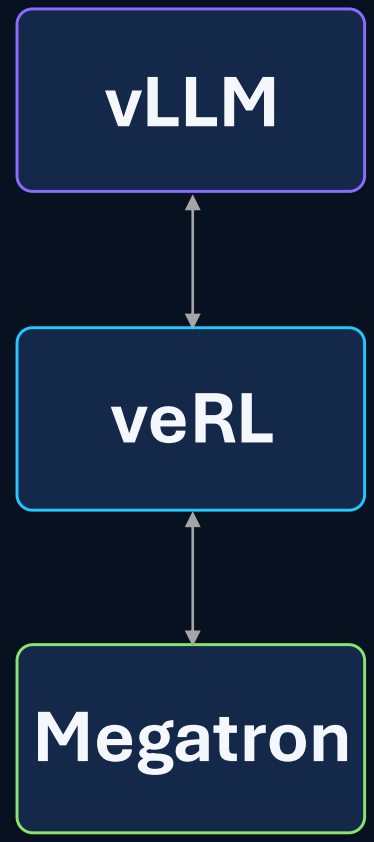
PyTorch CONFERENCE EUROPE 2026

Optimizing Reinforcement Learning at Trillion-Parameter Scale

Songlin Jiang
Aalto University & Mind Lab

hollowman@opensuse.org

April 7, 2026



About me

- Doctoral researcher in AI infrastructure
- Passionate free software developer
- Maintainer of VeRL
- Contributor to vLLM and Megatron-Bridge

Songlin Jiang

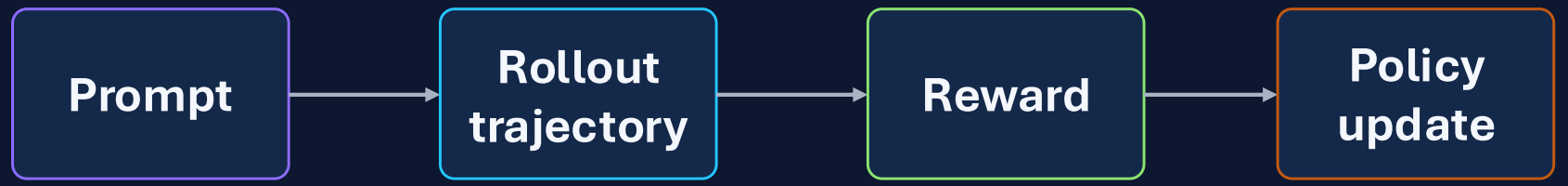
@HollowMan6

@songlin-jiang

@HollowM186

Why use Reinforcement Learning (RL) after pretraining

Pretraining teaches next-token prediction,
RL teaches the model to prefer behaviors that get a better outcome

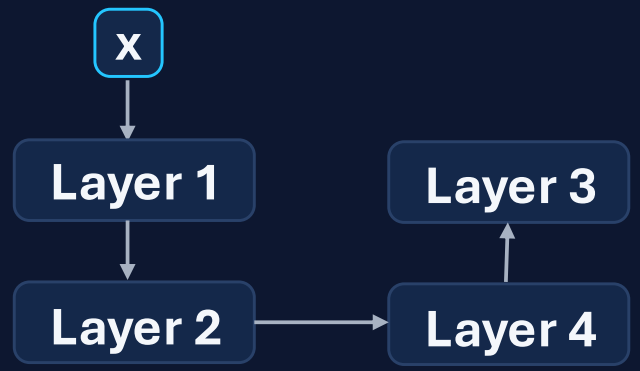


- Better multi-step reasoning
- Task-specific feedback
- Long-horizon behavior shaping

What is a Mixture-of-Experts model

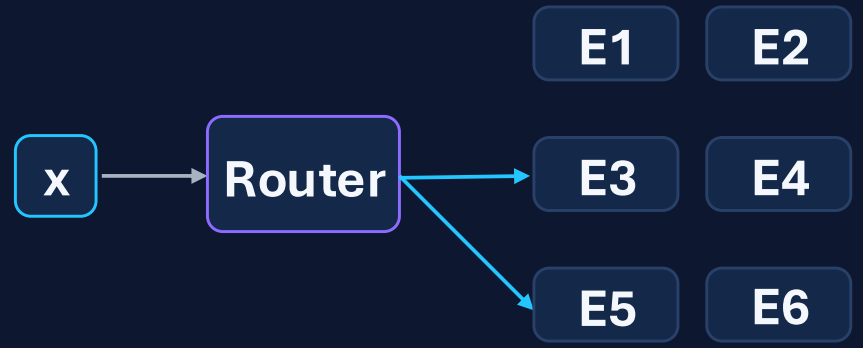
Dense model

Every token passes through every layer



MoE model

The router selects only a few experts for each token



high capacity

sparse compute

MoE-based LLM at trillion-parameter scale

Target model: moonshotai/Kimi-K2-Thinking

1.04T	32.6B	384
total parameters	active parameters	total experts

Need 288 A100 (80GB) GPUs at least
256 for training
32 for inference

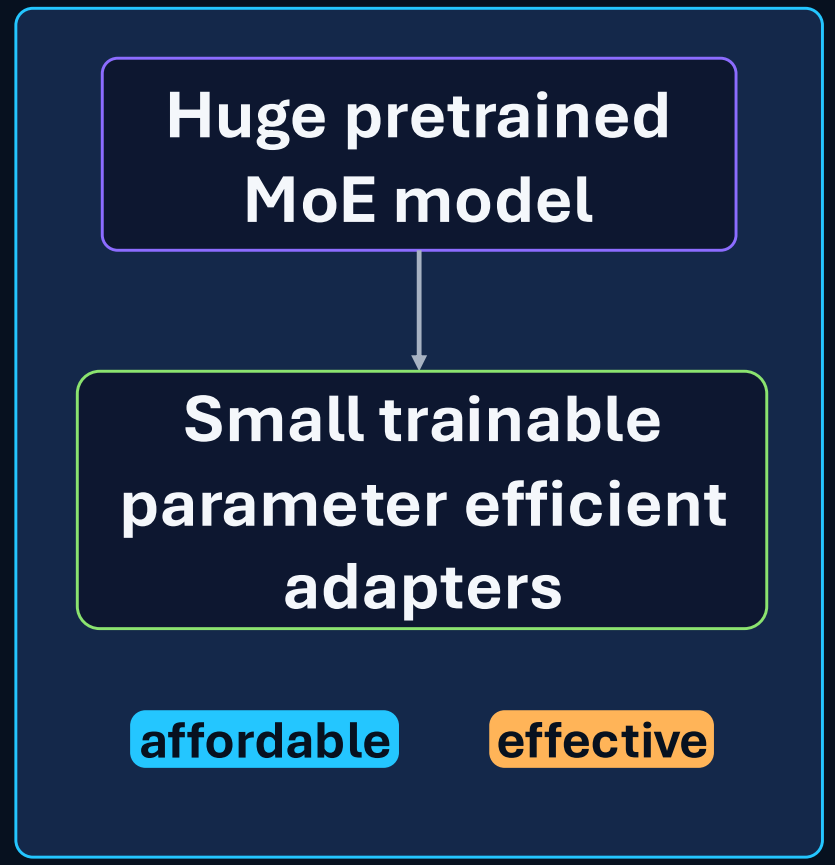
with BF16 full parameter, no offload, context length=4k

TP=64, PP=2 with last stage layers=31, EP=128, ETP=1

TP=32

Question

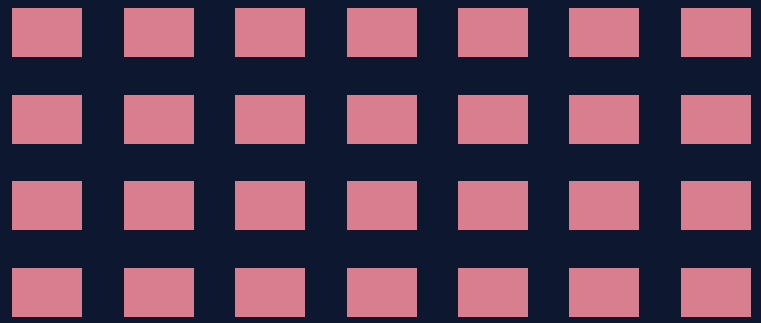
Can we improve Kimi-K2 using reinforcement learning without updating the whole model?



Full-parameter VS Low Rank Adapters (LoRA)

Full-parameter

Every weight gets updated



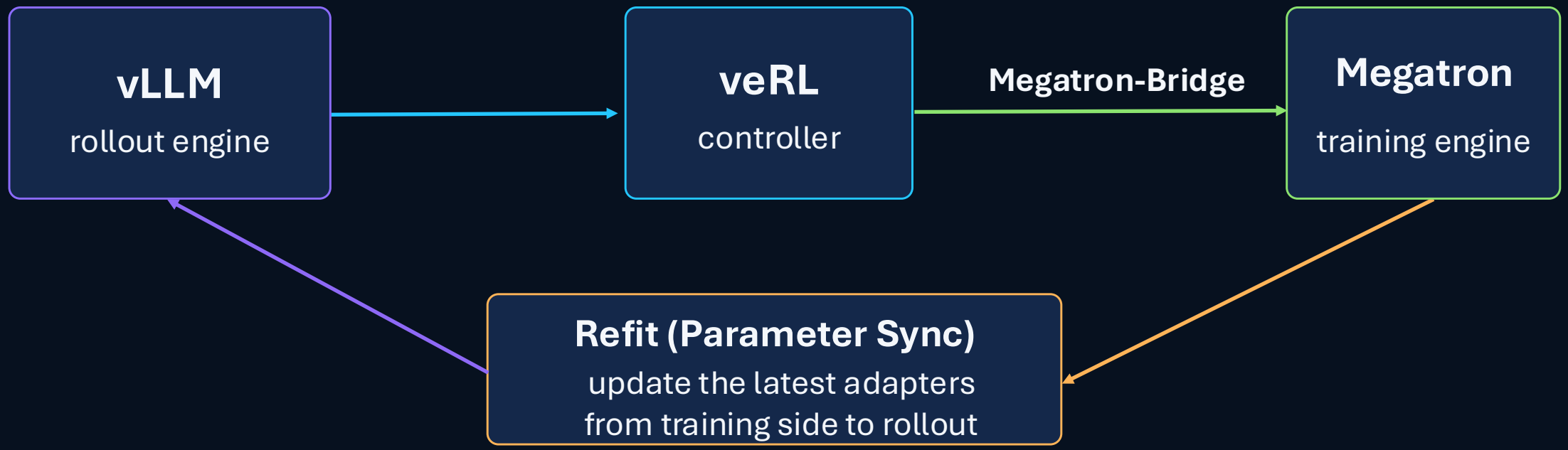
- More optimizer state
- More memory pressure
- Separate reference copy for KL compute

Low Rank Adapters (LoRA)

Freeze the base model

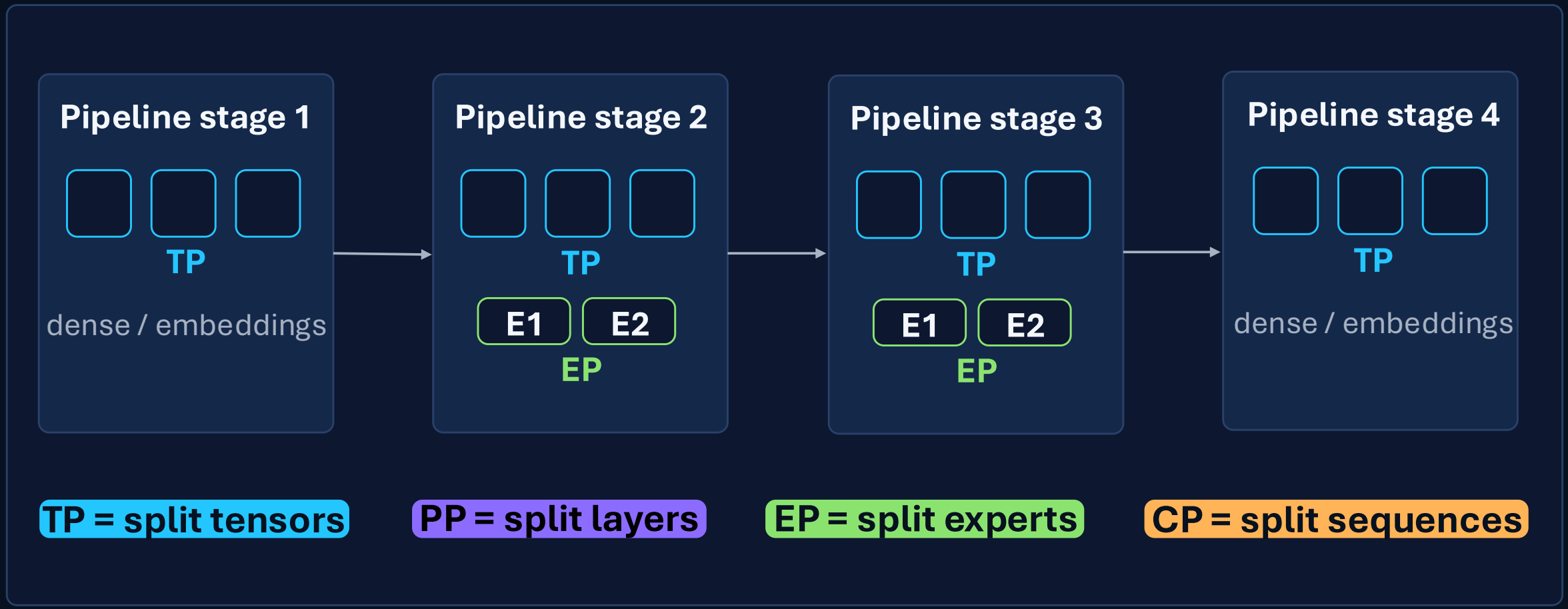
- Much smaller update (32B with r=32 for K2)
- Efficient with a strong prior
- Temporary disable adapters for KL compute

System overview

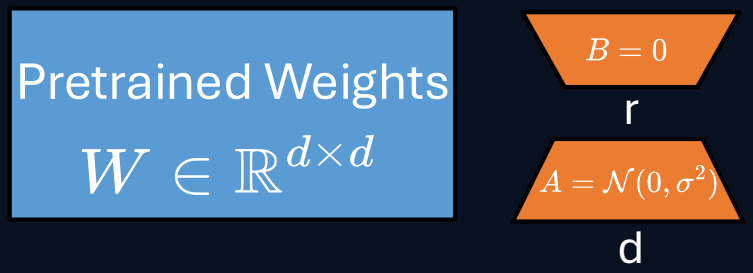


Make LoRA work at scale: hybrid parallelism

At trillion scale, different parts of the model need different sharding rules



Make LoRA work at scale: support for all linear layers



- Shard LoRA for TP:
  - LoRA A: follow base
  - LoRA B: column parallel
- Attach to both dense (attention) + expert part
- All experts shard (EP) of one layer share one adapter

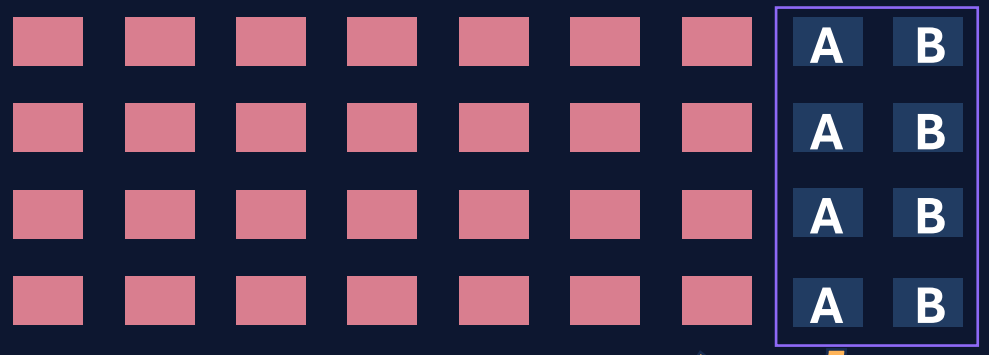
model.layers.2.mlp.experts.63 (3)		
model.layers.2.mlp.experts.63.down_proj.weight	[2 048, 1 408]	BF16
model.layers.2.mlp.experts.63.gate_proj.weight	[1 408, 2 048]	BF16
model.layers.2.mlp.experts.63.up_proj.weight	[1 408, 2 048]	BF16
model.layers.2.mlp.gate (2)		
model.layers.2.mlp.gate.e_score_correction_bias	[64]	BF16
model.layers.2.mlp.gate.weight	[64, 2 048]	BF16
model.layers.2.post_attention_layernorm.weight	[2 048]	BF16
model.layers.2.self_attn (6)		
model.layers.2.self_attn.kv_a_layernorm.weight	[512]	BF16
model.layers.2.self_attn.kv_a_proj_with_mqa.weight	[576, 2 048]	BF16
model.layers.2.self_attn.kv_b_proj.weight	[4 096, 512]	BF16
model.layers.2.self_attn.o_proj.weight	[2 048, 2 048]	BF16
model.layers.2.self_attn.q_proj.weight	[3 072, 2 048]	BF16
model.layers.2.self_attn.rotary_emb.inv_freq	[64]	BF16
model.layers.3 (4)		
model.layers.3.input_layernorm.weight	[2 048]	BF16
model.layers.3.mlp (3)		
model.layers.3.mlp.shared_experts (3)		
model.layers.3.mlp.shared_experts.down_proj.weight	[2 048, 2 816]	BF16
model.layers.3.mlp.shared_experts.gate_proj.weight	[2 816, 2 048]	BF16
model.layers.3.mlp.shared_experts.up_proj.weight	[2 816, 2 048]	BF16
model.layers.3.mlp.experts (64)		
model.layers.3.mlp.experts.0 (3)		

Make LoRA work at scale: refit

Refit is the bridge between training engine and serving engine

LoRA merge weight update

Every weight moves after merge



- ❌ Communication intensive, precision loss
- ✅ No need to enable LoRA on vLLM side
- ✅ Better inference speed and support

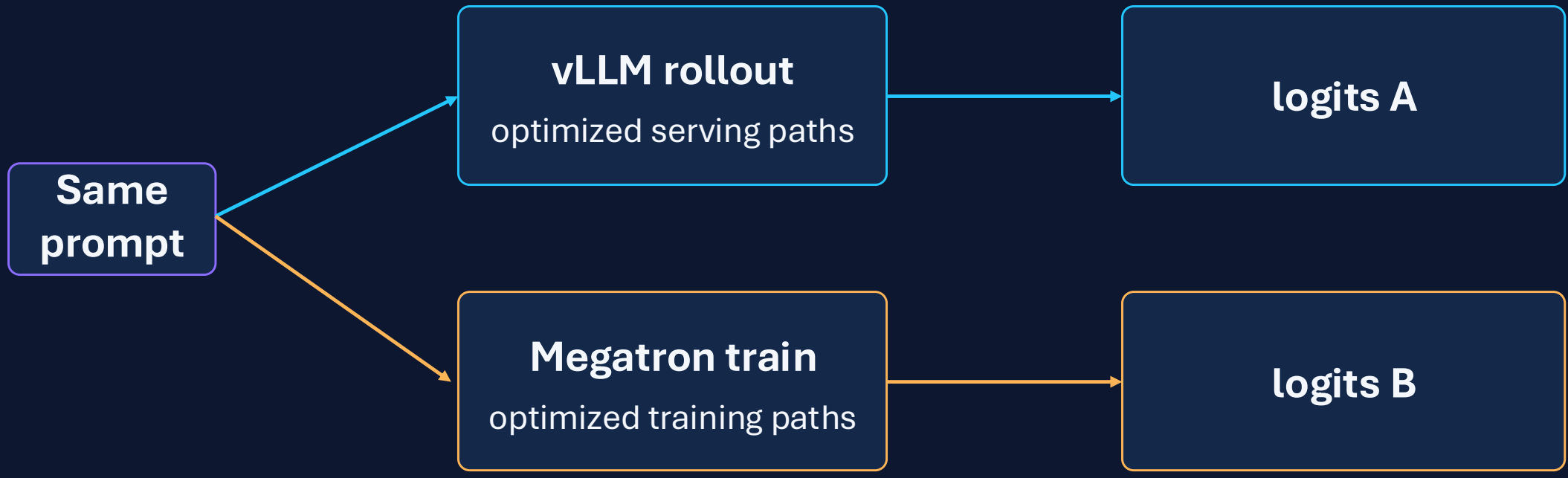
LoRA bridge weight update

Only moves adapter weights

- ✅ Only weight deltas
- ❓ Need model LoRA support on vLLM side
- ❓ Possibly degraded performance

Training inference mismatch

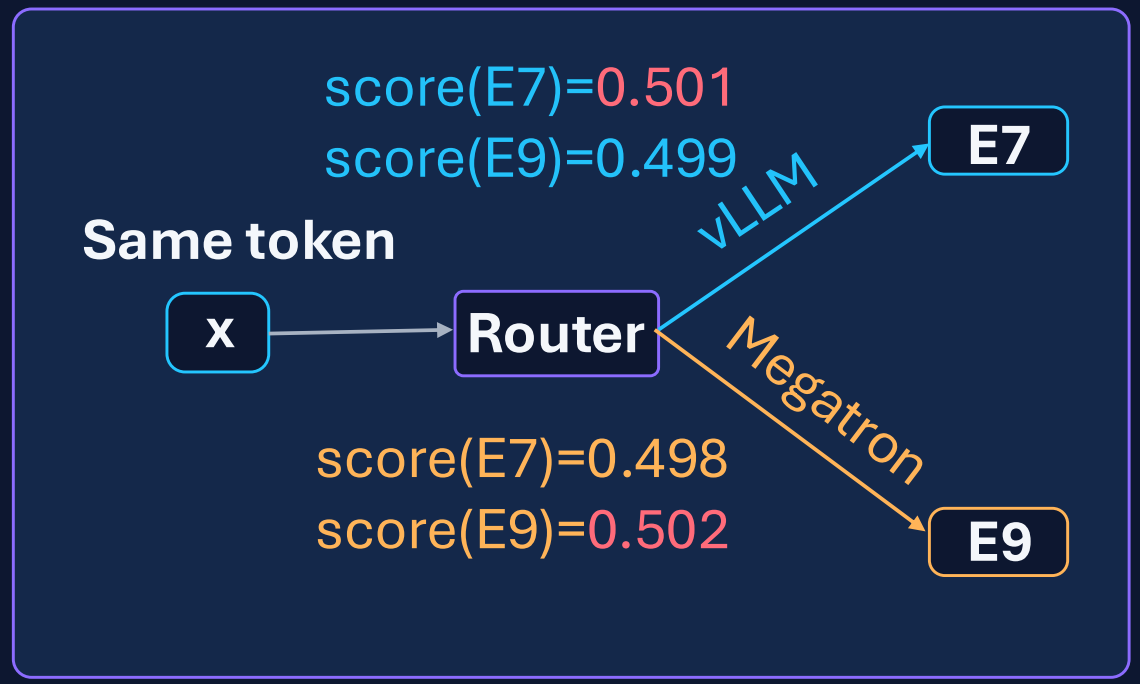
The rollout engine and the training engine are not perfectly identical



