: cluster count before/after code RLVR
- Wall-clock overhead: how much does each rehearsal method cost?

Phase 3: Scale to 3 tasks (2 weeks)

- RLVR on math $\rightarrow$ code $\rightarrow$ science reasoning
- Maintain math *and* code during science RLVR via $\sum_j \lambda_j \mathcal{L}_{\text{pred}}^{(j)}$
- Does the method degrade with more tasks to maintain?

Expected Results

Condition	Math Pass@1	Code Pass@1	Overhead
(a) No rehearsal	Low (forgetting)	High	0%
(b) Replay buffer	Medium-high	High	Medium
(c) KL regularization	Medium	Medium (hurts code)	Low
(d) $\mathcal{L}_{\text{pred}}$	Medium-high	High	$\sim 10\%$
(e) $\mathcal{L}_{\text{pred}} + \mathcal{L}_{\text{cal}}$	High	High	$\sim 15\%$

Key hypotheses:

- (d) and (e) match or beat (b) replay without any stored data
- (c) KL hurts code learning because it's too blunt — prevents useful distribution shift
- (e) outperforms (d) because calibration maintains judgment, not just prediction
- Overhead is modest ($\sim 10\text{--}15\%$) because math mini-batch is small

Prior Work

Paper	Approach	Gap from ours
GLM-5 (2025)	Cross-stage distillation	Retroactive recovery, not prevention
Shenfeld et al. (2025)	RL forgets less than SFT	Observation only, no method
Yue et al. (2025)	RLVR = distribution narrowing	Diagnosis only
DPH-RL (2025)	Forward-KL for diversity	Single task, not continual
Classical CL	EWC, replay, regularization	Not RLVR-specific
PackNet / HAT	Parameter isolation	Not scaled to LLMs

The gap: No one has proposed a rehearsal method that exploits the specific structure of RLVR — the fact that verifiable environments produce execution feedback that can serve as a free, always-fresh, label-free maintenance signal.

Feedback prediction as continual learning rehearsal is novel. The individual components exist ($\mathcal{L}_{\text{pred}}$ from self-supervised learning, rehearsal from CL), but their combination for RLVR is unexplored.

Risks and Mitigations

Risk	Severity	Mitigation
RLVR forgetting is mild (diagnostic shows minimal degradation)	High	If mild, paper motivation weakens; pivot to measuring that RL doesn't forget
$\mathcal{L}_{\text{pred}}$ doesn't prevent forgetting (prediction $\neq$ capability)	High	Add $\mathcal{L}_{\text{cal}}$; if both fail, the mechanistic hypothesis is wrong
Rehearsal overhead slows code RLVR	Medium	Small math mini-batch (10%); measure code Pass@1 isn't hurt
KL baseline is just as good and simpler	Medium	Show KL hurts code learning; $\mathcal{L}_{\text{pred}}$ is more surgical

“Self-Supervised Rehearsal for Continual RLVR via Feedback Prediction”

1. **Introduction:** Sequential RLVR forgets. Existing solutions are expensive or blunt.
2. **Diagnostic:** Is RLVR forgetting capability loss or distribution narrowing? (Pass@k analysis — novel finding)
3. **Method:** Feedback prediction as label-free, storage-free, on-distribution rehearsal.
 $\mathcal{L}_{\text{pred}} + \mathcal{L}_{\text{cal}}$.
4. **Why it works:** Representation preservation, distributional breadth, implicit calibration.
5. **Experiments:** 5-way comparison on sequential math $\rightarrow$ code RLVR. Scale to 3 tasks.
6. **Analysis:** What does $\mathcal{L}_{\text{pred}}$ preserve? (Probing, attention analysis.) Ablations.

Honest Assessment

Pros:

- Clean, focused story: one method, one problem, clear baselines
- The diagnostic alone is a contribution (nobody has characterized RLVR forgetting via pass@k)
- Falls naturally out of EGSR — same $\mathcal{L}_{\text{pred}}$, different application
- Label-free, storage-free, always on-distribution — genuinely better properties than replay
- RLVR-specific rehearsal is a novel niche (CL literature focuses on SFT)
- Low engineering complexity: one extra loss term during training

Cons: