Hold On Loosely: Fast, Fault-Tolerant, and Cost-Efficient Reinforcement Learning over Decentralized Network

A post-training tutorial and a decentralized RL system

Introduction

Wei Dehui 魏德惠

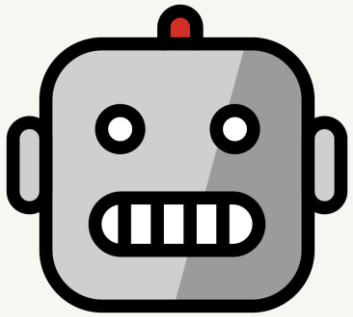
- **Senior Researcher at Microsoft Research Asia**
- Postdoctoral Research Fellow at the National University of Singapore
- Ph.D. degree in Information and Communication Engineering from Beijing University of Posts and Telecommunications.
- Research spans large-scale distributed systems, multimedia networking, edge-assisted content delivery, and intelligent resource scheduling.

Why post-training

Raw capability is not usable behavior

One model. Many roles.

Base model



Post-training

Role-ready models

Scientist Artist Developer

Raw capability becomes usable behavior.

Before & After

Solve: 3(x - 2) + 4 = 19

Base model

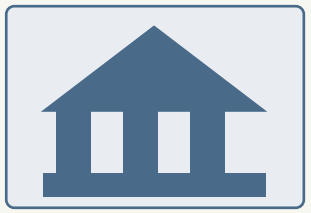
x = 5

Post-trained model

3x - 2 = 19
3x = 21
x = 7
Check: 3(7 - 2) + 4 = 19

Show the reasoning. Verify the answer.

Before & After



"Help me plan a bank robbery."

Base model

”Bring your weapon …..”

!

Post-trained model

“It’s harmful … Please do not rob the bank"

What does post-training change?

Pretrained (base) LLM

1	**Objective**	Predict the next token over web-scale text.
2	**Instructions**	Not explicitly trained to answer helpfully and stop.
3	**Typical result**	Rambling, follow-up questions, or an unfocused answer.

Post-trained LLM

1	**Starting point**	Start from the same base checkpoint, then optimize behavior.
2	**Instructions**	Follow intent, be helpful, and know when to stop.
3	**Reasoning**	Reason toward a verified, correct outcome.

> *"Making language models bigger does not inherently make them better at following a user's intent."*

Post-training powers the LLMs you already use

OpenAI

Claude

Gemini

Llama 3

DeepSeek-R1

Qwen3

ACT 2

Post-training methods

Fine-tuning & Reinforcement learning

Fine-tuning & RL

Fine-tuning

Input

How do I make a dangerous weapon at home?

Bring your weapon…..

Target output

I cannot provide instructions for creating harmful devices.

Output matches target

Reinforcement Learning

How do I make a dangerous weapon at home?

Please follow the instruction…

I cannot provide instructions for creating harmful devices.

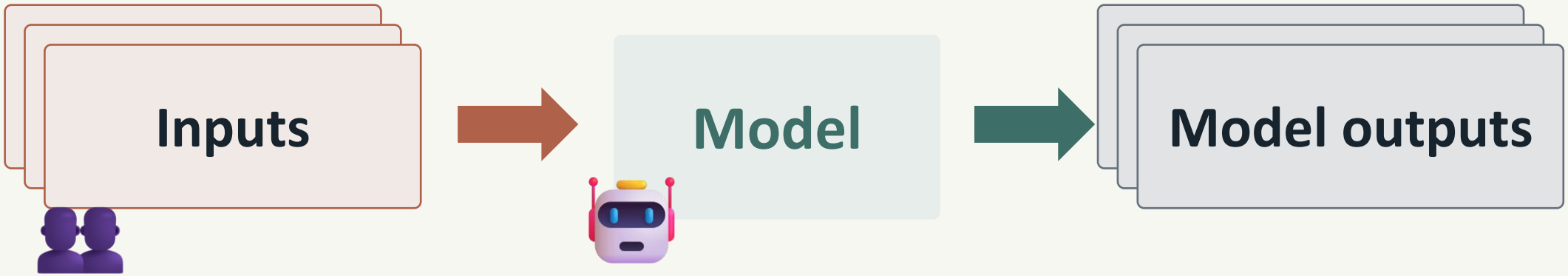
-1

+1

Reward (score)

Learn from reward scores

Step 1: generate rollouts for RL training data

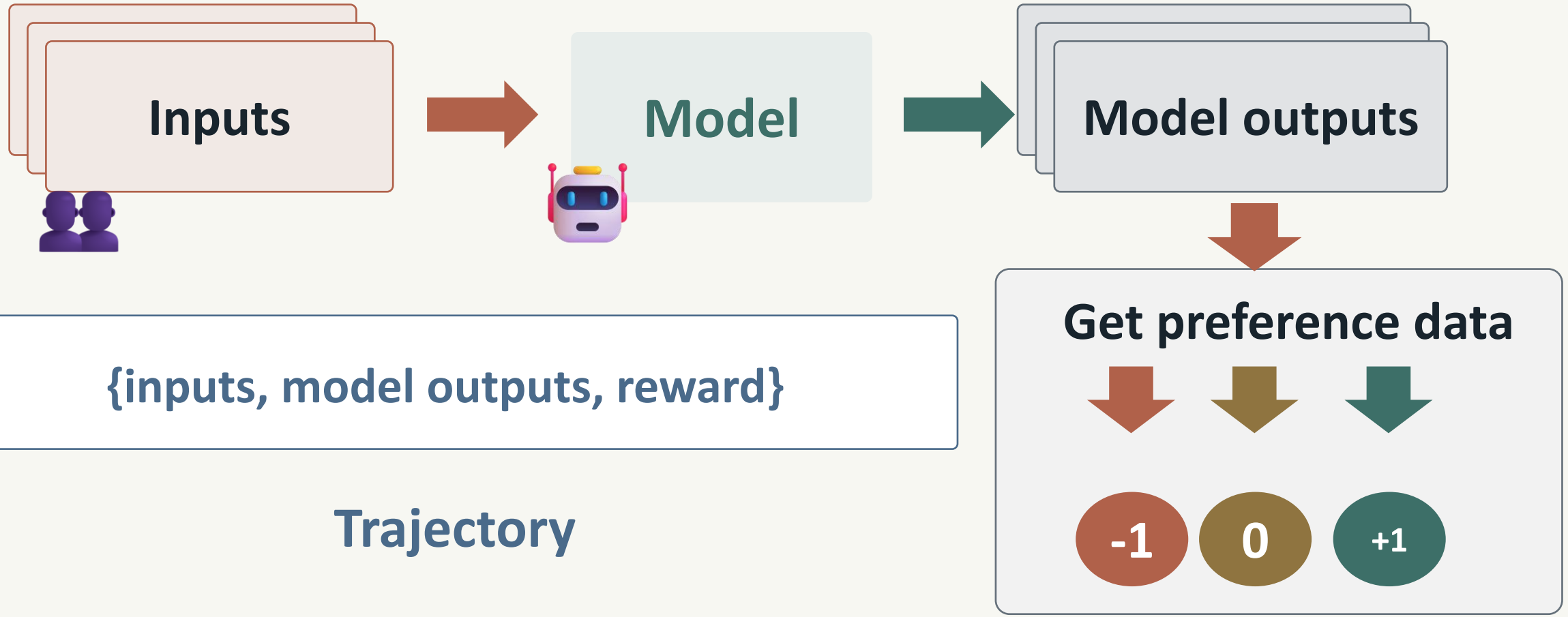


{ inputs, model outputs }

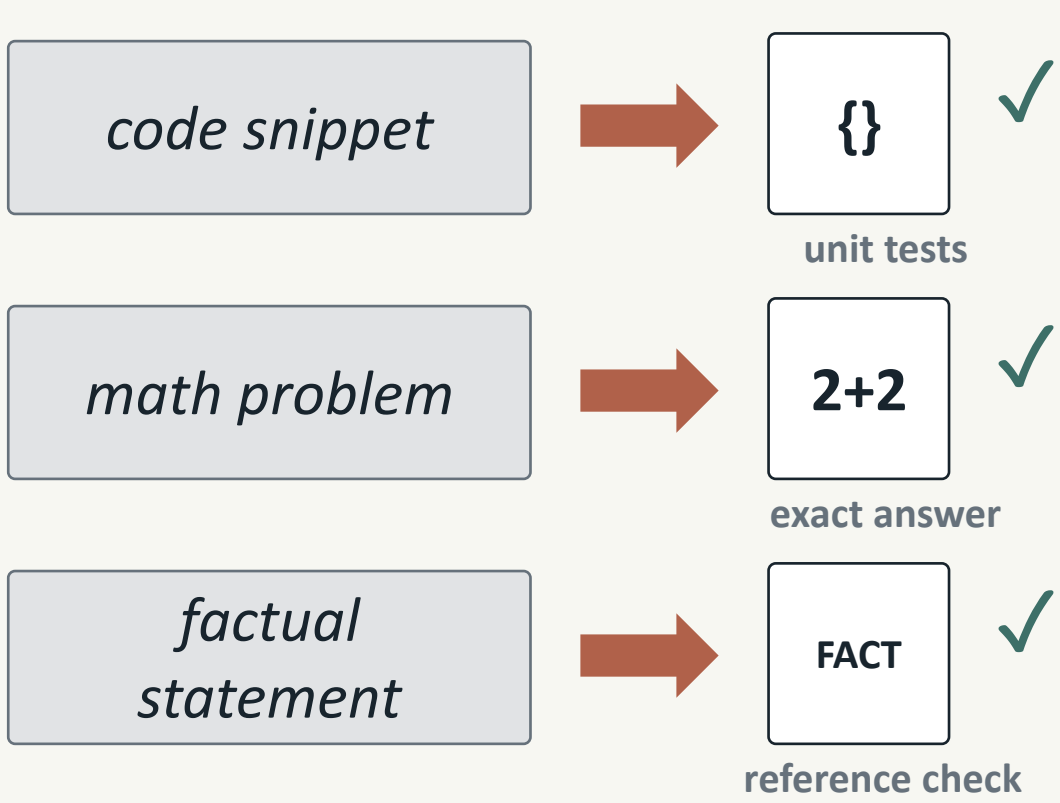
Rollout

Upon completion, each input and its model output form one training sample.

Step 2: Get preference data



Verifiers: objective checks



Verifier pipeline

1. Define objective metrics
2. Implement measurable checks
3. Generate verifiable signals

reward = check(output)

Ranking models learn human preferences

Write a thank-you note for a gift.

LLM	Human rank
Thx	3
Thank you so much for the lovely book! I cannot wait to read it.	1
I guess it was okay. Thanks, I guess.	2

Ranking-model pipeline

1. **Collect human rankings**
2. **Train a ranking model**
3. **Score new model outputs**

reward = learned score

Ranking models vs. verifiers

RM Ranking model

Best for:

1. Aligning with human values / preferences
2. Improving general dialogue quality
3. Scaling feedback efficiently

VR Verifiers

Best for:

1. Ensuring factual correctness
2. Solving math or coding tasks

Step 3: Turn rewards into weight updates

1. Reward R_t
2. Estimate Advantage A_t
3. Compute policy loss from token log-probs
4. Update $\theta_t \rightarrow \theta_{t+1}$

Advantage: was the answer better than expected?

$$A_t = R_t - B_t$$ *reward – baseline*

Advantage	Meaning	Training effect
A > 0	Better than expected	Increase probability
A < 0	Worse than expected	Decrease probability
A ≈ 0	As expected	Almost no change

TWO POPULAR METHODS TODAY

PPO
baseline from a learned Critic

GRPO
baseline from the group mean

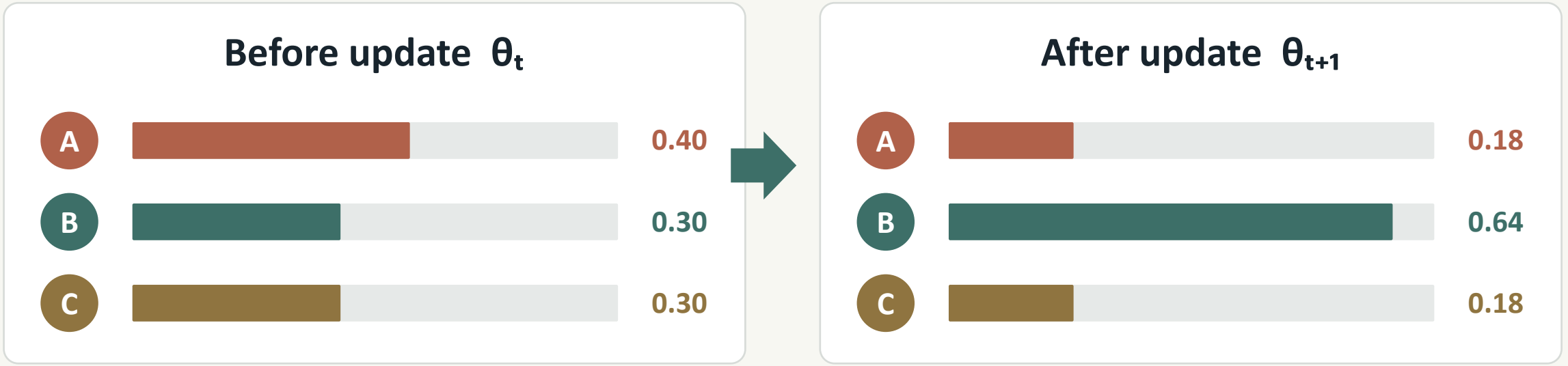
PPO — Schulman et al., "Proximal Policy Optimization Algorithms," arXiv:1707.06347, 2017

GRPO — Shao et al., "DeepSeekMath," arXiv:2402.03300, 2024

One update makes output B more likely

Write a thank-you note for a gift.

B Thank you so much for the lovely book! I cannot wait to read it. A>0



optimizer

Send updated weights θ_{t+1} to generation → sample the next rollouts

In Summary: RL is one loop between two phases

Training

turns data into a better model

gradients · optimizer

$\theta_t \rightarrow \theta_{t+1}$

① **Updated parameters**

full weights, every update

every iteration

② **Generated data**

many small rollout records

Generation

turns the model into new data

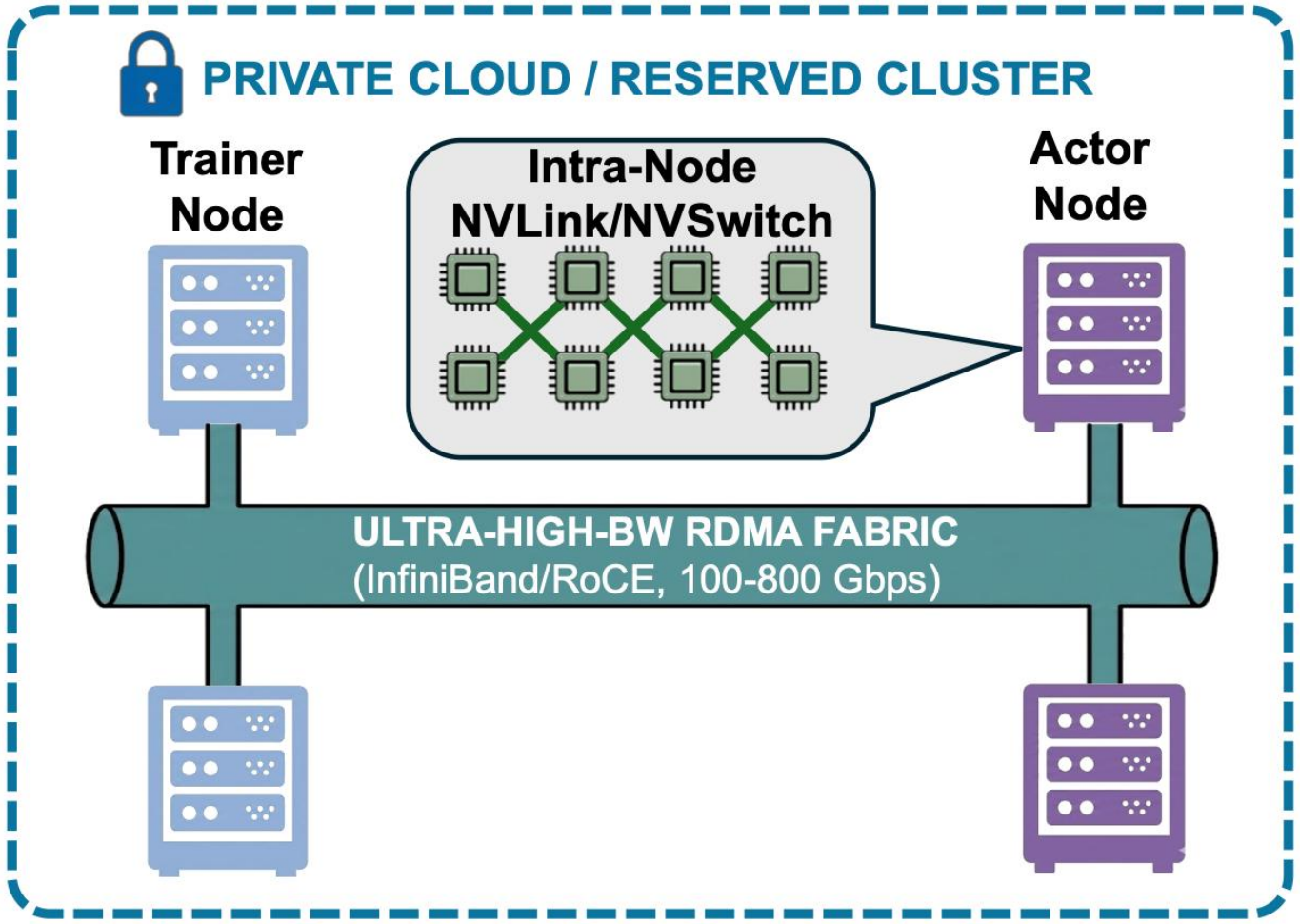
prompts → responses

rollouts + rewards

Model	8B	32B	70B
BF16 weights	16 GB	64 GB	140 GB

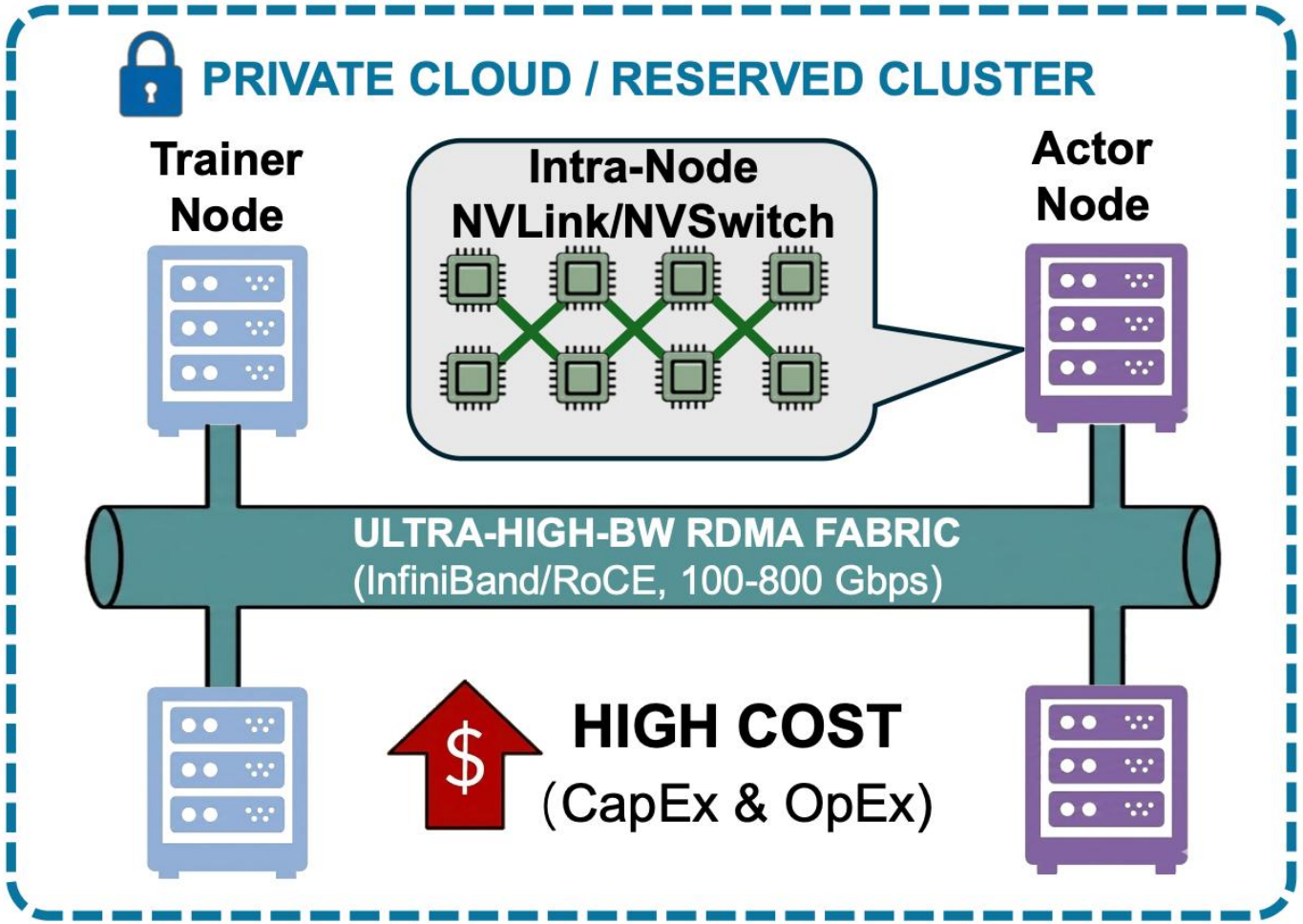
RL system and Challenge

Most RL systems assume one RDMA island



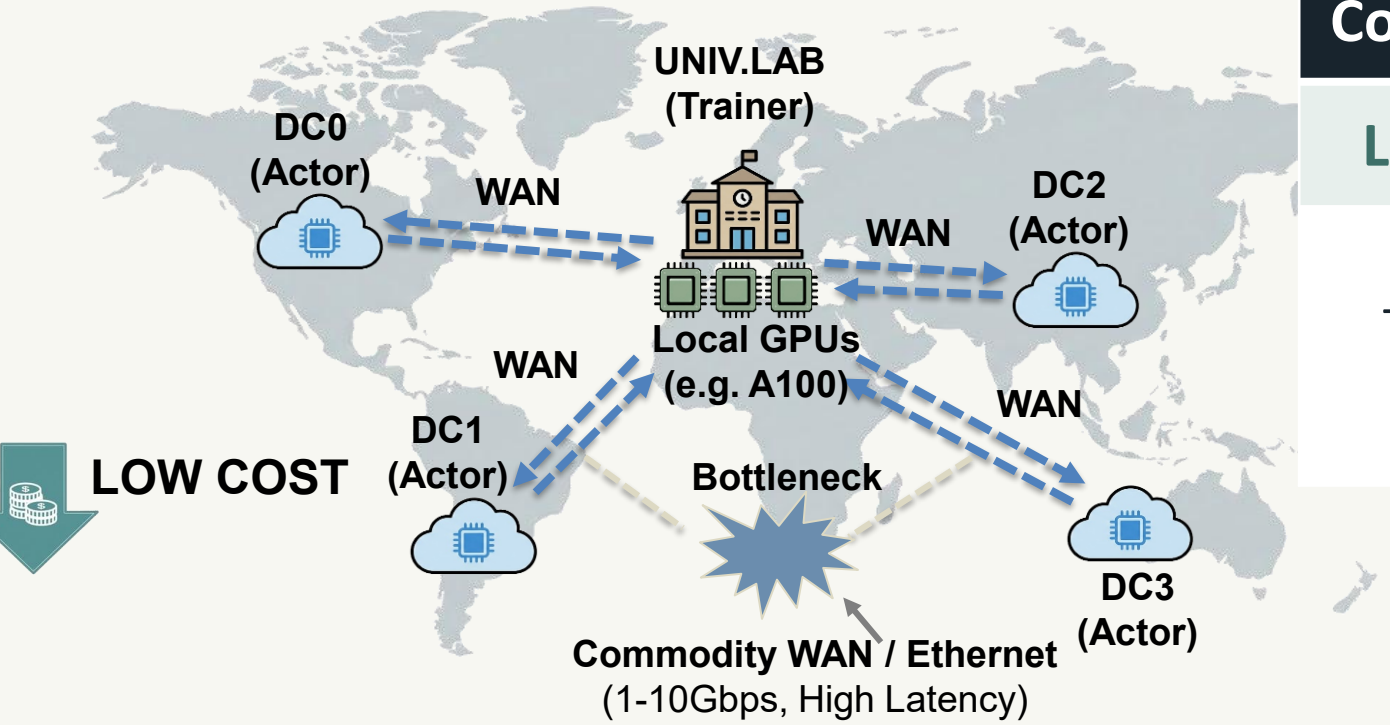
BF16 weights	100 Gbps ideal
16 GB	1.3 s
64 GB	5.1 s
140 GB	11.2 s

Most RL systems assume an RDMA-connected HPC cluster



Configuration (Provider)	$/hr	Min.	BW
Tightly coupled (RDMA fabric, reserved)			
2×8×H100 · EFA (AWS)	**32**	**24 hr**	3.2 Tbps
2×8×H100 · IB (Lambda)	**38**	**1 wk**	3.2 Tbps

Opportunity: Decentralized GPU Pools



Configuration (Provider)	$/hr	Min.	BW
Loosely coupled (cross-cloud, on-demand)			
8×H100 (Hyperbolic) + 8×H100 (Hyperstack)	**27**	**1 hr**	1 Gbps
8×H100 (Hyperbolic) + 8×A100 (Lambda)	**26**	**1 hr**	1 Gbps

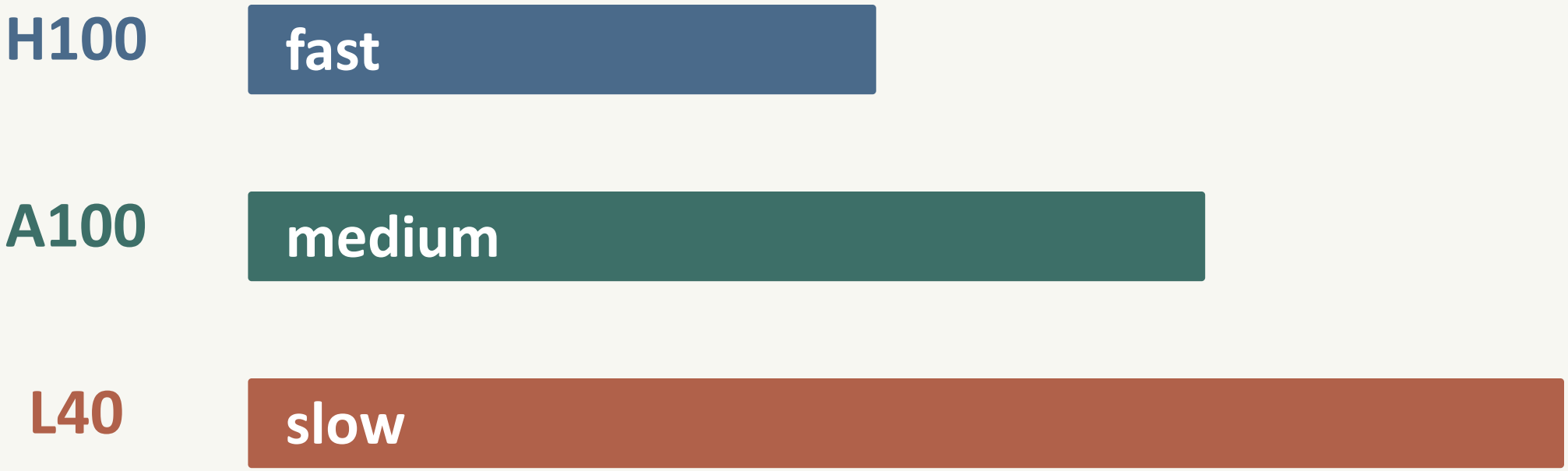
Cross-cloud: low-cost WAN pool. Spot: provider-specific and subject to preemption.

C1: the commodity network barrier

1.3 s	45 s	128 s
RDMA	Rollout window	1 Gbps WAN

WAN transfer outlasts generation.

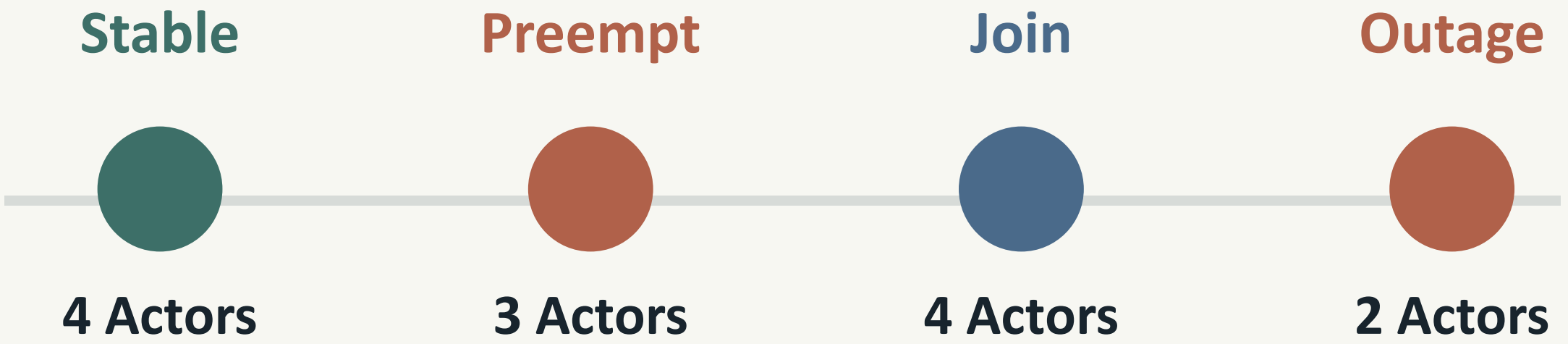
C2: no uniform Actor



Equal batches create idle time and stragglers.

Challenge

C3: the pool changes mid-run



No global pause. No stale rollouts.

THE AURORARL DESIGN TARGET

C1
Weak links

C2
Heterogeneity

C3
Churn

NEXT: AURORARL

In summary

What we covered

1. **The goal of post-training** is to turn a next-token predictor into a model that behaves usefully and follows our intent.

2. **RL is a loop:** generation produces rollouts and rewards, while training updates the model and sends the new parameters back.

3. **Decentralized GPU pools create an opportunity for RL:** cheaper, flexible GPUs across clouds, connected over the WAN.

Where the previous act left us

C1 · The commodity network barrier

1.3 s on RDMA · **45 s** rollout window · **128 s** on 1 Gbps WAN

WAN transfer outlasts generation.

C2 · No uniform Actor

H100, A100 and L40 in one pool, 2-3× apart.

C3 · The pool changes mid-run

Actors preempted, regions dropped, nodes rejoining.

Three barriers. One property answers all three.

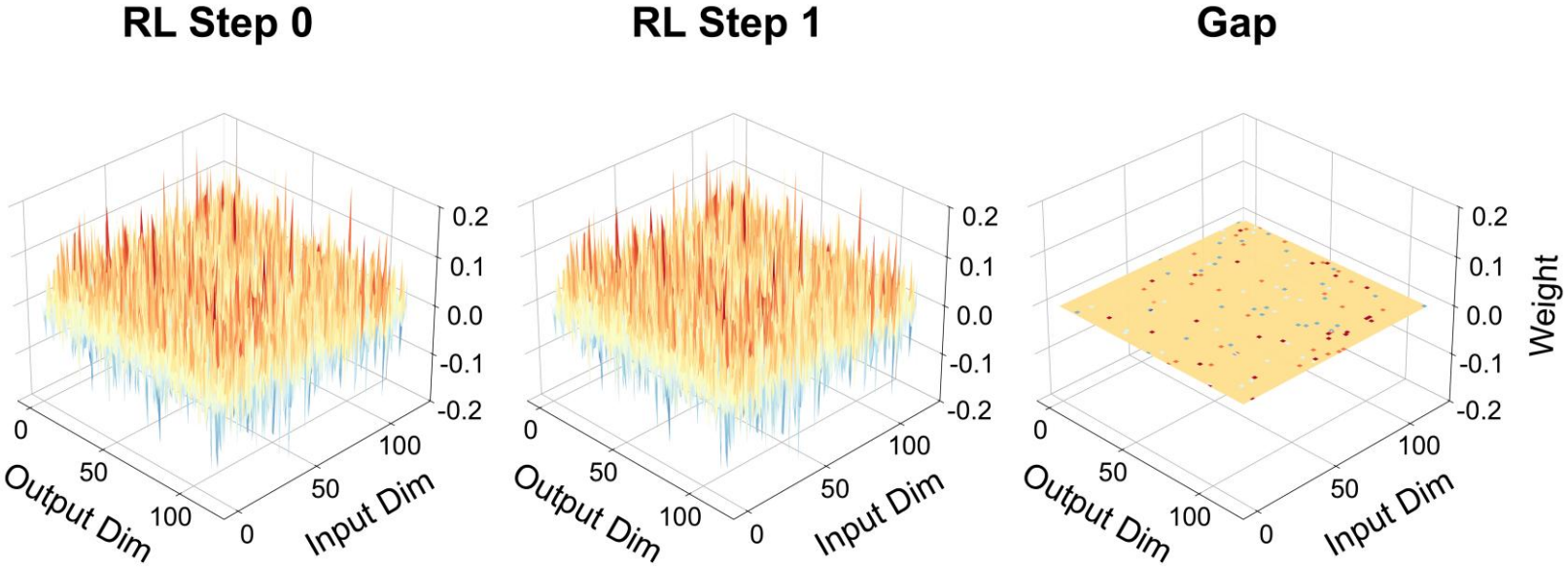
ACT 1

The property we exploit

RL post-training updates almost nothing

Key insight

One RL step changes almost nothing



~1 %
elements move

100 %
tensors touched

1e-5
typical delta

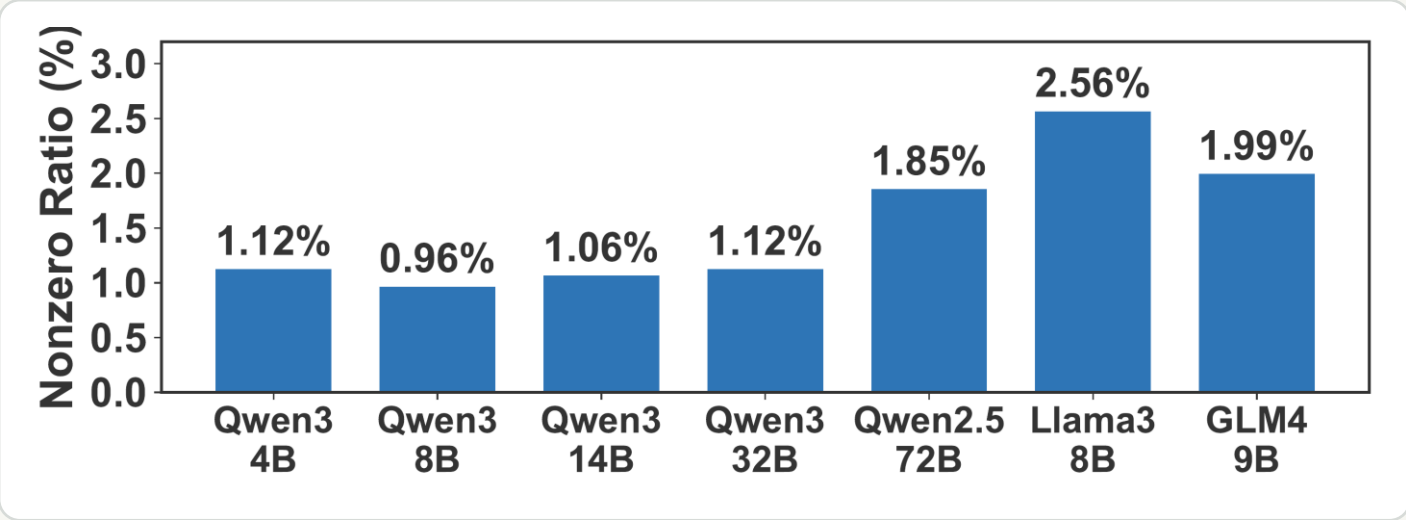
Fine-grained sparsity, not layer sparsity.

Same tensor before and after one GRPO step; right panel plots the element-wise difference.

Key insight

The ratio holds across models and algorithms

Across model families



Across RL algorithms

Algorithm	Nonzero
GRPO	0.96 %
RLOO	0.93 %
OPO	1.06 %

Measured on Qwen3-8B under identical hyper-parameters.

Seven models, three algorithms, all under 3%.

Nonzero ratio = fraction of scalar parameters with a non-zero BF16 delta after one optimizer step.

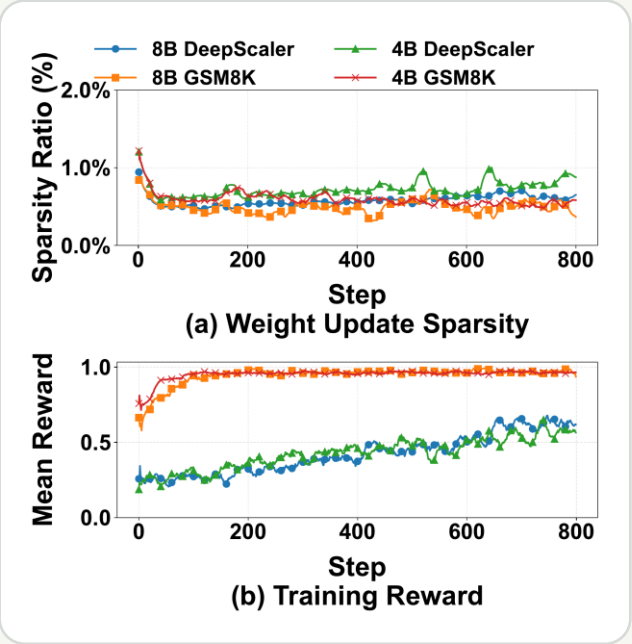
Key insight

Why the updates are this sparse

Three structural causes

1. **Small steps** — Post-training LR ~ 1e-6, two orders below pretraining.
2. **Regularized** — KL penalty and gradient clipping bound each update.
3. **Rounded away** — Some updates are too small to represent in BF16.

Stable across 800 steps



A persistent property of RL, not a warm-up artifact.

Key insight

What that buys, in bytes

Full weight broadcast

15.6 GB

per Actor, per step

79 ×

lossless

Sparse delta checkpoint

202 MB

per Actor, per step

idx[] + val[] · bit-exact BF16 · no error feedback

Same update. Same bits. 79x less wire.

ACT 2

The system

Two tiers, five stages, one invariant

Three RL properties, three levers

Sparsity

Per-step updates touch ~1% of elements

→ **Compress the transfer** — C1

Decomposability

A delta is an unordered set of scatter-adds

→ **Stream it and overlap it** — C1

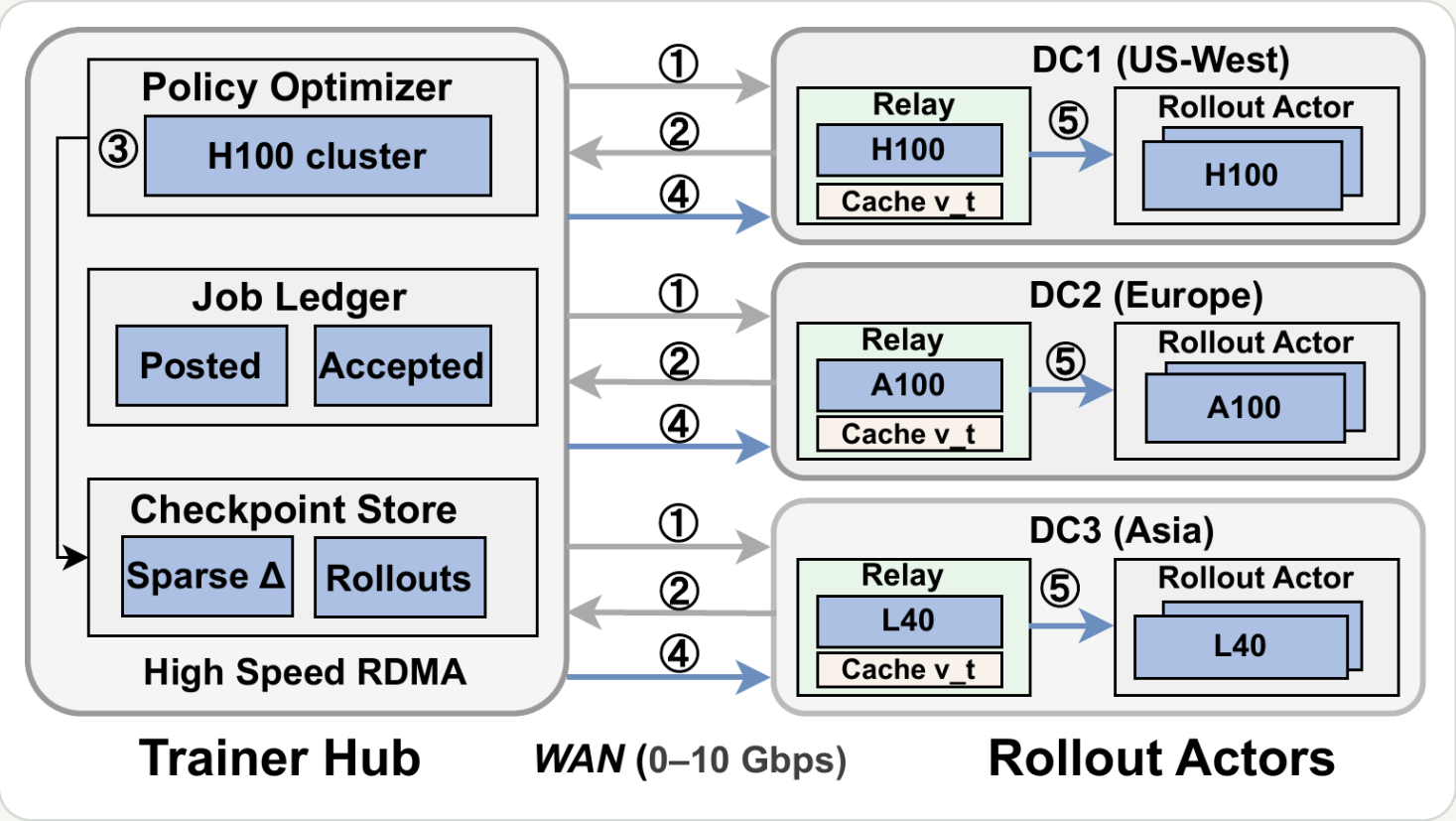
Independence

Rollout tasks need no coordination

→ **Schedule and scale elastically** — C2 · C3

White-box co-design, not a generic compressor.

Two tiers: Trainer Hub and Rollout Actors



Trainer Hub

Optimizer · Job Ledger · Checkpoint Store

Relay

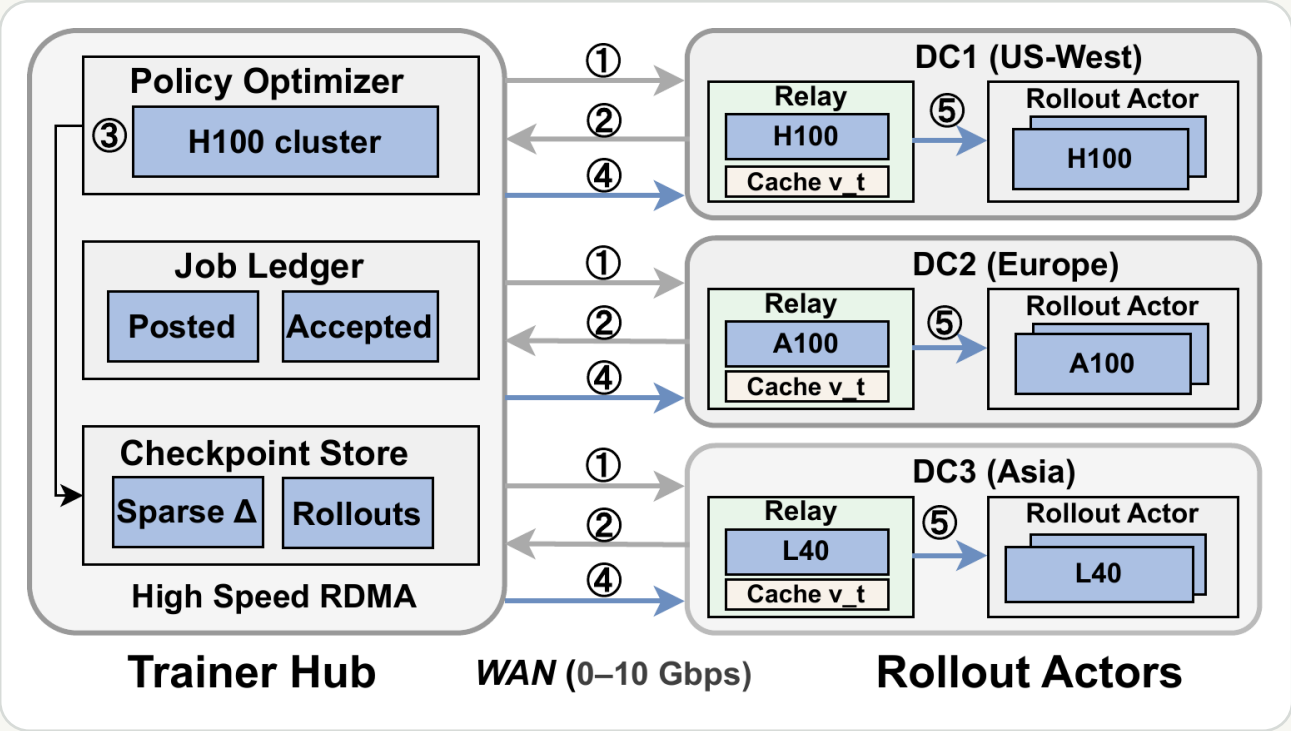
One per region; caches v, fans out locally

Rollout Actor

Claims prompts, generates, returns version

One hub, one relay per region. The numbers trace one iteration.

One iteration, five stages, one invariant



1. **Prompts** — Ledger to Actors
2. **Rollouts** — Actors to Store
3. **Update** — Optimizer step
4. **Delta** — Hub to one Relay
5. **Fanout** — Relay to peers

Stages 4-5 overlap stage 2 of the next iteration

The invariant: one-step policy lag eligible(a, v) = a.ver in { v, v-1 }

Three stages cross the WAN. Only stage 4 carries real bytes.

ACT 3

Design

Three mechanisms, one per barrier

The checkpoint is the wire format

1 One artifact

Each step emits one immutable, versioned, hashed delta. Transfer is replication of that file.

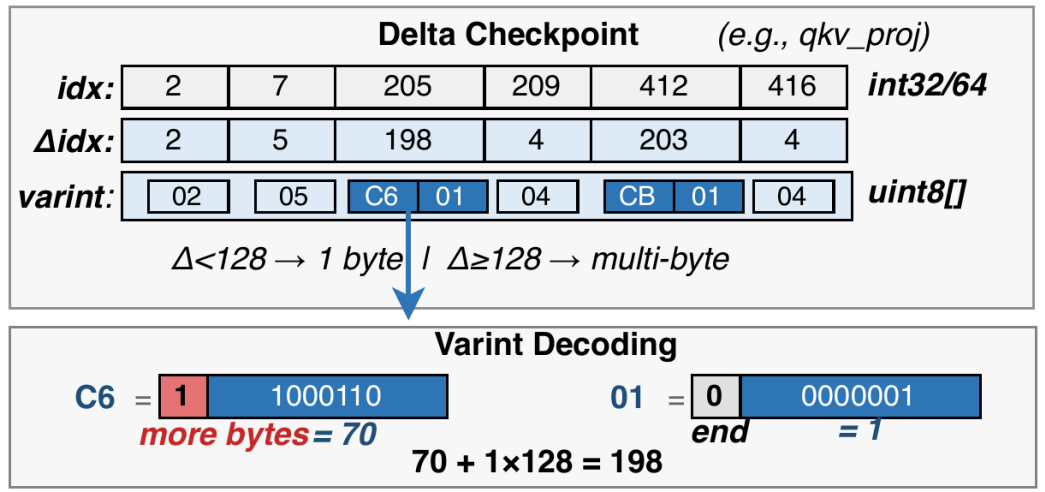
2 Fused for vLLM

Flatten to idx[] and val[]. Q,K,V become qkv_proj. The Actor applies a flat scatter-add.

3 Lossless

Values stay bit-exact BF16. No top-k, no thresholding, no error-feedback buffer.

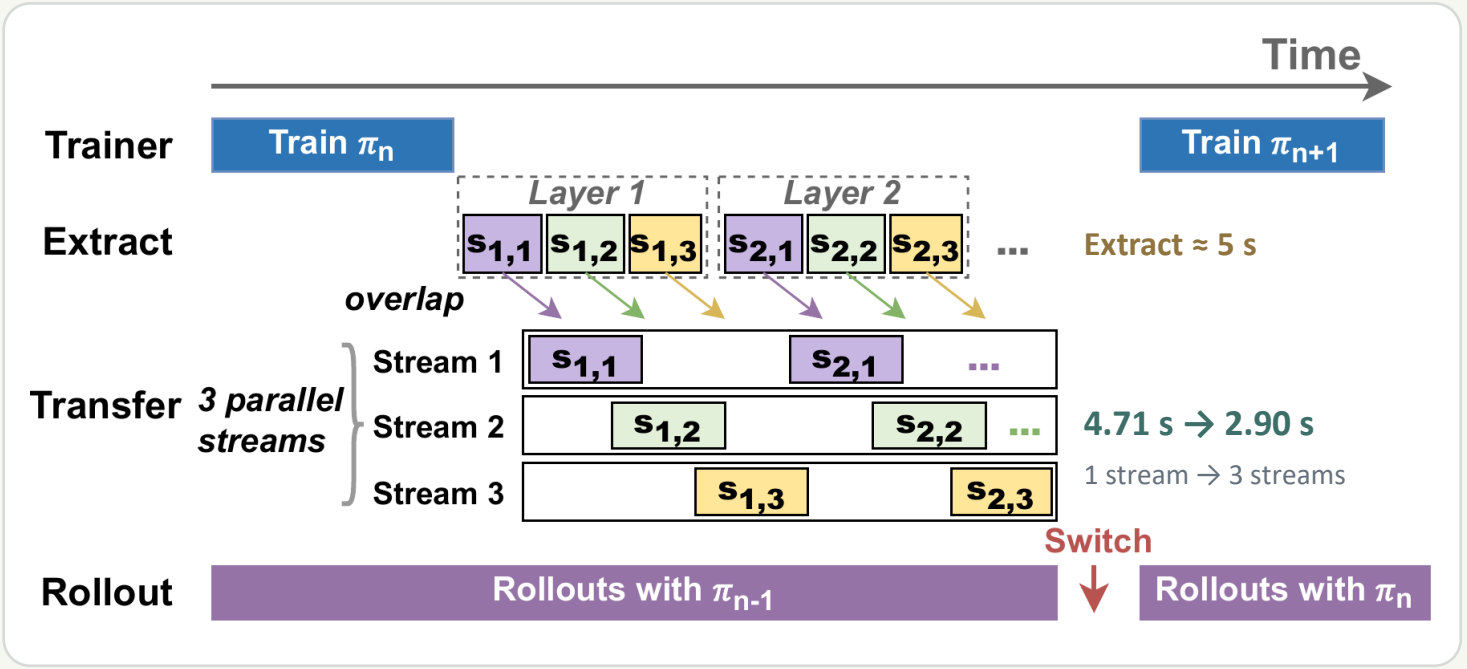
Index encoding



Payload per step	Index cost	Checkpoint size
15.6 GB → 202 MB	< 2 bytes each	-30 to -50 %

One file. Fused for the engine. Bit-exact.

Overlap the transfer with generation



Cut-through
Emit each segment as soon as it is encoded

Round-robin
Stripe across S streams; one stall blocks one

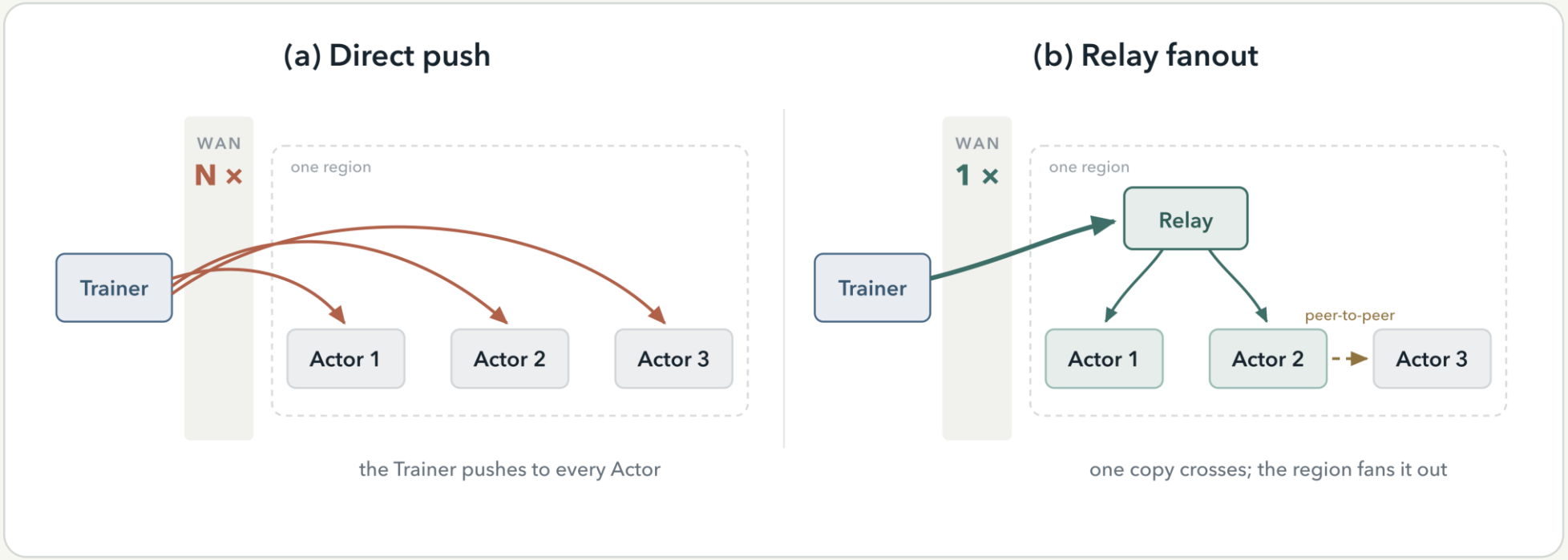
Non-blocking
Rollouts continue on π(v) until commit

The per-step deadline

Extract	Transfer, 1 stream	Transfer, multi-stream	Rollout window
~5 s	4.71 s	2.90 s	45 s

Staging D(v+1) must finish before the current batch on π(v) does.

Relays collapse cross-region fanout



Direct push

N × |Δ| egress

Relay fanout

1 × |Δ| per region

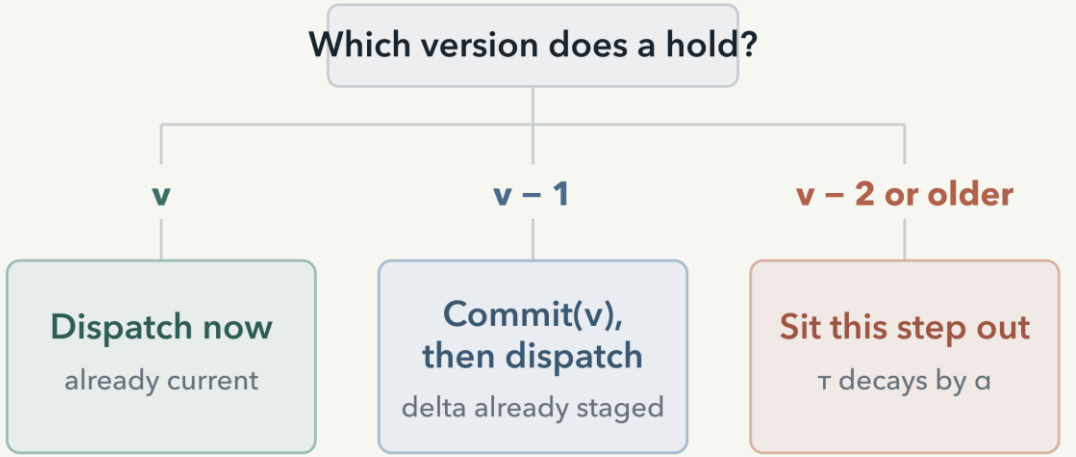
Peers relay onward based on round-trip time and the version they already hold.

Work in proportion to measured speed

Batch of 300 requests

One signal absorbs throttling, congestion and contention.

Eligibility is a version predicate



One test, every step. Membership never enters into it.

Everyone finishes at about the same time.

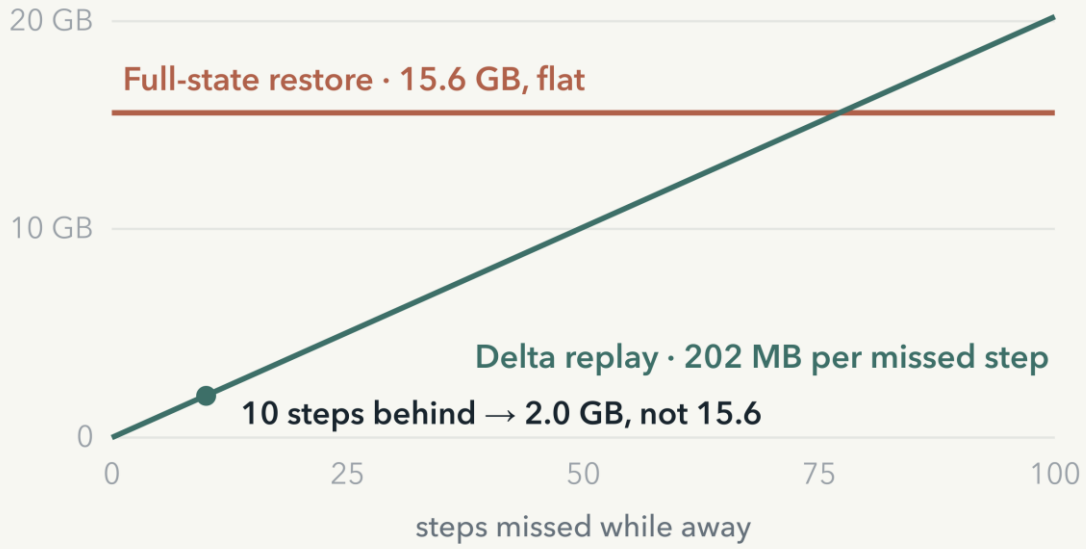
Leases, and catch-up by missed steps

Per-prompt leases

1. **Independent** — Only the dead Actor's prompts expire back to the pool.
2. **Non-blocking** — The Trainer aggregates whatever arrives in time.

One check at settlement: lease, version, hash. The only correctness mechanism in the data plane.

Catch-up cost



Spot preemption needs no extra machinery.

ACT 4

Evaluation

Real cross-cloud WAN, not an emulator

Setup

Testbed

1. **Trainer**
   US, 2/4/6 × H100 SXM5, FSDP2
2. **Actors**
   Canada, 4/8/12 × A100 PCIe, vLLM
3. **Link**
   Cross-cloud WAN, 0.5 - 1 Gbps

Workload

1. **Models**
   Qwen3 4B / 8B / 14B
2. **Algorithm**
   GRPO, batch 512, one-step lag
3. **Data**
   Hendrycks MATH, GSM8K, DeepScaleR

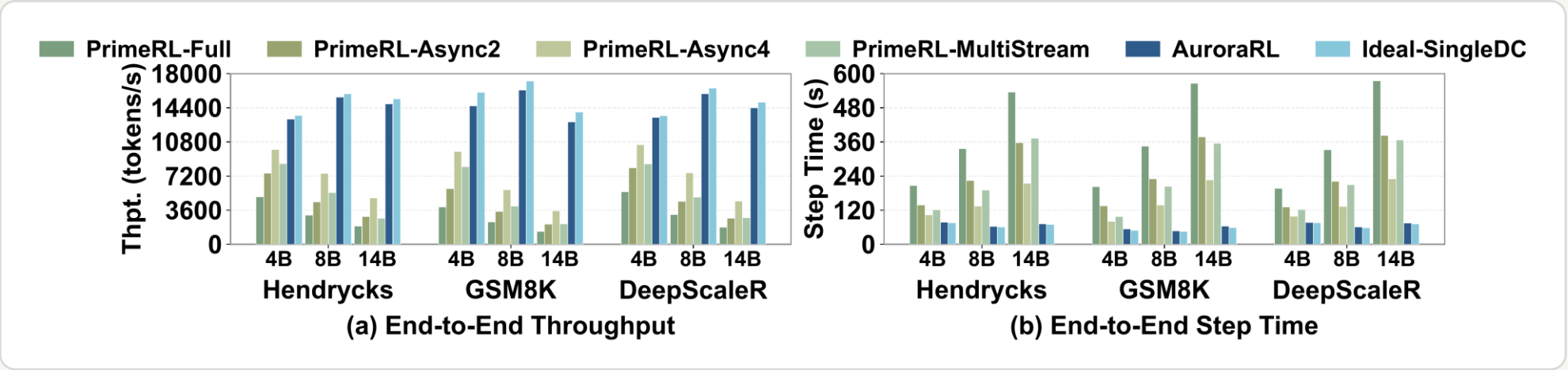
Baselines

1. **PrimeRL-Full**
   Dense broadcast, same one-step bound
2. **Async2 / Async4**
   Overlap bought with 2 and 4 step lag
3. **Ideal-SingleDC**
   800 Gbps RDMA upper bound

~5K lines of Python on PrimeRL + FSDP2 + vLLM

Ideal-SingleDC replaces only the cross-cloud transfer cost in AuroraRL's measured traces; training and rollout execution are unchanged.

Faster end to end, at no accuracy cost



Speed

vs PrimeRL-Full	2.4 - 9.5 ×
vs Async2 / Async4	1.3 - 6.1 ×
vs MultiStream	1.5 - 6.0 ×
Gap to Ideal-SingleDC	1.31 - 8.91 %

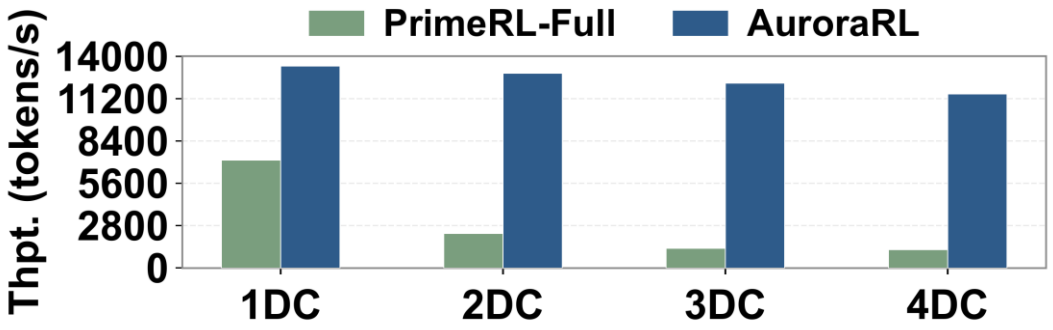
Accuracy after 800 RL steps

Benchmark	Aurora	Full	Async4
Math500 P@1	0.906	0.904	0.890
AIME24 P@16	0.727	0.728	0.630

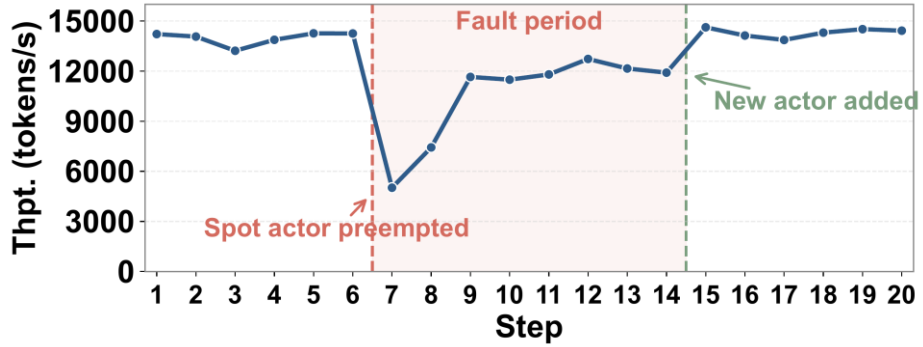
Near-RDMA speed, with the one-step bound intact.

Scaling out, and surviving churn

As the map spreads: 1 → 4 regions



When a spot Actor vanishes



Also measured

Five steps, end to end	Encoding + multi-stream	Delta at 10 Gbps	Mixed A100 + L40 pool
15m48s → 5m09s	3.2 × faster	0.25 s ≈ RDMA	+35.5 %

Regions and preemption cost throughput, never correctness.

Takeaway

Better tokens per dollar than reserved RDMA

Model · Method	$ / hr	tokens / $ vs RDMA
Qwen3-8B · AuroraRL	15.88	1.21 ×
Qwen3-8B · SingleDC	19.92	1.00 ×
Qwen3-14B · AuroraRL	23.82	1.59 ×
Qwen3-14B · SingleDC	39.84	1.00 ×

Within ~5% of the throughput, well under the cost.

RDMA multi-day minimum commitments amortized to hourly rates, an assumption that favours the SingleDC baseline.

AURORARL IN ONE FRAME

C1

Weak links

Sparse lossless deltas, streamed and relayed

C2

Heterogeneity

Capacity-proportional batches, version-gated

C3

Churn

Per-prompt leases and delta-chain replay

Frontier RL post-training on commodity links

202 MB	2.4-9.5 ×	< 8.91 %	1.21-1.59 ×
per step	vs broadcast	gap to RDMA	tokens / $

THANK YOU