Small implementation differences can create a different behavior policy than the one you think you are updating

Why MoE makes mismatch much worse

A tiny router-score change can flip the expert choice entirely



Rollout engine
picks E7

Training engine
picks E9

- Dense part is mostly affected by numerical stability
- MoE router difference can change the computation path
- Small numerical drift in router becomes large behavior drift

Rollout correction

Importance sampling
PPO-side correction

- Reweights the gradient for policy mismatch
- Useful when rollout and training are close

They correct the update

Rejection sampling

- Directly drops tokens or sequences that look too far off-policy
- Useful when mismatch is severe
- But it reduces the effective sample set

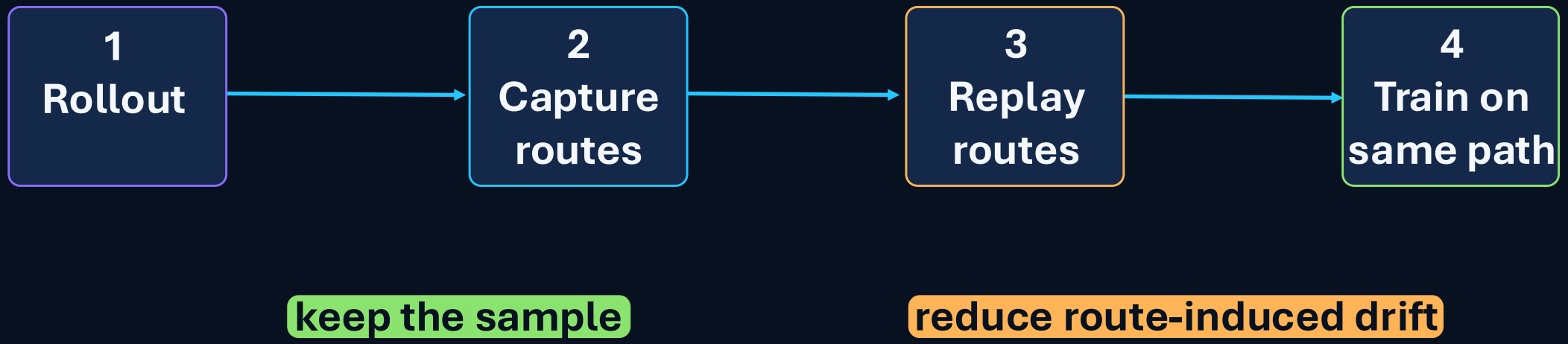
They filter the update

They do not make the router take the same path

Router replay R3: fix the routing, not just the loss

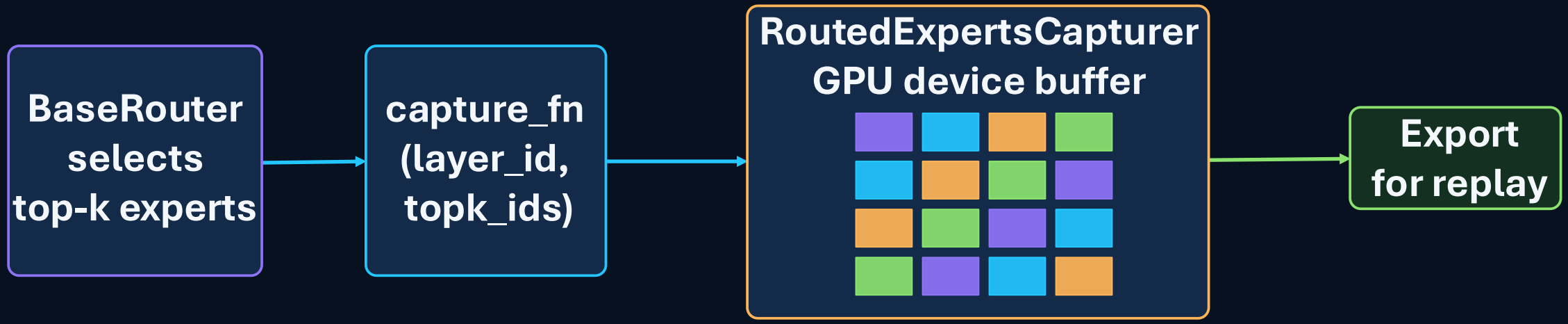
Core idea:

- During rollout, save the routing decisions
- During training, replay those same decisions



Router replay in vLLM: what gets captured

During inference, vLLM captures the routed experts for every MoE token and layer



- The GPU buffer is shaped with [tokens, layers, topk]
- Replay record is expert IDs instead of router logits

Router replay in vLLM: the correctness fixes that mattered

Symptom in practice: training follows completely wrong experts

Bug 1: Expert layers mismatch

- vLLM reports experts ID **across all** transformer layers (including dense)
- Megatron only has routers for MoE layers

Bug 2: Wrong captured IDs under EPLB

- EPLB (Expert Parallelism Load Balancer) duplicates heavily-loaded experts and remaps logical experts to physical GPUs
- Training replay needs logical expert IDs
- Fix: capture logical top-k **before EPLB**

TAKEAWAYS

Three design principles for trillion-scale RL systems

PyTorch CONFERENCE EUROPE 2026

