

Continual RLVR: When Sequential Post-Training Forgets

Research Direction — Connecting EGSR, Multi-Objective, and Population Ideas

Rohan Sikand · Stanford University

March 2026

The Problem

Modern LLM post-training is multi-stage and multi-task:

$$\pi_{\text{base}} \xrightarrow{\text{SFT}} \pi_1 \xrightarrow{\text{RLVR (math)}} \pi_2 \xrightarrow{\text{RLVR (code)}} \pi_3 \xrightarrow{\text{RLHF (general)}} \pi_4$$

At each stage, capabilities from prior stages degrade. After RLVR on code, math performance drops. After RLHF, reasoning sharpness dulls.

This is not hypothetical:

- GLM-5 found it severe enough to add an entire recovery stage
- DeepSeek-R1 carefully interleaves SFT and RL stages to mitigate it
- Yue et al. showed RLVR narrows the distribution — sequential RLVR narrows it twice

RLVR Forgetting is Different from SFT Forgetting

SFT forgetting:

- Loss directly maximizes likelihood on new data
- Aggressively overwrites representations
- The model *can't* do task A anymore
- Capabilities destroyed

RLVR forgetting:

- RL reshapes the *distribution*, not the weights directly
- Sharpens on task B modes
- Mass covering task A modes gets reallocated
- Capabilities *de-prioritized*, not destroyed

Key question: If you increase temperature after RLVR on task B, can you recover task A solutions? If so, the model hasn't truly forgotten — it's just unlikely to *sample* task A solutions under greedy decoding. This is distribution narrowing, not capability loss.

This distinction matters for the solution. If capabilities are destroyed, you need replay or regularization. If they're de-prioritized, you need distribution-preserving training.

The Diagnostic Question Nobody Has Asked

Is RLVR “forgetting” actually forgetting?

Experiment: Train RLVR on math, then RLVR on code. Measure:

Metric	Capability loss	Distribution narrowing
Pass@1 (math)	Drops	Drops
Pass@64 (math, high temp)	Drops	Recovers
Pass@256 (math, high temp)	Still drops	Near original
Math solution diversity	Collapses	Preserved under sampling

If Pass@1 drops but Pass@256 recovers, the model still *can* solve math — it just doesn't *choose to* under the narrowed distribution. This is a much easier problem to fix than genuine capability loss.

This diagnostic doesn't exist in the literature. It would be a contribution on its own.

Existing Solutions and Their Limitations

Method	Mechanism	Used by	Limitation
Interleaved stages	Alternate SFT/RL	DeepSeek-R1	Manual scheduling
Cross-stage distill	Recovery from teachers	GLM-5	Retroactive fix; extra stage
Replay buffer	Mix old task data	Classical CL	Staleness; storage cost
KL regularization	Penalize drift from ref	DPO, PPO-clip	Coarse; prevents all change
EWC / Fisher	Protect important weights	Classical CL	Expensive; LLM-scale issues
Forward-KL	Mass-covering divergence	DPH-RL	Single-task only so far

Common thread: all are **general-purpose** continual learning methods applied to LLMs. None exploit the specific structure of RLVR — the fact that verifiable environments produce rich feedback that can serve as a maintenance signal.

Solution 1: Multi-Objective RLVR (Avoid the Problem Entirely)

Core insight: Continual learning is only a problem because tasks are trained *sequentially*. If all tasks are optimized *simultaneously*, there's nothing to forget.

Sequential: $\xrightarrow{\text{RLVR (math)}} \xrightarrow{\text{RLVR (code)}}$ vs. Simultaneous: $\xrightarrow{\text{RLVR (math + code)}}$

Lexicographic GRPO:

- Primary objective: math correctness (never degraded)
- Secondary: code correctness (optimized subject to primary)
- Or: treat all tasks equally and use Pareto GRPO

Limitation: Requires all tasks available at training time. If task B arrives after deployment, you can't retroactively add it. But for the common case (train on math +

Solution 2: Self-Supervised Rehearsal via Feedback Prediction

Core insight: Classical replay requires storing labeled examples from task A. But RLVR environments can *generate* rehearsal signal on demand — no storage needed.

During RLVR on task B (code), add a maintenance objective on task A (math):

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

The model must *predict what execution feedback math code would produce*. This forces it to maintain the internal representations needed for math reasoning — even though the RL loss is only optimizing code.

Properties:

- **Label-free:** No correct solutions needed, just problem + rollout + feedback

Why Feedback Prediction is a Natural Rehearsal Signal

Classical rehearsal: “remember what the correct answer to this math problem looks like.”

Feedback prediction rehearsal: “remember what happens when you run math code.”

The second is **more fundamental**. Knowing what correct math solutions look like is a surface-level behavior. Understanding the causal relationship between math code and its execution outcomes is the underlying *capability*. If the model maintains its internal execution model for math, it can still solve math problems — the representations are preserved even if the sampling distribution has shifted toward code.

Analogy: A chess player who switches to studying Go doesn't forget how pieces move — they might forget their favorite openings (distribution narrowing) but the underlying spatial reasoning is preserved. Feedback prediction preserves the `how pieces move` level, not the `favorite opening` level.

Solution 3: Teacher-Based Continual RLVR (GLM-5 Adapted)

Core insight: When starting RLVR on task B, keep the task A checkpoint as a teacher. Use GLM-5's advantage formulation as a *per-token regularizer*.

During task B training, for task A prompts:

$$\hat{A}_{i,t}^{(\text{task A})} = \text{sg} \left[\log \frac{\pi_{\theta_{\text{task A}}}(y_{i,t} \mid x, y_{i,<t})}{\pi_{\theta}^{\text{current}}(y_{i,t} \mid x, y_{i,<t})} \right]$$

If the current policy starts diverging from the task A teacher on task A prompts, the advantage signal pushes it back. Token-level specificity: only penalizes tokens where the policy has actually drifted, not a blanket KL penalty.

Comparison to KL regularization:

- KL : penalizes total divergence (blunt - prevents all change)

Solution 4: Population-Based Continual RLVR

Core insight: Don't train one policy sequentially. Train specialist policies in parallel and merge periodically.

$$\pi_{\text{base}} \xrightarrow[\text{parallel}]{\text{fork}} \begin{cases} \pi_{\text{math}} & \text{RLVR on math} \\ \pi_{\text{code}} & \text{RLVR on code} \end{cases} \xrightarrow{\text{merge}} \pi_{\text{unified}}$$

No forgetting because no policy trains on both tasks. Each specialist fully optimizes its domain. The merge step (via teacher-based advantage, forward-KL, or SFT on union) combines capabilities.

For truly continual settings (task C arrives after deployment):

- Fork from π_{unified} , train $\pi_{\text{task C}}$ specialist

How the Solutions Compare

	Multi-obj	Feedback pred	Teacher reg	Population
Prevents forgetting?	By design	Partially	Partially	By design
Needs all tasks upfront?	Yes	No	No	No
Extra compute	Moderate	Minimal	Minimal	$K \times$
Extra storage	None	None	1 checkpoint	K checkpoints
Truly continual?	No	Yes	Yes	Yes
Novel for RLVR?	Yes	Yes	Partially (GLM-5)	Yes
Complexity	High	Low	Low	Medium

For known task sets: Multi-objective RLVR is the principled answer.

For continual/streaming tasks: Feedback prediction is the cheapest intervention. Teacher regularization is a stronger guarantee. Population is the most robust but most expensive.

These are **complementary**, not competing. Use multi-objective where possible, add

The Unified Picture

All four solutions are instances of one principle:

*Maintain multi-dimensional capability representations
rather than collapsing to a single task's distribution.*

- Multi-objective: maintain multiple *objectives* simultaneously
- Feedback prediction: maintain the internal *execution model* for all domains
- Teacher regularization: maintain *behavioral alignment* with prior checkpoints
- Population: maintain multiple *policies*, merge capabilities

Scalar reward → scalar optimization → single task → forgetting. Vector reward → multi-dimensional optimization → multi-task → preservation. The continual learning

What's Actually Novel Here

Contribution	Novelty	Status
Diagnostic: is RLVR forgetting real? (Pass@k recovery after sequential RLVR)	High	Nobody has measured pass@k under high temp
Feedback prediction as rehearsal ($\mathcal{L}_{\text{pred}}$ on old-task problems)	High	Falls out of EGSR but never proposed
Teacher-based continual regularization (per-token, task-specific)	Medium	GLM-5 adjacent but cross-task version is new
Multi-objective RLVR avoids forgetting (lexicographic/Pareto GRPO)	Medium-High	Framework exists for classical MORL, not LLM
Population for continual RLVR (fork-specialize-merge cycle)	Medium	PBT exists, not for continual LLM RLVR

Experimental Plan

Phase 1: Diagnostic (1 week, can piggyback on EGSR infra)

- RLVR Qwen3-4B on math (200 steps), then RLVR on code (200 steps)
- Measure math Pass@1, Pass@64, Pass@256 at checkpoints
- If Pass@256 recovers $\rightarrow$ distribution narrowing. If not $\rightarrow$ capability loss
- **This diagnostic is a contribution regardless of what solutions you try**

Phase 2: Feedback prediction rehearsal (2 weeks)

- Same sequential setup, but add $\mathcal{L}_{\text{pred}}$ on math problems during code RLVR
- Compare: no rehearsal, $\mathcal{L}_{\text{pred}}$ rehearsal, replay buffer, KL regularization
- Measure: math Pass@1 retention, code Pass@1 improvement, total compute

Phase 3: Teacher regularization (1 week)

- Same setup, but add GLM-5-style teacher advantage from math checkpoint

Paper Structure (If Pursued)

Option A: Focused paper (recommended)

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

- Diagnostic: RLVR forgetting = distribution narrowing (novel finding)
- Method: $\mathcal{L}_{\text{pred}}$ as label-free, replay-free rehearsal
- Baselines: replay buffer, KL reg, teacher advantage, no rehearsal
- Clean story, single mechanism, falls naturally out of EGSR

Option B: Broad framework paper (ambitious)

“Continual Reinforcement Learning with Verifiable Rewards”

- Full framework: diagnostic + four solutions + analysis of when to use which
- Unifying principle: multi-dimensional capability preservation
- Higher ceiling, much more work, needs multi-objective RLVR infrastructure

Connection to All Backburner Ideas

Idea	Continual learning connection	Mechanism
EGSR	$\mathcal{L}_{\text{pred}}$ is a natural rehearsal signal	Feedback prediction
Multi-objective RLVR	Avoids sequential forgetting entirely	Simultaneous optimization
Population RLVR	Fork-specialize-merge cycle	Modular specialists
Diverse rewards	Multiple axes maintain broader repr.	Reward landscape geometry
Latent perturbation	Broader latent coverage resists narrowing	Activation diversity

Everything converges. The exploration problem (RLVR only reweights), the multi-objective problem (scalar reward is lossy), and the continual learning problem (sequential training forgets) are all manifestations of the same root cause: **RLVR narrows the policy distribution**. Any method that maintains distributional breadth addresses all three simultaneously.

Honest Assessment

Pros:

- Continual RLVR is genuinely understudied (most CL work is SFT)
- The diagnostic alone is publishable (nobody has measured pass@k recovery)
- Feedback prediction rehearsal is novel, cheap, and falls out of EGSR
- Clean narrative: “EGSR’s maintenance mechanism naturally extends to continual settings”
- Multiple tractable paper options (focused, broad, or EGSR extension)

Cons:

- “Continual learning for LLMs” is a broad field with many general-purpose methods
- RLVR-specific forgetting may turn out to be mild (diagnostic kills the motivation)