nsdi '26

RLBoost: Harvesting Preemptible Resources for Cost-Efficient Reinforcement Learning on LLMs

Yongji Wu*, **Xueshen Liu***, Haizhong Zheng, Juncheng Gu, Beidi Chen,
Z. Morley Mao, Arvind Krishnamurthy, Ion Stoica

Google

UNIVERSITY OF MICHIGAN

UC Berkeley

Carnegie Mellon University

RL Unlocks Advanced Capabilities on LLMs

Math

Code

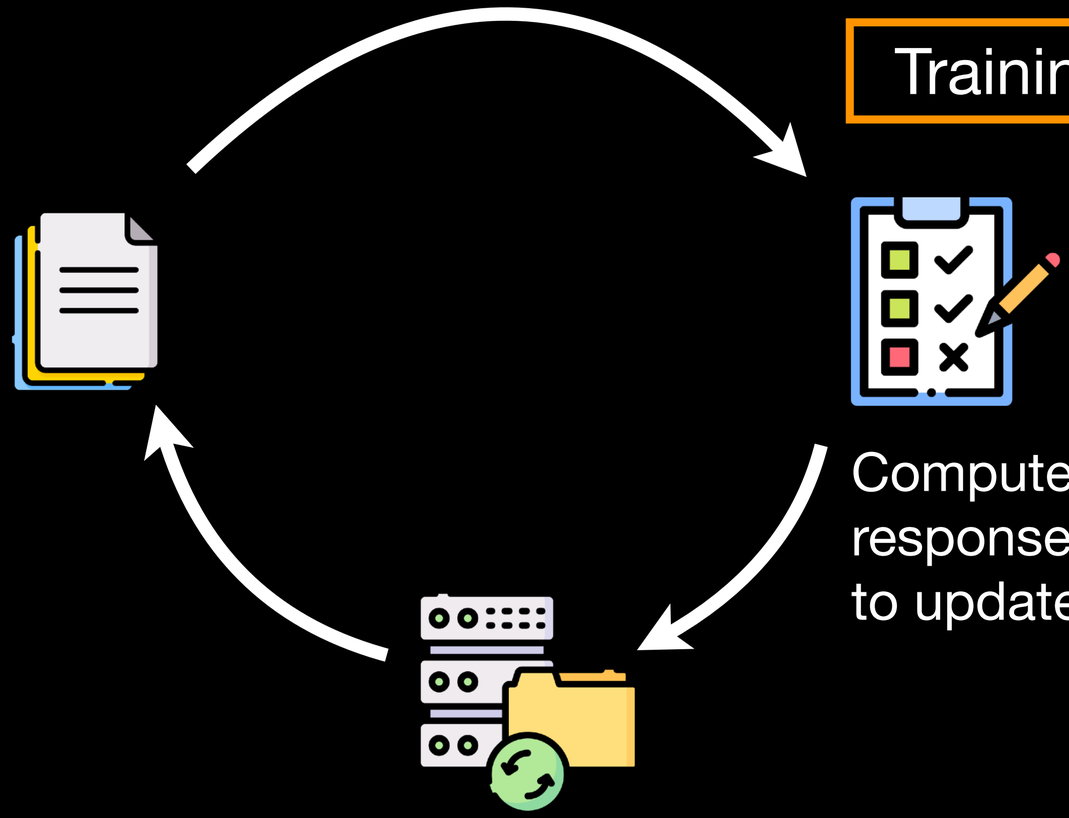
Tool Use

Cycle of RL on LLMs

SGL vLLM

Rollout

LLM randomly generates multiple responses for each question.



Training

PyTorch

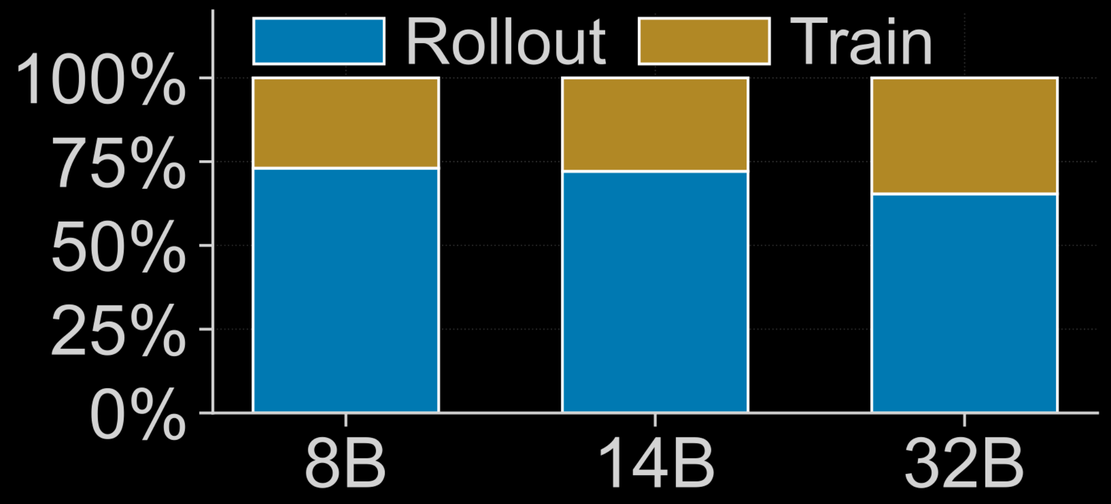
FSDP Megatron

Compute reward for each response and derive losses to update the model.

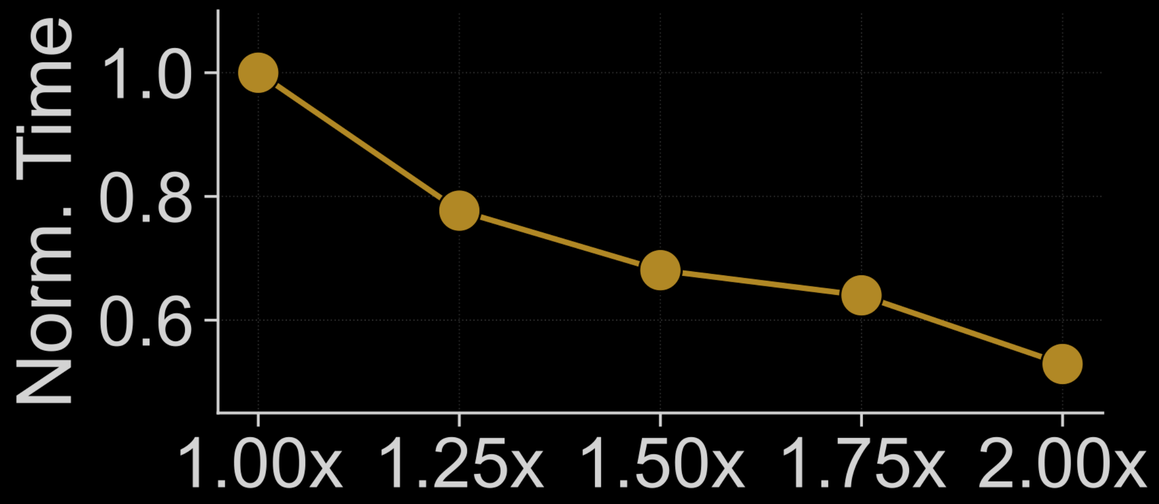
Sync Model Weights

Rollout Dominates Time

Rollout dominates a RL step, but scales efficiently with more resources



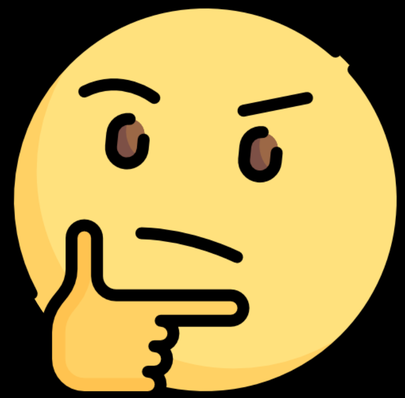
Over **70%** time is spent on rollout!
Even worse on agentic tasks.



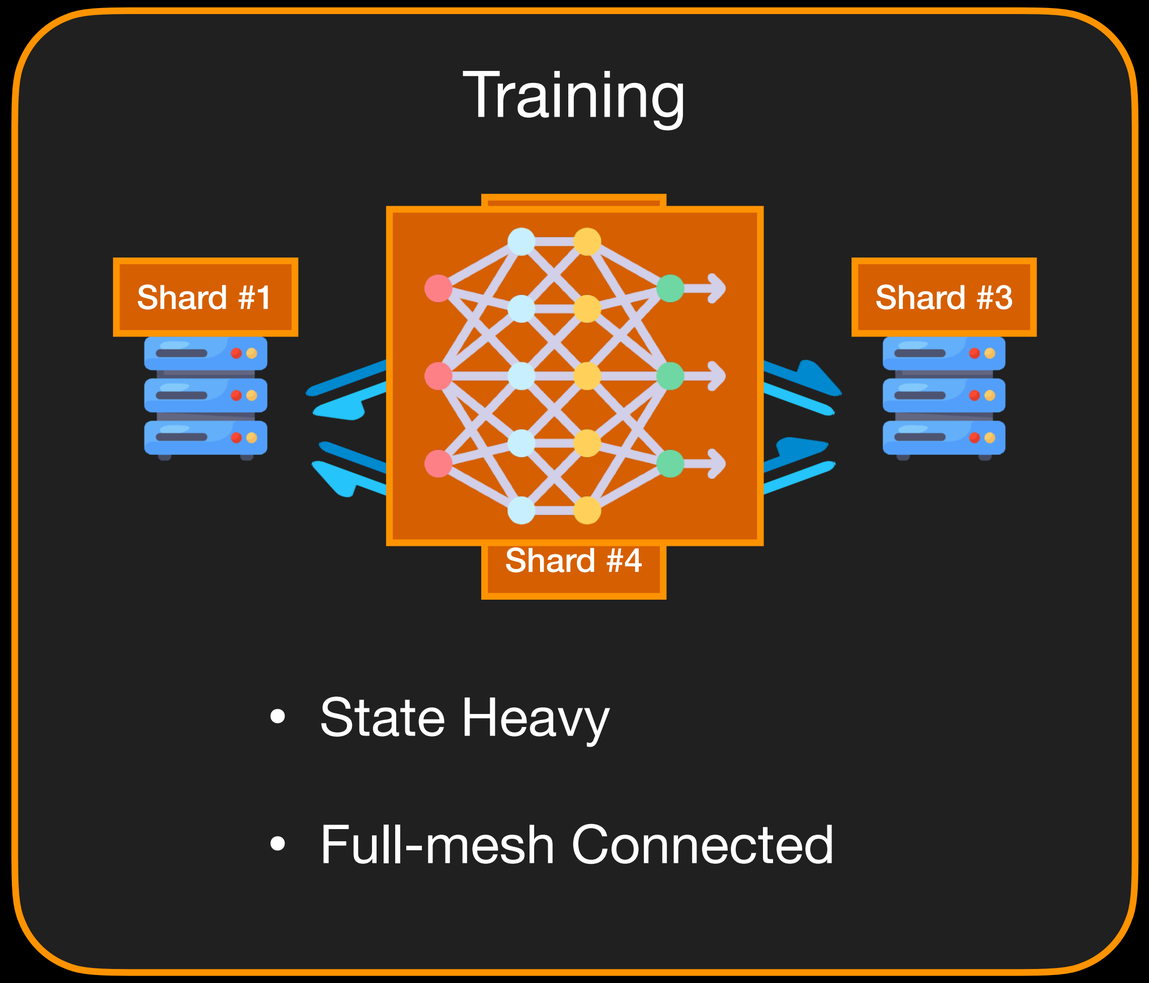
#Rollout GPUs / #Training GPUs
Speedup rollout by adding resources.

Question

How to speedup rollout
while minimizing the cost?

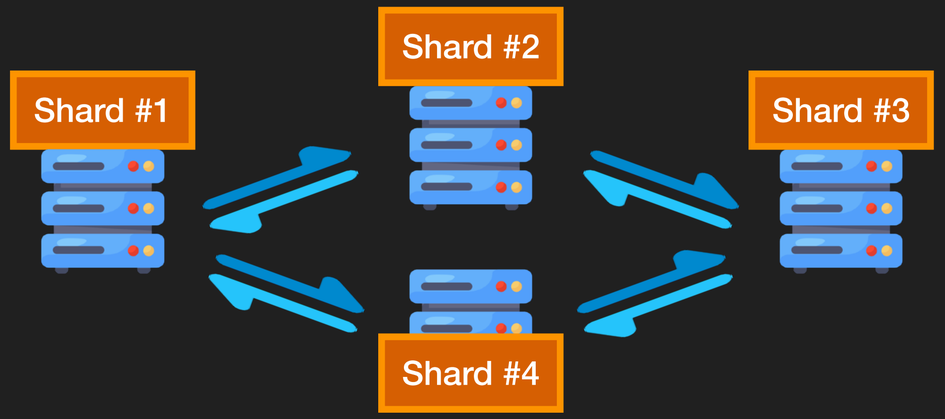


State Heavy Training vs. Embarrassingly Parallel Rollout



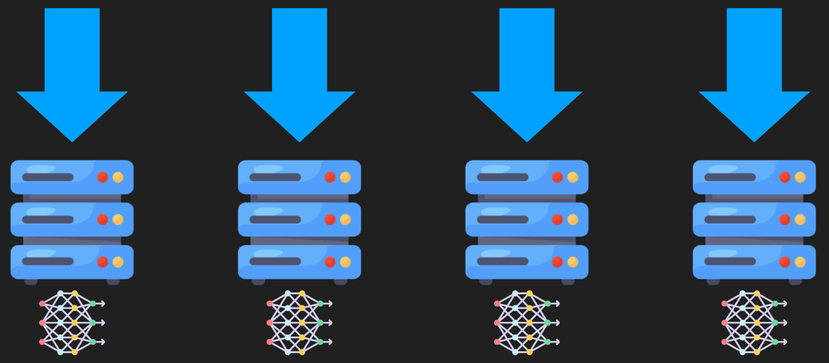
State Heavy Training vs. Embarrassingly Parallel Rollout

Training



- State Heavy
- Full-mesh Connected

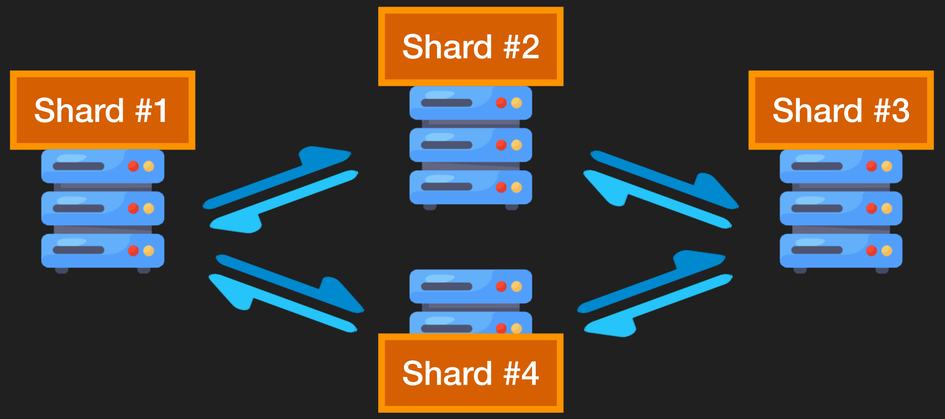
Rollout



- Nearly Stateless
- Independent

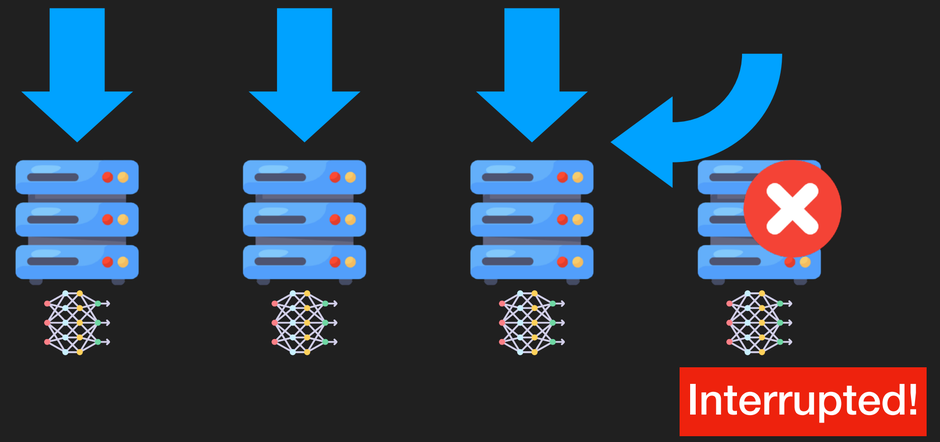
State Heavy Training vs. Embarrassingly Parallel Rollout

Training



- State Heavy
- Full-mesh Connected

Rollout



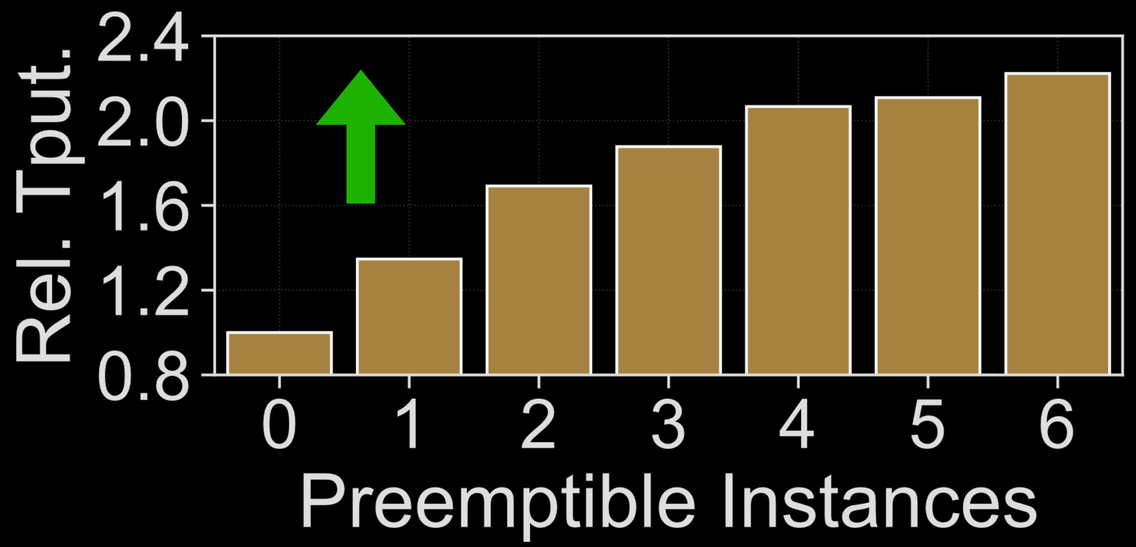
- Nearly Stateless
- Independent

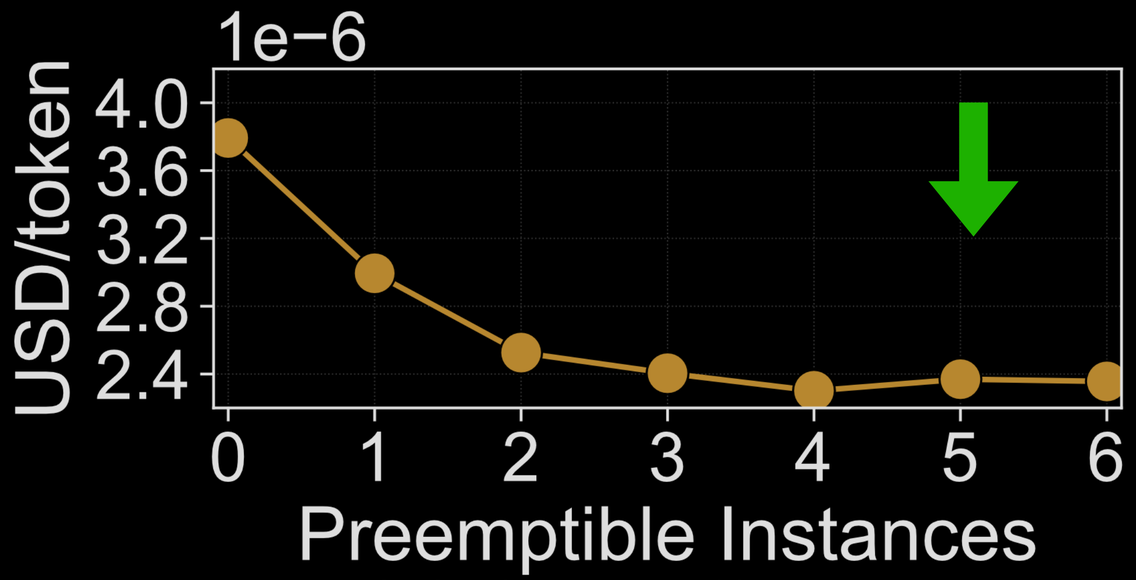
Insight: Rollout Can Exploit Preemptible Resources

Training → Reserved (standard), tightly coupled GPU resources

Rollout → Preemptible (spot), fragmented GPU resources

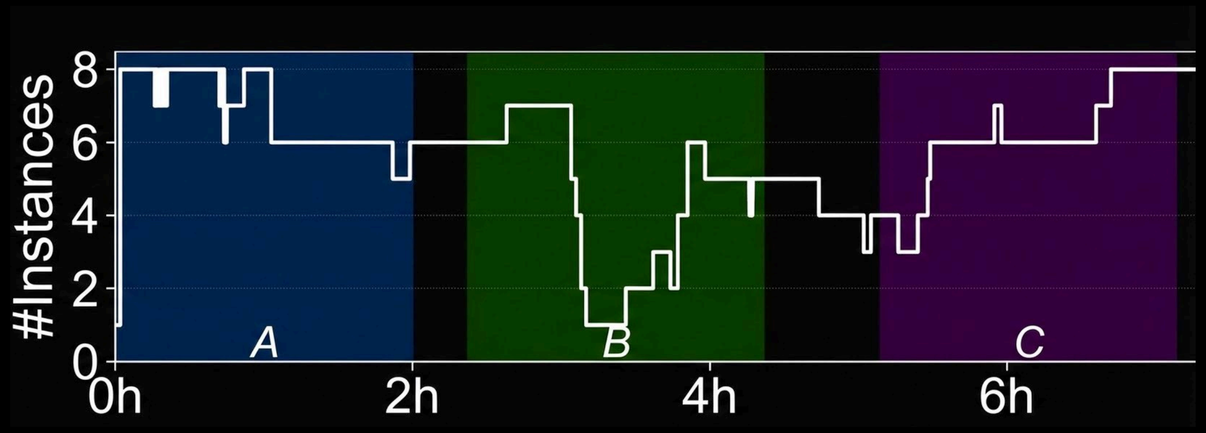
Match hardware to workload characteristics





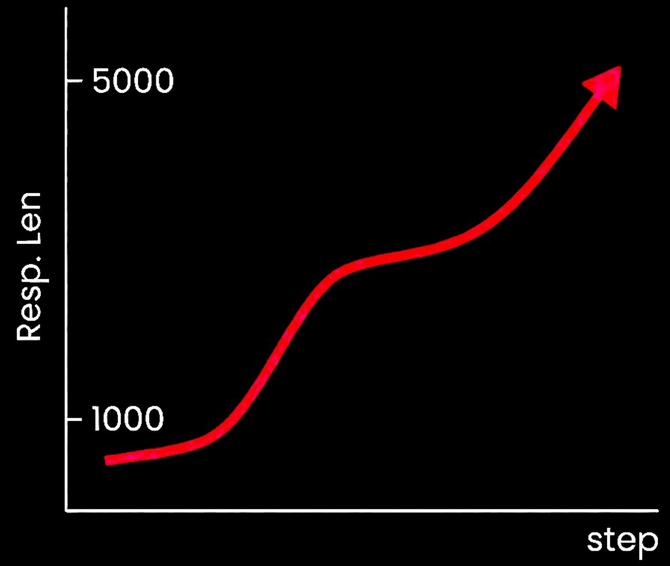
Unpredictable Resources

- Spot instance availability changes over time.
- Rollout throughput gain is unstable.



Dynamic Workload

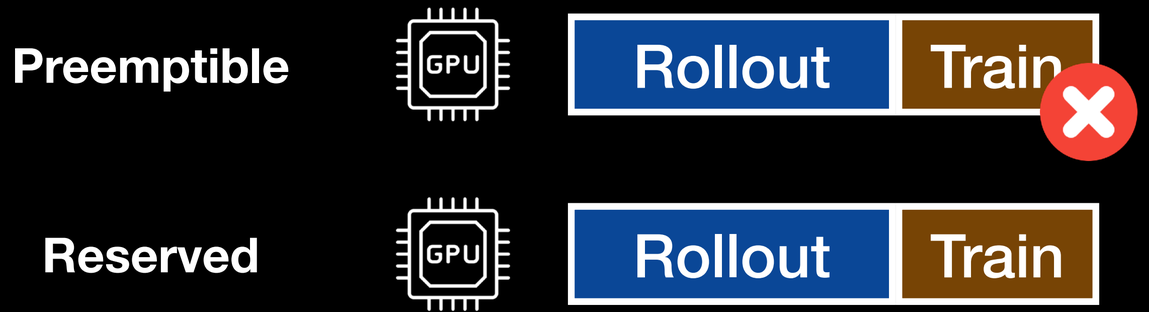
- Reasoning response length increases.
- Rollout becomes more and more dominant.



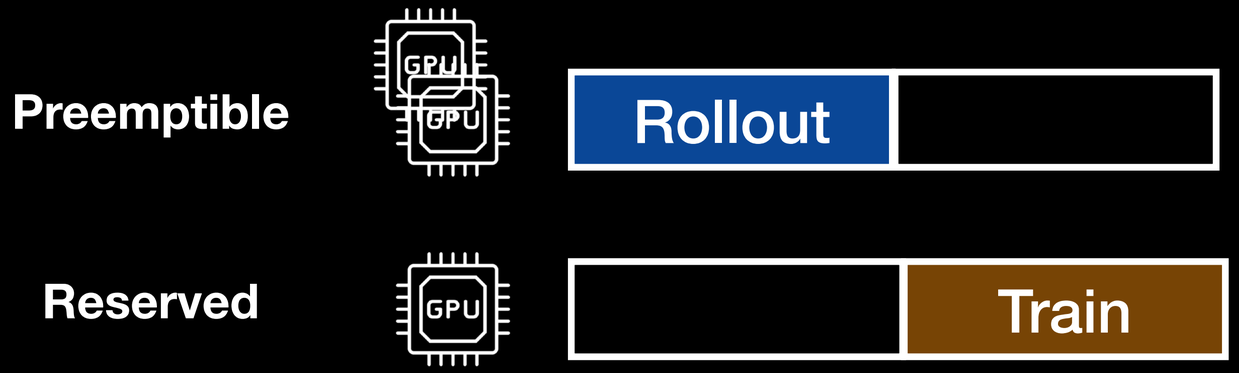
Goal

Design an elastic system to bridge dynamic workload and resources.

Limitation of Existing Architectures



Colocated Architecture[1]

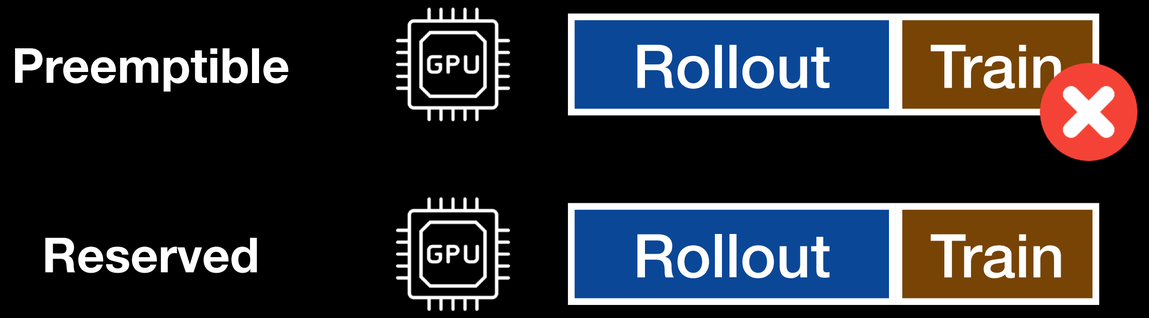


Decoupled Architecture[2,3]

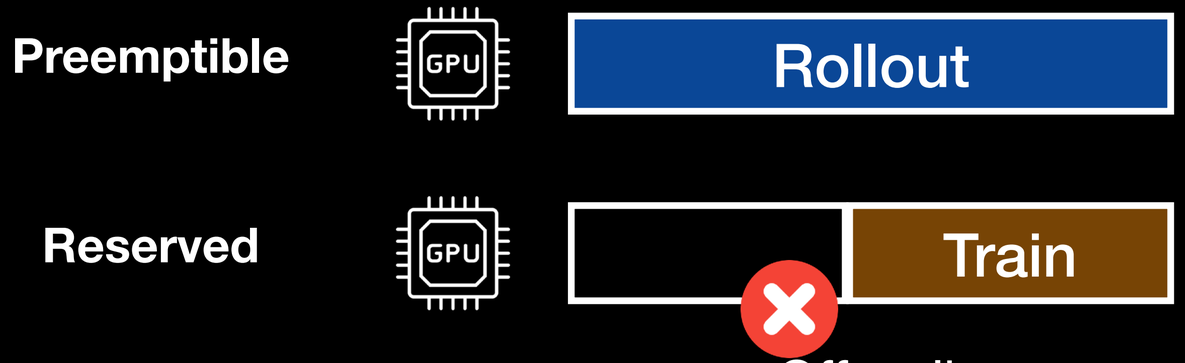
Off-policy

1. veRL: https://github.com/verl-project/verl
2. OpenRLHF: https://github.com/openrlhf/openrlhf
3. StreamRL: https://arxiv.org/abs/2504.15930

Limitation of Existing Architectures



Colocated Architecture[1]

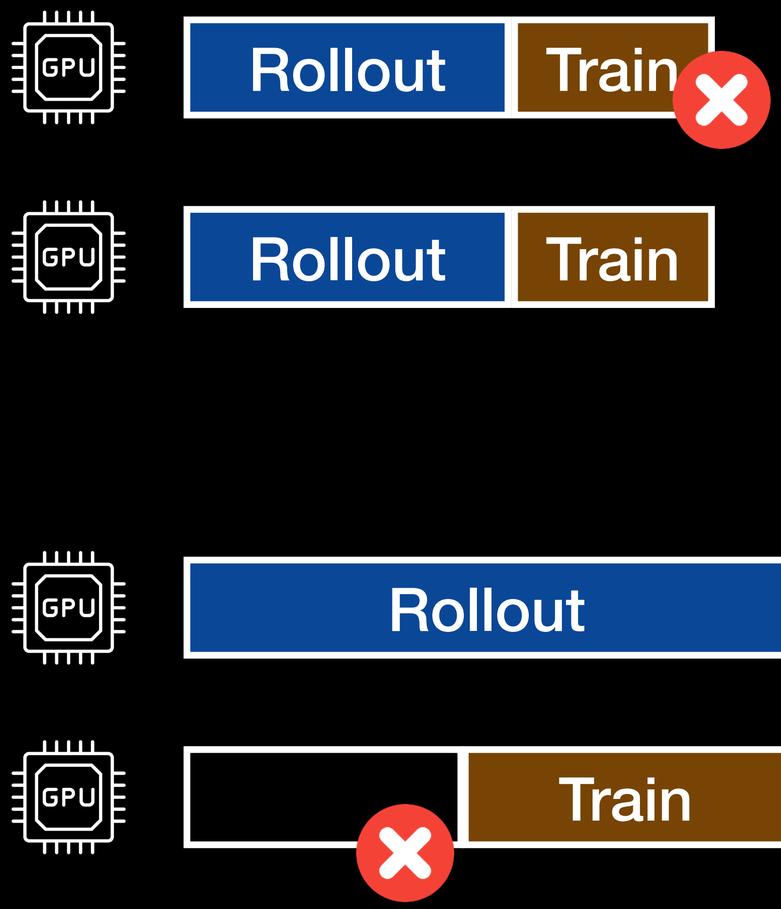


Off-policy

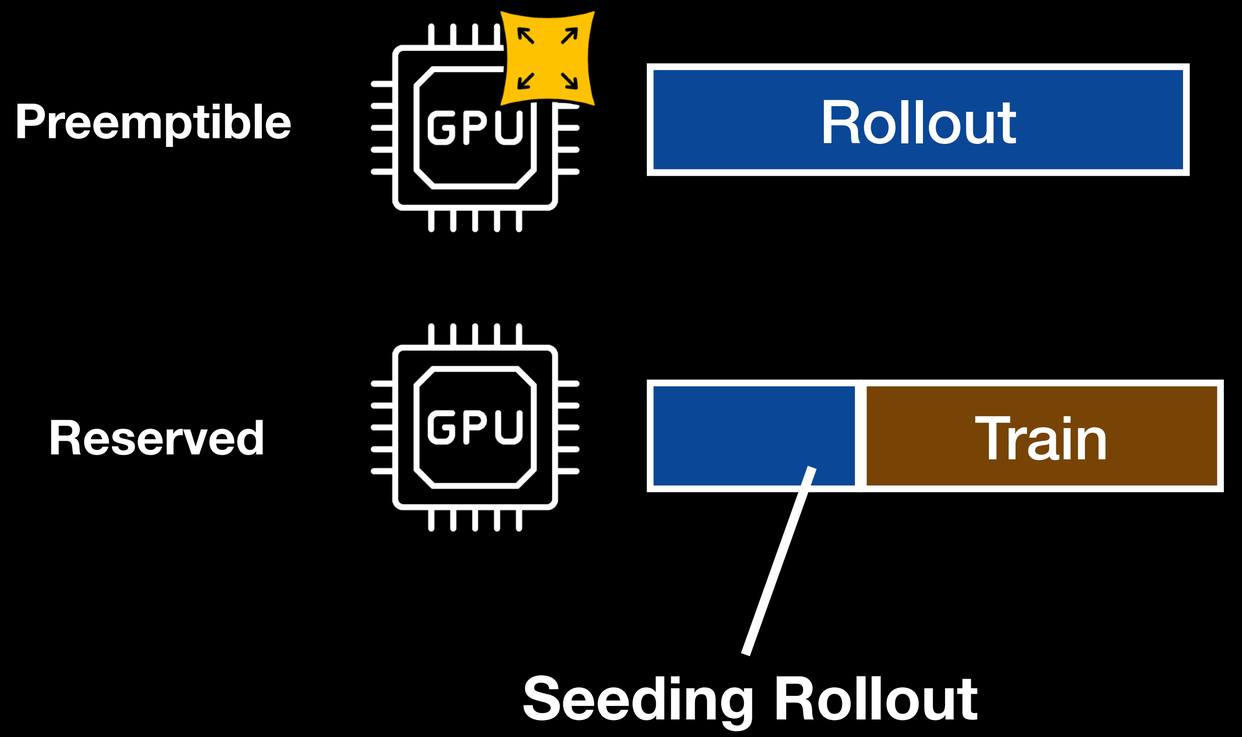
Decoupled Architecture[2,3]

1. veRL: https://github.com/verl-project/verl
2. OpenRLHF: https://github.com/openrlhf/openrlhf
3. StreamRL: https://arxiv.org/abs/2504.15930

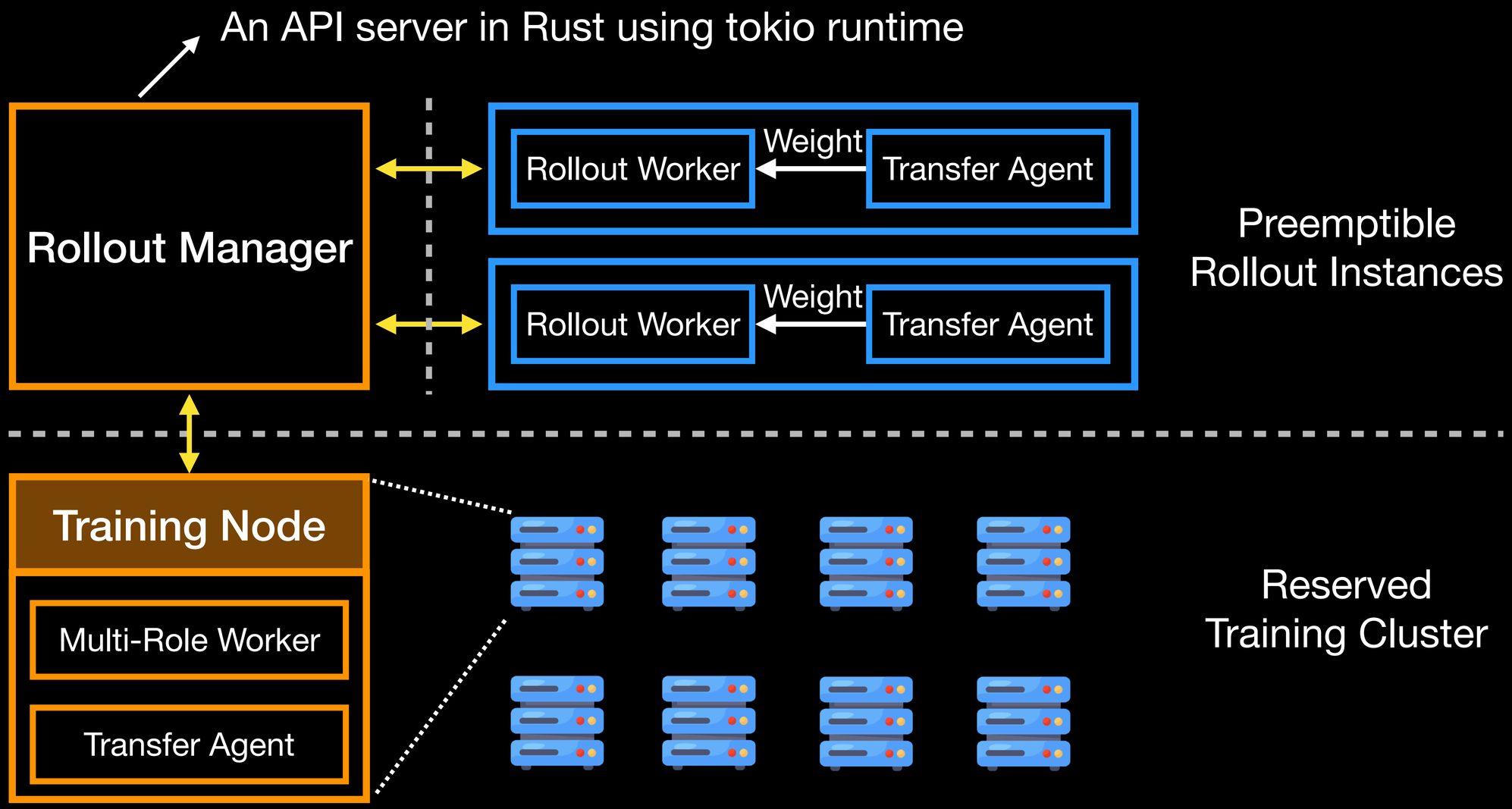
Insight: A Hybrid Architecture



Hybrid Architecture

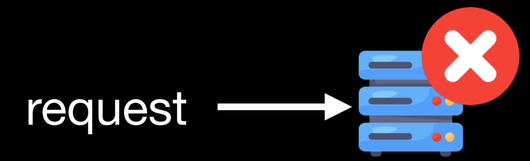


System Overview



Challenges

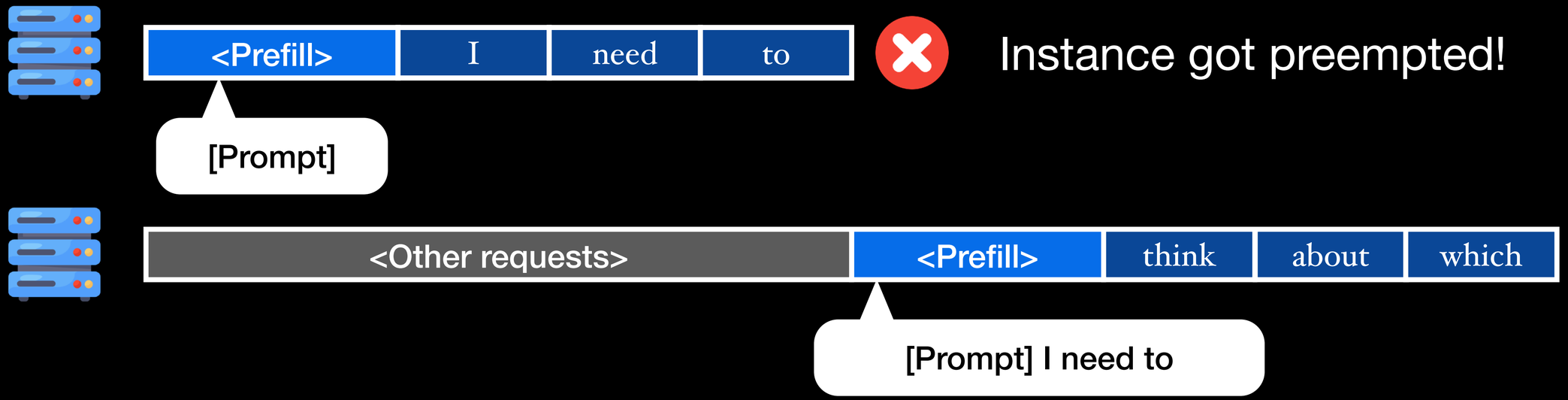
1. How to handle preemptions at mid-step?



2. How to opportunistically harvest the dynamic resources?

3. How to balance load across rollout workers?

1 Preemption: Continue with Partial Rollout

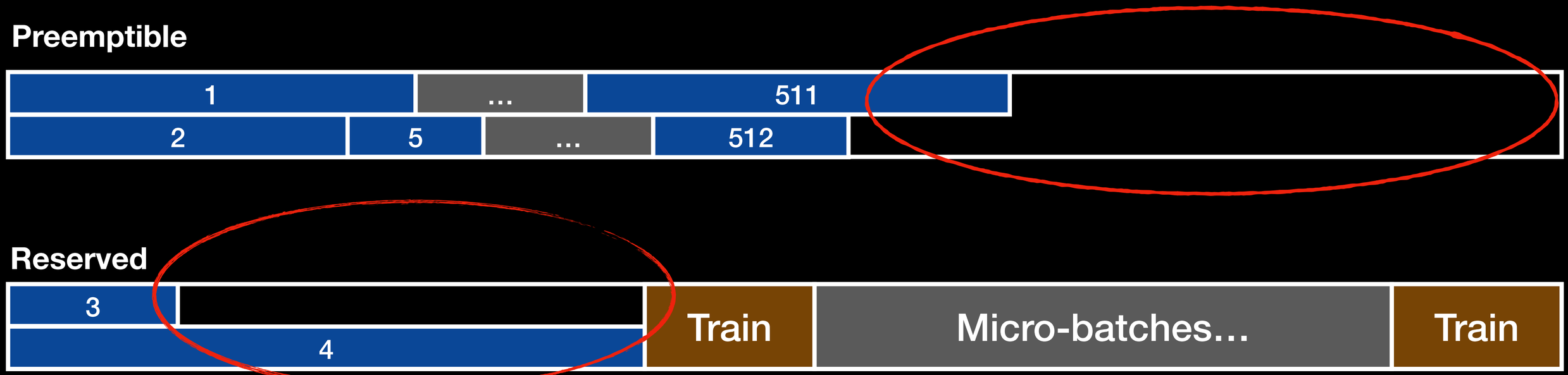


Collect responses with token streaming
Migrate requests with prompts + already generated tokens
Zero waste of progress on preemption and migration

2 Adaptive: Bubbles of Request-level Offload

Offload a subset of rollout requests to spot instances based on availability

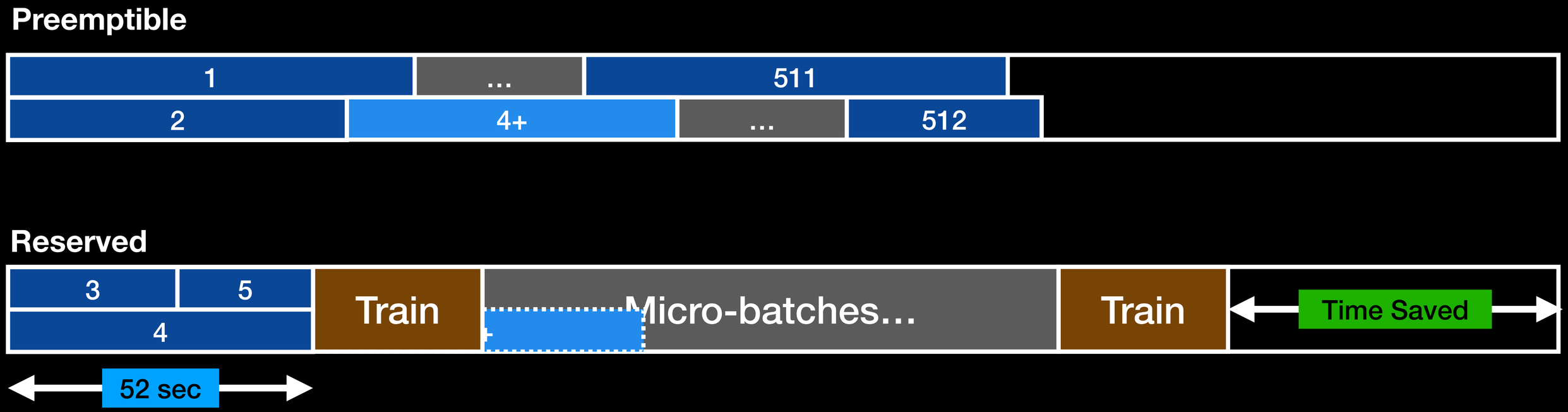
- Long tails requests causing bubbles.
- Enlarge rollout and training gap.



2 Adaptive: Partial Response Offload

Set a time limit and migrate responses in the middle

- Fine-grained workload balancing in second unit.
- Reactively adjusting time window based on availability.

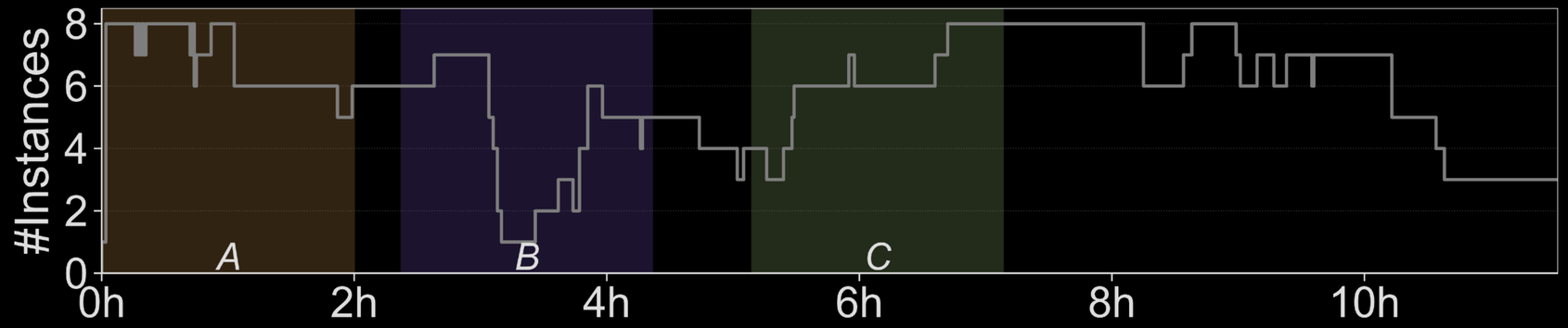


3 Reactive: Continuous Inter-device Balancing

Migrate pending requests to underutilized instances

Evaluation: Setup

- Data: Real world spot instance trace with 3 representative segments[1].

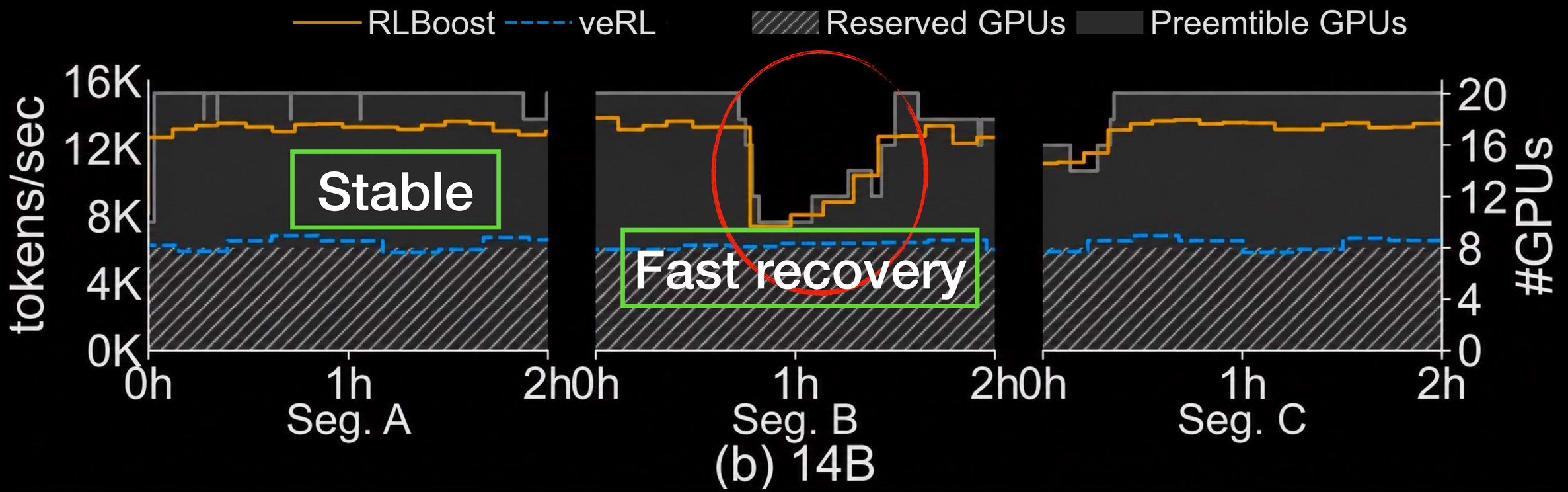


- Baselines:
  1. veRL: SOTA colocated RL framework; use ONLY reserved training cluster.
  2. Disagg.BAL: Adapted from StreamRL; balanced rollout and training GPUs
- Hardware:

Reserved	8xH100	(4+1)x200 Gbps	$10.47/hour/GPU
Preemptible	2xH100	50 Gbps	$2.66/hour/GPU

1. Thorpe, John, et al. "Bamboo: Making preemptible instances resilient for affordable training of large {DNNs}." *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023.

Evaluation: Throughput

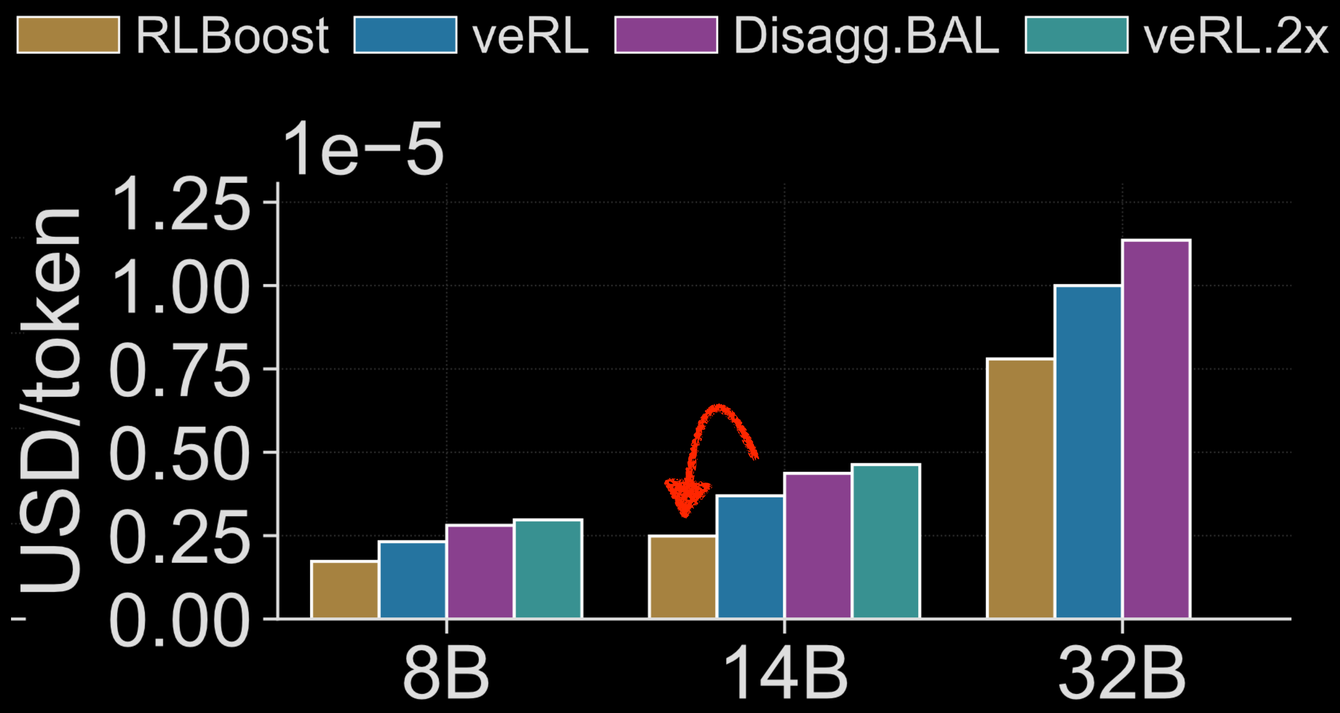


Preemptible instance number stack on top of reserved.

RLBoost's throughput ∝ #available instances

Evaluation: Cost Efficiency

cost efficiency = throughput / cost



Improves cost efficiency
by up to 49%

The Position of RLBoost

RLBoost

- Is a boost to existing RL systems.
- Harvests preemptible instances when available.
- Retains base system’s efficiency in worst cases.

Elastic, Flexible, Cost Efficient

Conclusion

RLBoost: Cost-efficient RL with Preemptible Resources

- Hybrid Architecture: **Opportunistically** rollout offload
- **Zero-waste** on preemption and **instant** provision
- **Reactive** Inter-stage and inter-device load balancing
- Improves cost efficiency by **28%-49%**


liuxs@umich.edu


https://github.com/Terra-Flux/PolyRL