1 Preserve the strongest prior you have

LoRA is much **cheaper** for trillion param model

A small trainable delta on top of a much **stronger model** can be better than full RL on a weaker model

2 Trade-offs between LoRA merge / bridge

LoRA merge is **communication** intensive, but can reduce **inference time**

LoRA bridge is **communication** cheap, but can increase **inference time**

3 Treat routing as part of the policy

For MoE RL, high precision alone is not enough if the expert path changes

Router replay is a first-class stability mechanism

TAKEAWAYS

From software to service infrastructure: MinT

PyTorch CONFERENCE EUROPE 2026

The goal is not only to make trillion-scale RL possible, but also to make it **repeatable**

Users define the loop

- what to train
- what data / experience
- how to learn
- how to evaluate

›

MinT handles the infrastructure

- schedule compute
- run distributed jobs
- manage model state
- recover from failures

Managed Service

post training + RL

Python SDK

from quickstart to scaled jobs

Real Scale

Qwen3 4B → Qwen3 30B A3B → Qwen3 235B A22B → GLM5 744B → Kimi K2 1T

Why it matters

- Abstracts away the operation burden behind large-scale RL
- Keeps the train → eval → deploy loop tight
- Turns one-off systems work into something repeatable

Background | System | Stability | Takeaways

Find out how to use MinT:
mint-doc.macaron.im

Or scan QR code to join Discord:

Thank you!

Useful links

- LoRA Without Regret
  thinkingmachines.ai/blog/lora
- Building trillion-parameter reasoning RL with 10% GPUs
  macaron.im/mindlab/research/building-trillion-parameter-reasoning-rl-with-10-gpus
- Router Replay R3: Why It Failed and How We Fixed It
  macaron.im/mindlab/research/router-replay-r3-why-it-failed-and-how-we-fixed-it
- Defeating Nondeterminism in LLM Inference
  thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference

Questions?

Find out how to use MinT:
mint-doc.macaron.im

Or scan QR code to join Discord: