

1. Why Day-0 RL matters

Model-specific integration can delay RL adoption of new open models.

- Adapt new capabilities to task-specific goals through rewards.
- Accelerate RL research with emerging architectures.
- Enable more teams to explore RL with less integration work.

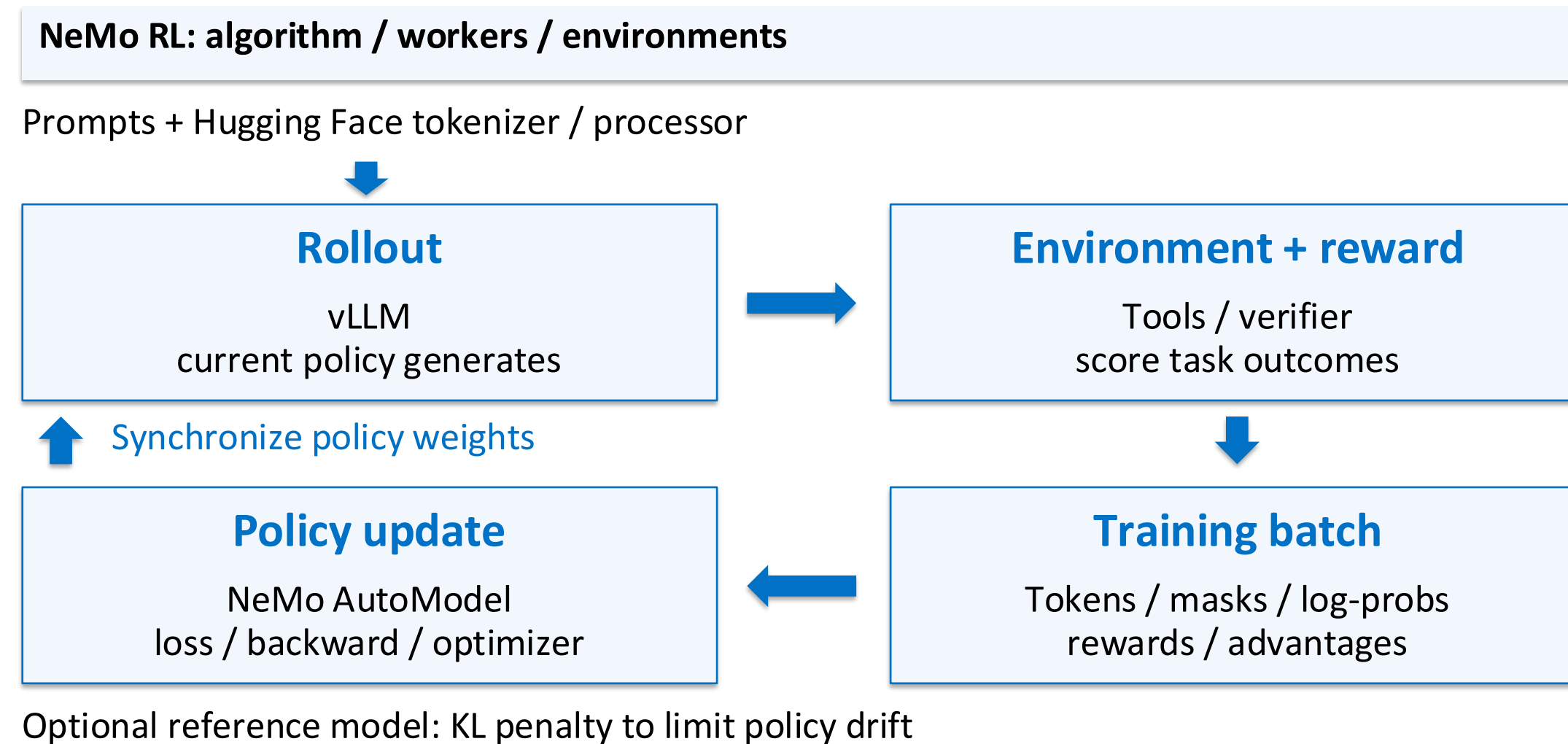
Goal: shorten the path from model release to reliable, scalable RL.

2. Challenges of Day-0 RL support

- Architecture diversity: varying attention patterns, expert routing, and multimodal components require model-specific handling across training and rollout.
- Training–rollout consistency: align input semantics, token log-probabilities, and updated weights.
- Numerical correctness: detect precision errors, incorrect gradients, and instability beyond reward.
- Distributed scalability: manage model and sequence memory, rollout capacity, and weight transfer.

3. Solution: a PyTorch-native foundation for Day-0 RL

The RL loop: NeMo RL + NeMo AutoModel [1]



NeMo AutoModel: beyond RL [2]

Day-0 model support as a design principle.

Shared foundation: pretraining / SFT / RL policy training

- Hugging Face interfaces: reuse model-loading APIs, tokenizers/processors, and checkpoints.
- PyTorch-native SPMD: DTensor, DeviceMesh, and FSDP2 with supported model parallelism (TP, PP, EP).
- Reusable configurations: compose parallelism, activation checkpointing, and distributed checkpointing.

Across model families and architectures

Dense and MoE LLMs, VLMs, dLLM and diffusion models. Model support enables RL integration; each end-to-end configuration still needs validation.

Best practices: validate, stabilize, then scale

1. Verify numerical correctness

Use identical weights and inputs to compare outputs, token log-probabilities, and gradients against a reference implementation.

2. Validate the RL loop

Check input consistency, training–rollout log-probability agreement, policy-weight synchronization, and checkpoint/resume.

3. Test sustained stability

Monitor nonfinite values, gradient norms, KL, entropy, and response lengths—not reward alone.

4. Revalidate after scaling

Repeat correctness and stability checks after changing precision, parallelism, or execution settings.

A model that runs is not yet a validated RL configuration.

Scaling: match the mechanism to the bottleneck

Model-state memory

Shard training states with FSDP2; add supported tensor or expert parallelism for the architecture.

Long-sequence memory

Use activation checkpointing to reduce saved activations; use supported context parallelism to distribute sequence computation.

Rollout throughput

Tune generation parallelism and batch capacity. Shared GPUs reuse resources; separate pools may overlap work, with weight-transfer costs.

Measure peak memory, rollout time, policy-update time, and weight-transfer time before changing the configuration.

4. Results: broad model support and validated RL training at scale

Selected NeMo AutoModel RL integrations [5]

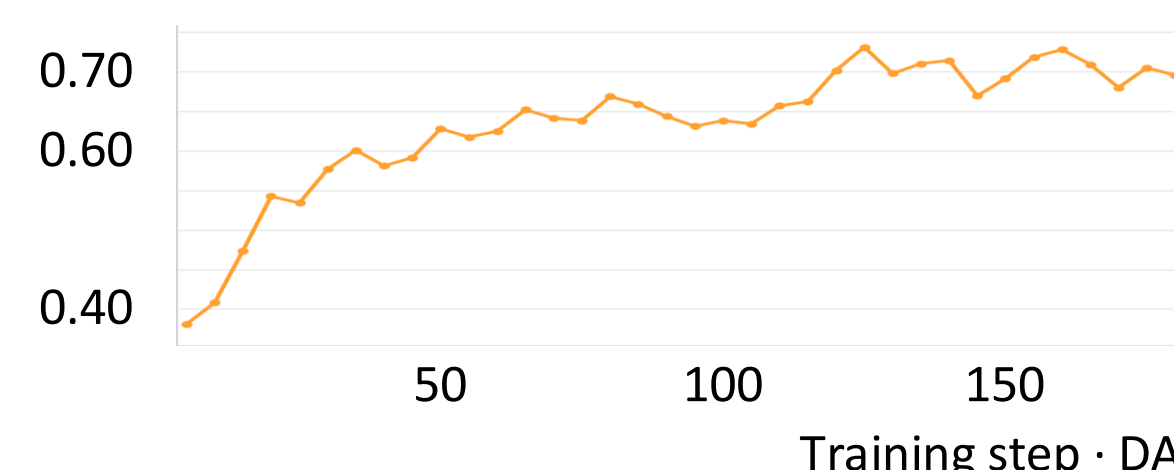
[Qwen3.8 Flash Next*](#)
[Muse Glimmer*](#)
[Gemma 4](#)
[Nemotron 3 Nano Omni](#)
[DeepSeek V4 Flash](#)
[MiniMax M3](#)

Text / VLM · dense / MoE
*Support-branch examples.
Validation scope varies by recipe.

Qwen3.8 Flash Next: RL training on 64 GPUs [3]

180B · 64 GPUs · FSDP2 + EP · 4K sequence

AIME 2024 accuracy



Train–rollout KL error (x 10⁻³)

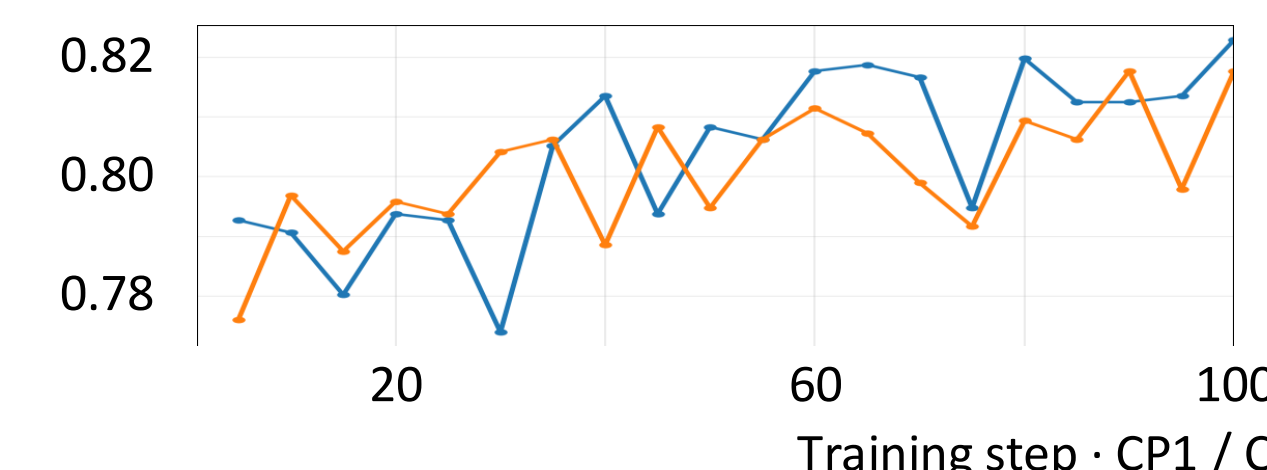


KL: training–rollout disagreement, not reference-policy drift.

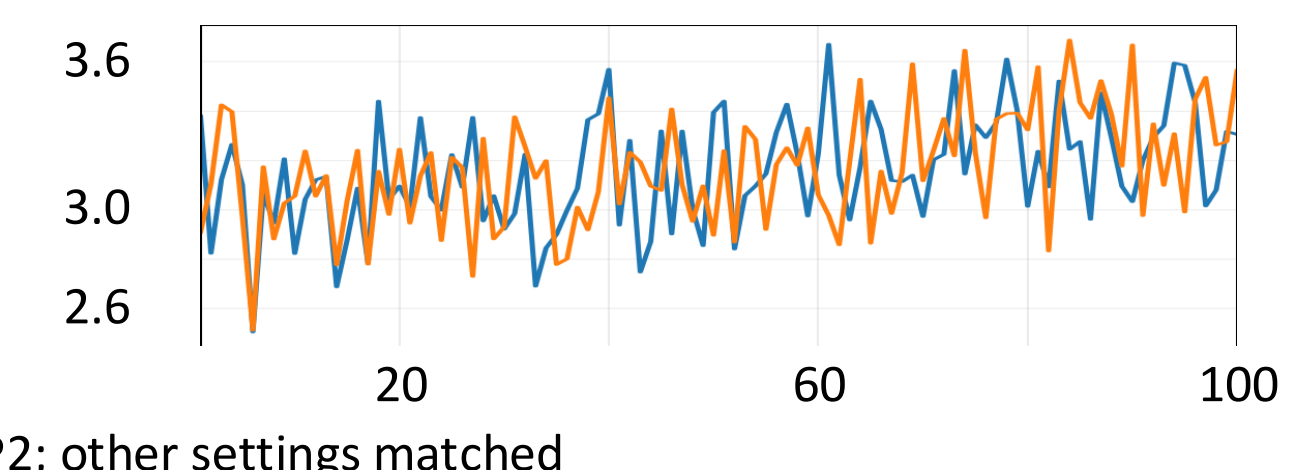
Gemma 4 31B: validation with context parallelism [4]

32 GPUs · CP1 vs. CP2 · 100 steps

AIME 2024 accuracy



Train–rollout KL error (x 10⁻⁴)



32K on 4 H100 nodes; context_parallel_size: 16 (separate setup).

References

[1] NeMo RL docs
[NeMo RL GitHub](#)

[2] NeMo AutoModel docs
[NeMo AutoModel GitHub](#)

[3] Qwen3.8 support + results
[\[4\] Gemma 4 CP results](#)

[5] Model guides / support branches
[CP: context parallelism · EP: expert parallelism](#)