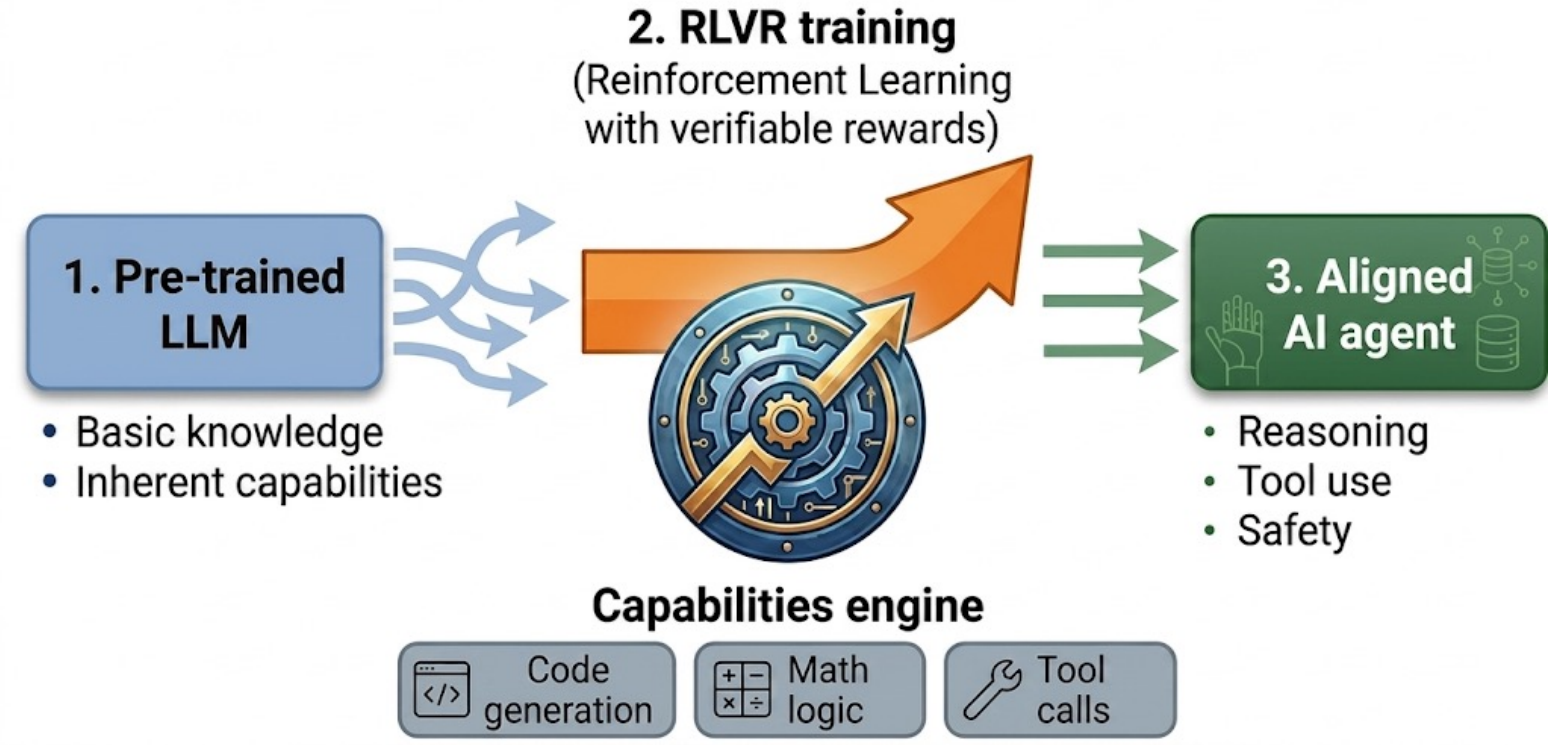
DistRS: Disaggregated Reward Service for RLVR with Batch-Level Constraint

Ruidong Zhu, Mingcong Han, Yinmin Zhong, Wencong Xiao, Xuanzhe Liu, Xin Jin

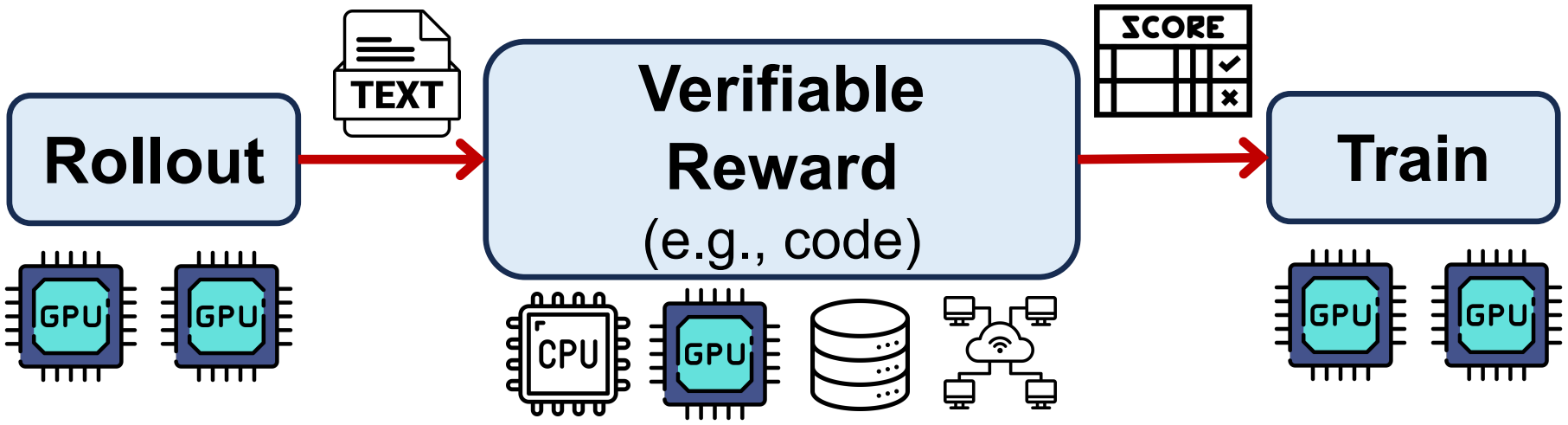
Reinforcement Learning with Verifiable Rewards (RLVR)

- RLVR has emerged as a key post-training paradigm for agentic LLM



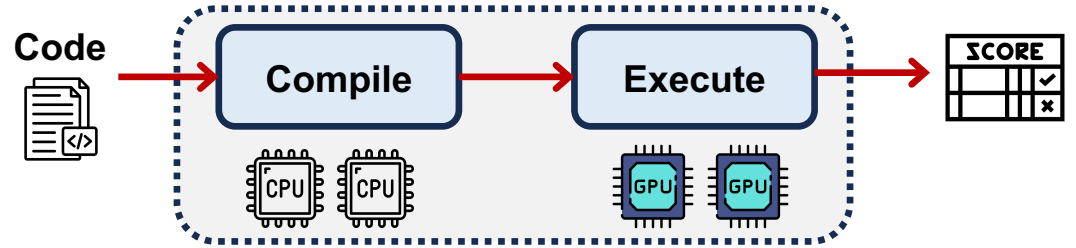
Reinforcement Learning with Verifiable Rewards (RLVR)

- Workflow of RLVR:
  - Rollout
  - Verify and produce reward
  - Train

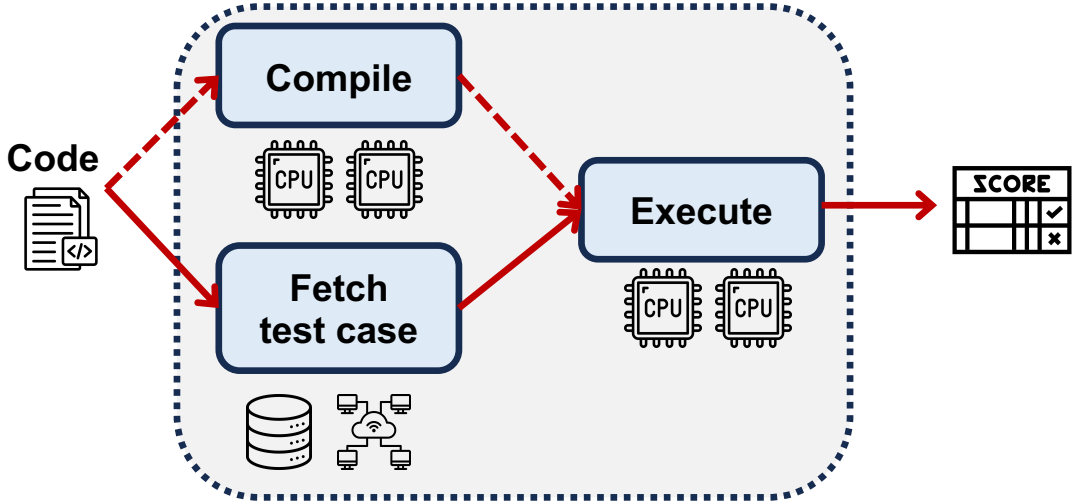


Examples of Verifiable Rewards

- Reward computation may use variable resources
  - CPU, GPU, network, storage, …



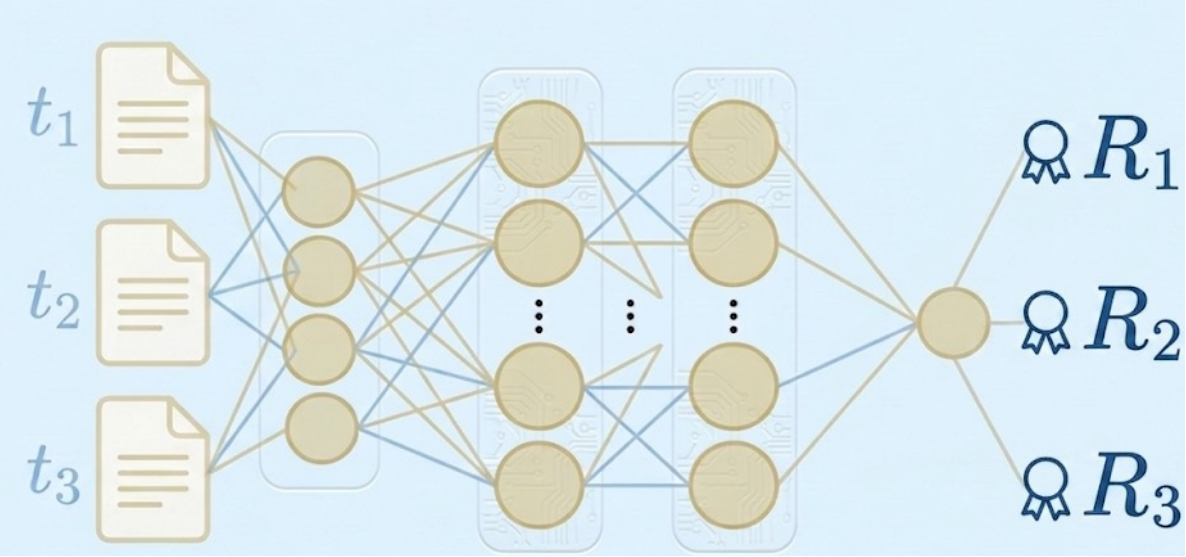
(a) CUDA reward computation.



(b) OJ reward computation.

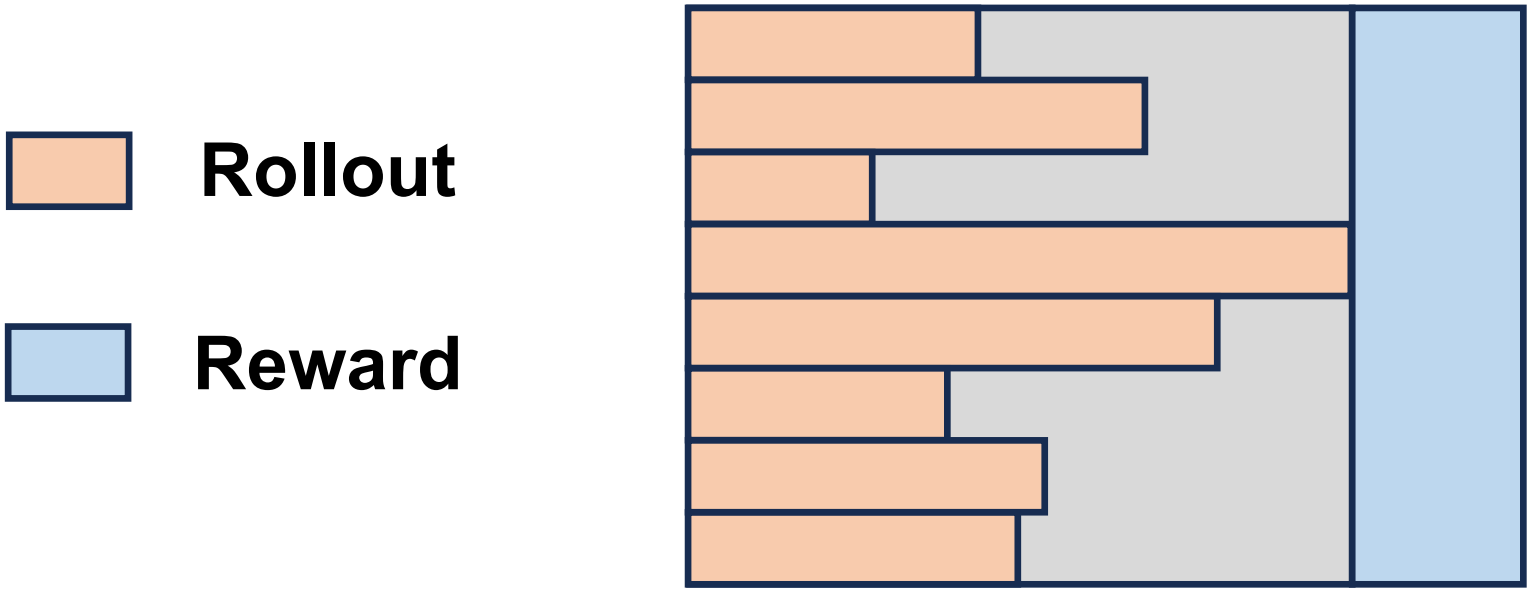
Reward in RLHF (Reinforcement Learning from Human Feedback)

- RLHF uses reward model
  - Batch requests and colocate reward computation with training/rollout



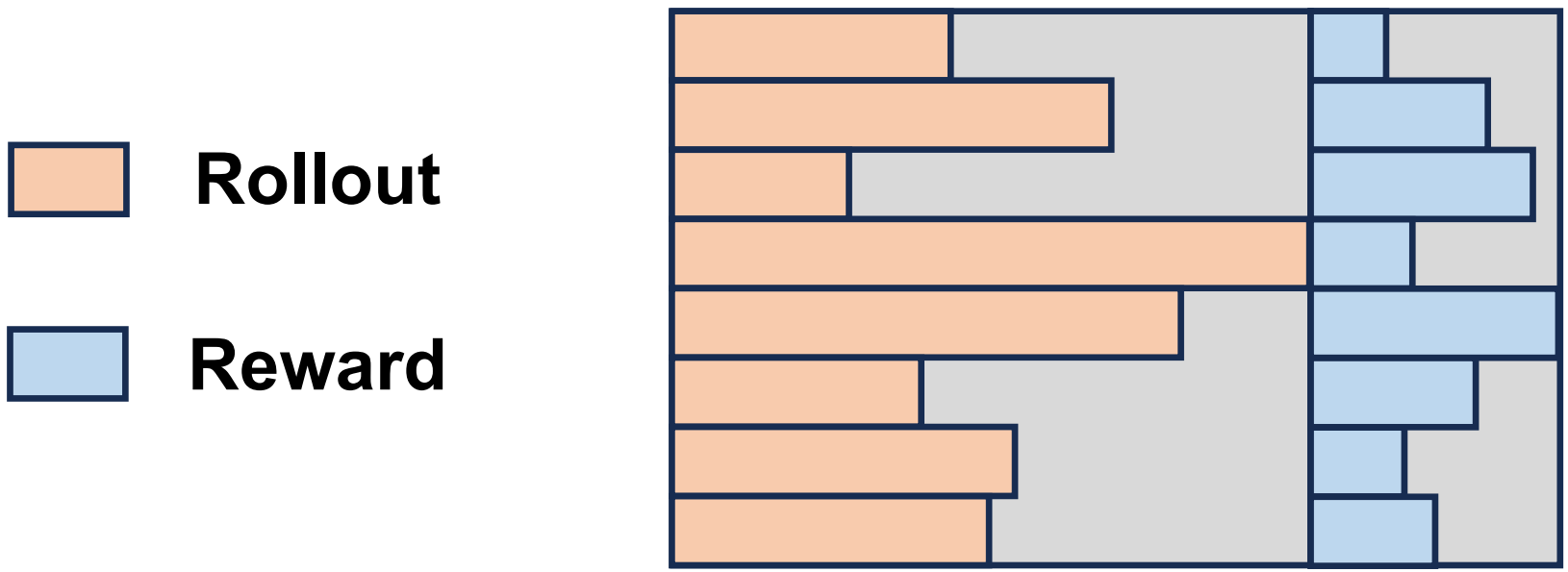
Reward in RLHF

- RLHF uses reward model
  - Batch requests and colocate reward computation with training/rollout



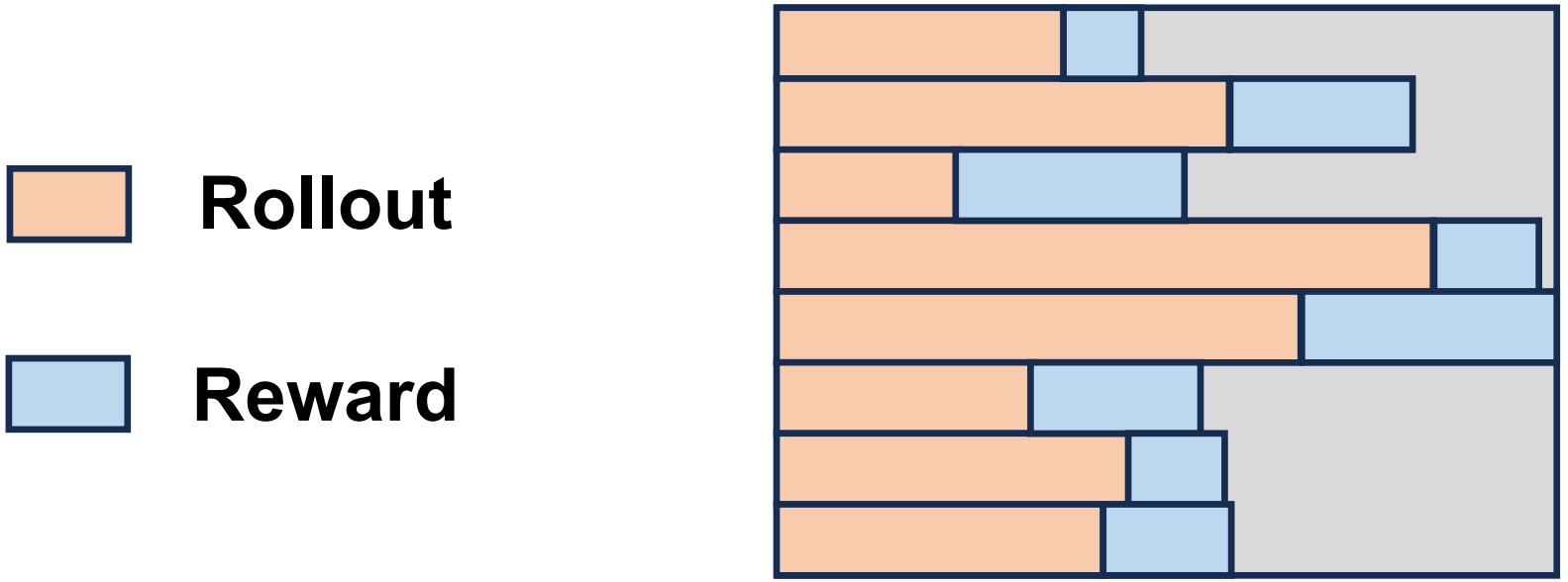
Reward in RLVR

- Each reward computation is independent
- Reward requires various resources
- Colocating and batching leads to inefficiency



Reward as a Service

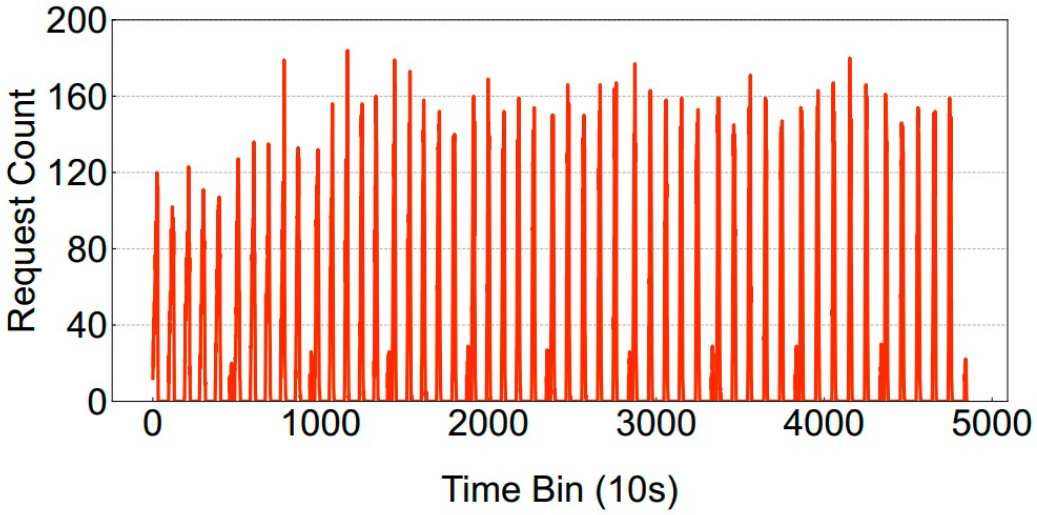
- Deploy the computation of reward as a disaggregated service
  - Compute reward at the request level
  - Flexible resource allocation



Workload Characteristics: A CUDA Reward Service

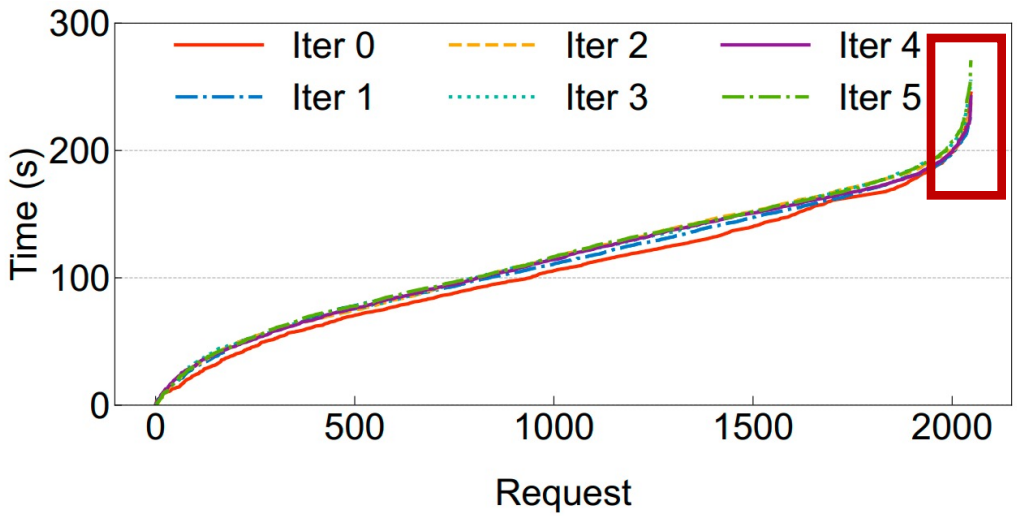
- Arrival pattern

Bursty



Arrival rate during RLVR training

Long-tail

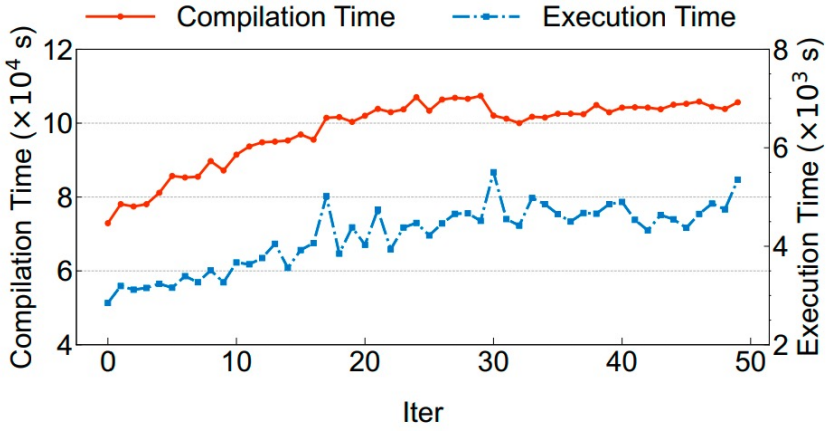


Request arrival time for 6 iterations

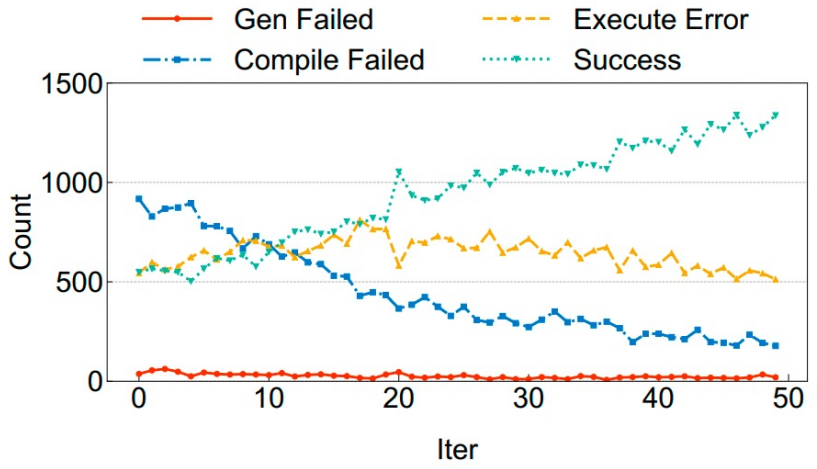
Workload Characteristics: A CUDA Reward Service

- Resource consumption

Fluctuate during training

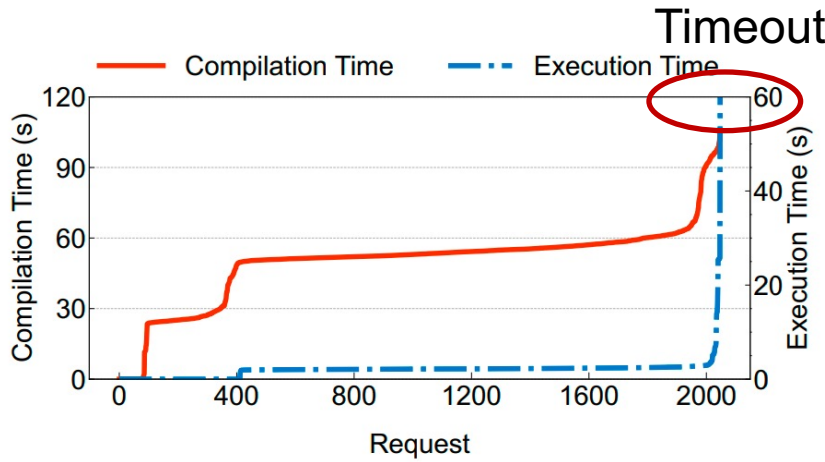


Total compilation and execution time of each iteration



State distribution of each iteration

Long-running request



Compilation and execution time of each request in an iteration

Summary and outlook

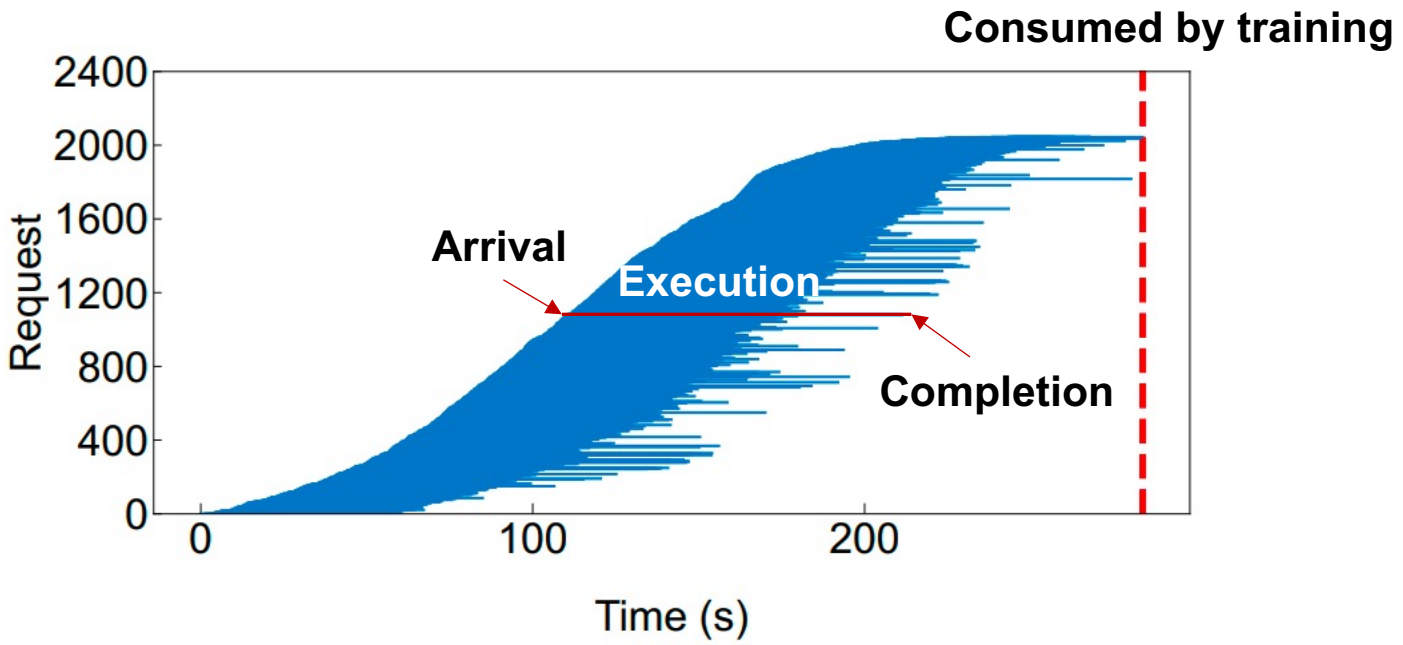
Dynamic workload call for an elastic, multi-tenant reward service

Key challenge: what policy should govern adaptation?

Scaling policy **Scheduling policy**

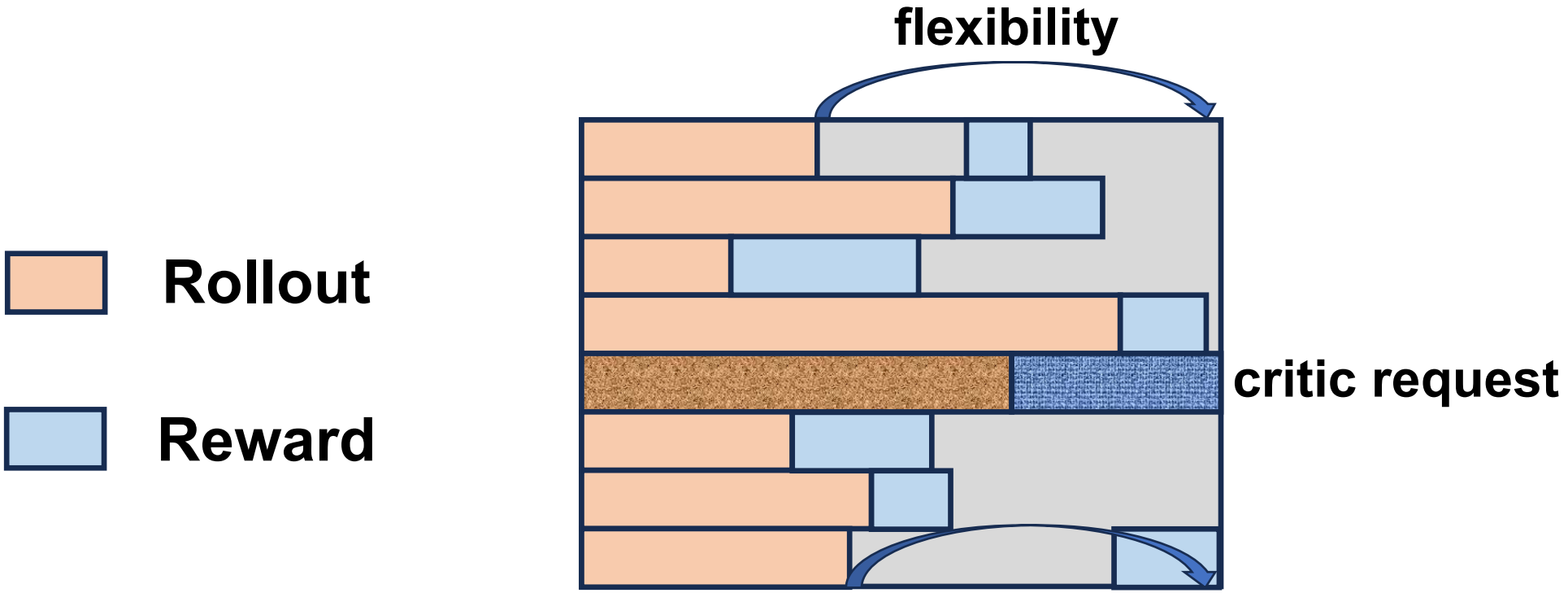
Request-in, Batch-out

- Each request arrives at different time
- But the output is consumed by batch



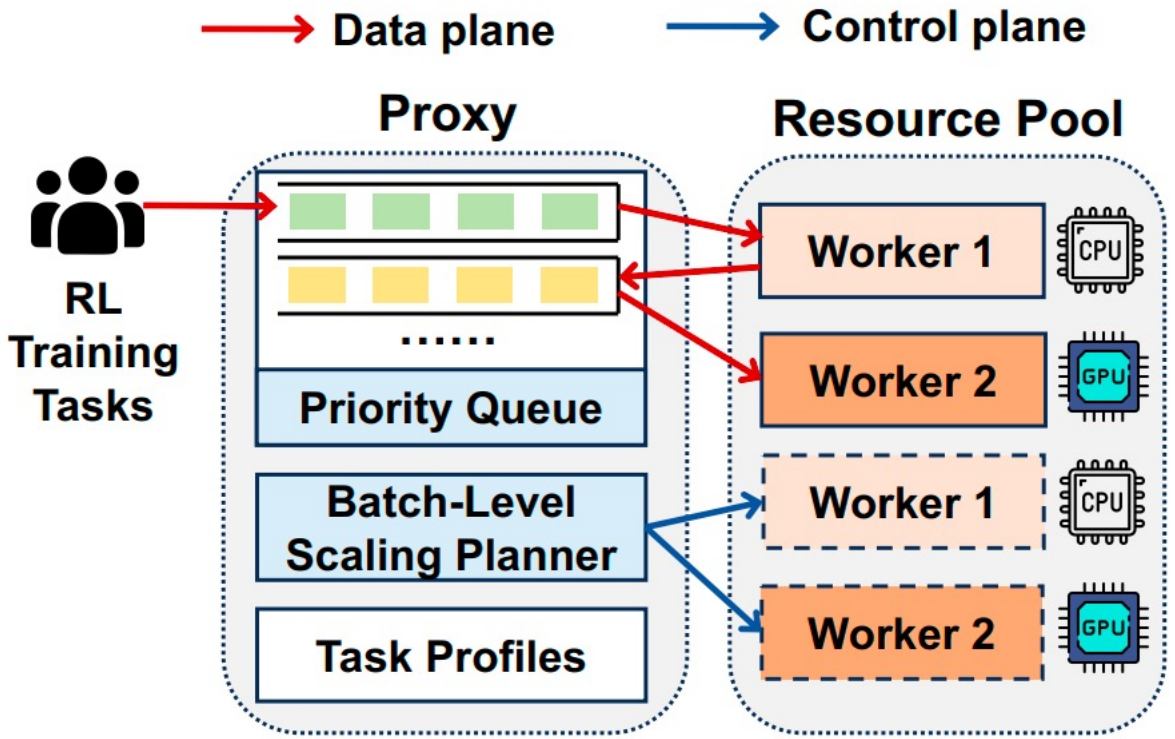
Request-in, Batch-out

- **Batch-level constraint:** SLO of reward service is defined on batch-level
- **Request-level flexibility:** Requests can be delayed until it can complete before the batch-level deadline



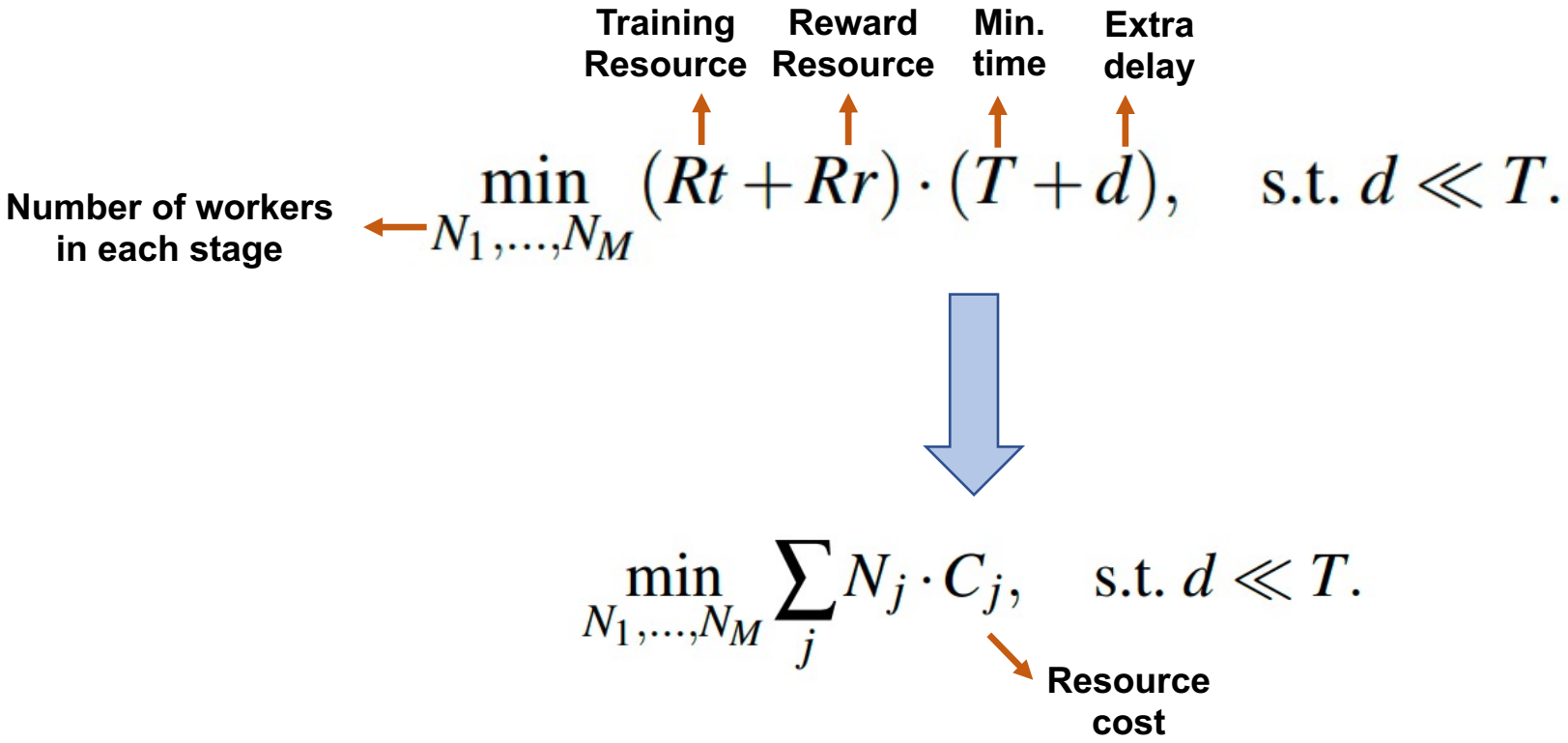
DistRS

- Elastic, multi-tenant reward service
- Micro-service architecture
  - Each stage has its own worker
- Determine scaling policy and scheduling policy at batch-level



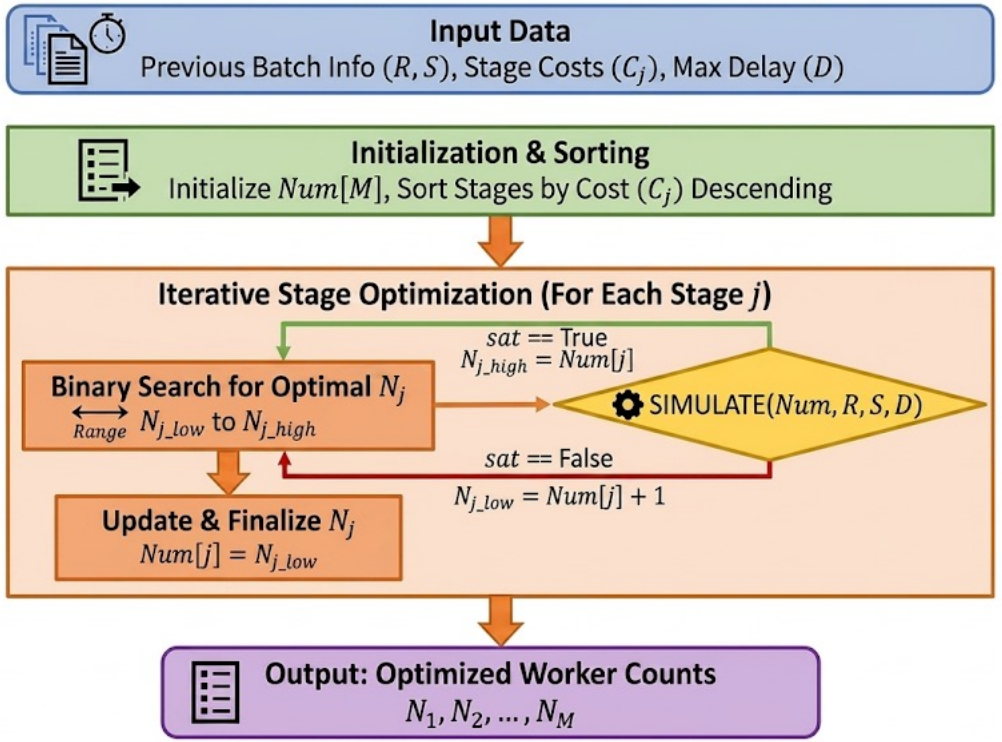
Scaling Policy for Single Task

- Scale at the granularity of batch
- When: the first request of a new batch arrives
- Output: the number of workers allocated to each stage



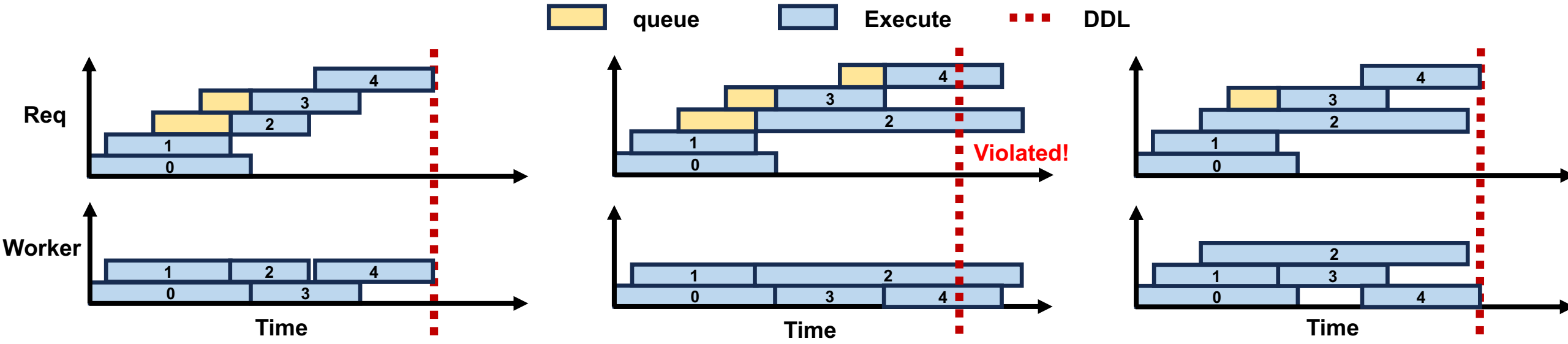
Scaling Policy for Single Task

- Objective: minimize the resource allocation with minimal impact on training
- Solution: Simulation-based searching



Handling Long-Running Requests

- Long-running requests may cause constraint violation



(a) History-based allocation.

(b) Violation due to long-running requests.

(c) Ideal allocation.

Handling Long-Running Requests

Why reactive solutions fail

- Long-running requests are unpredictable

Timeout happens after waiting accumulates

Our approach

- Proactive, timeout-aware adjustment

Detect risk early before execution

Detect

Estimate maximum latency when queuing

$$LT = t_s + \Sigma_{j=k..M} TIMEOUT_j$$

Decide

Will it violate the constraint?

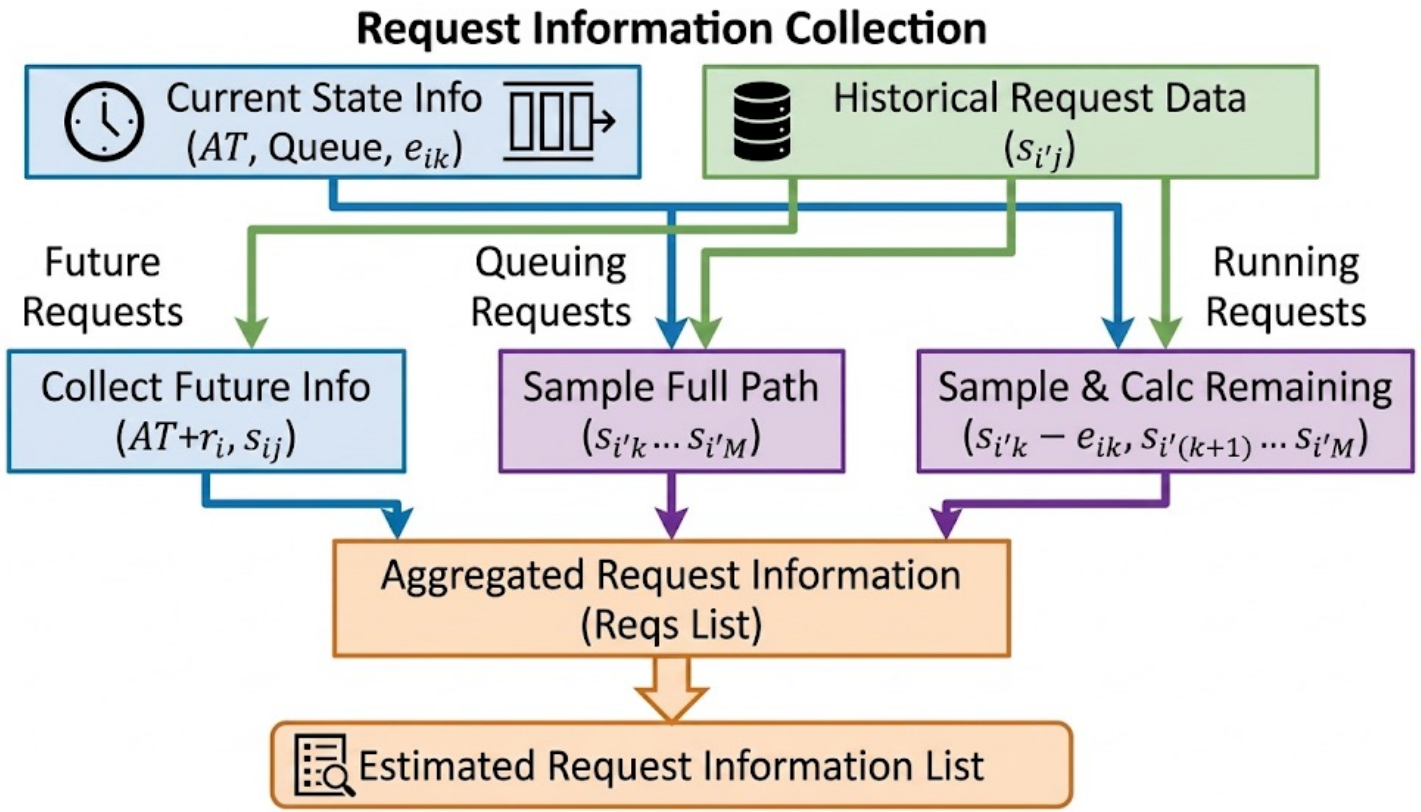
$$LT > T + D$$

Adjust

Treat as unsatisfied allocation and reallocate

Scaling and Scheduling Policy for Multiple Tasks

- Resource Scaling
  - Collect information from both Current state and history



Scaling and Scheduling Policy for Multiple Tasks

- Request Scheduling

Baseline: FCFS

Serve requests in arrival order

- Pros: simple, no per-stage state
- Cons: ignores batch deadlines and stage contention

Ours: Earliest Batch First (EBF)

Prioritize requests from the batch with the earliest deadline

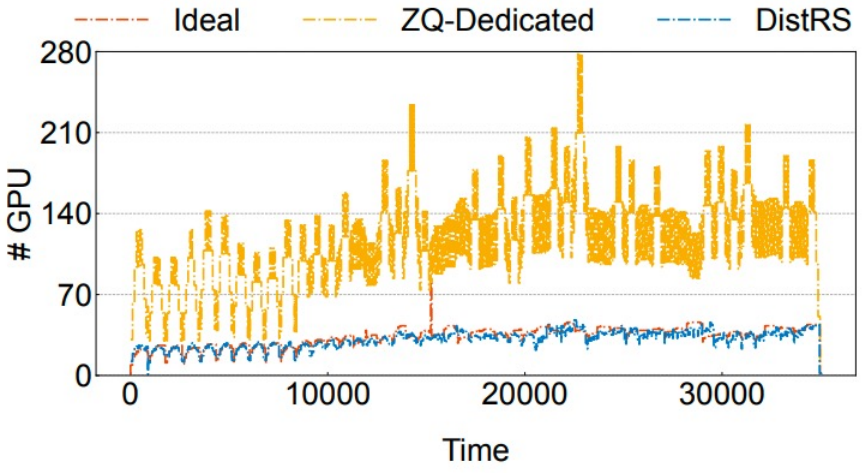
- Implementation: a priority queue per stage
- Benefit: reduces head-of-line blocking across tasks

Evaluation

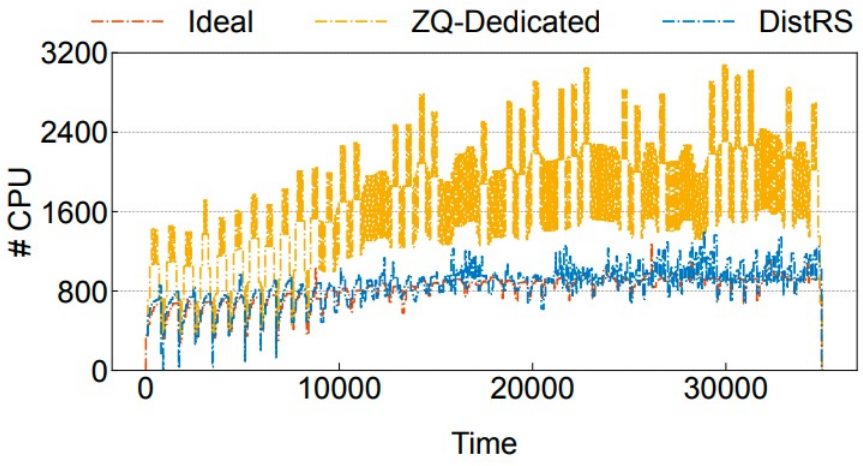
- Testbed: CUDA reward service
  - Compiler with 4 CPU cores
  - Runner with one NVIDIA GPU
- Workload:
  - Six RL training tasks for CUDA code generation from the trace
- Metrics:
  - Total GPU / CPU time
  - Extra delay of each batch

End-to-End Performance

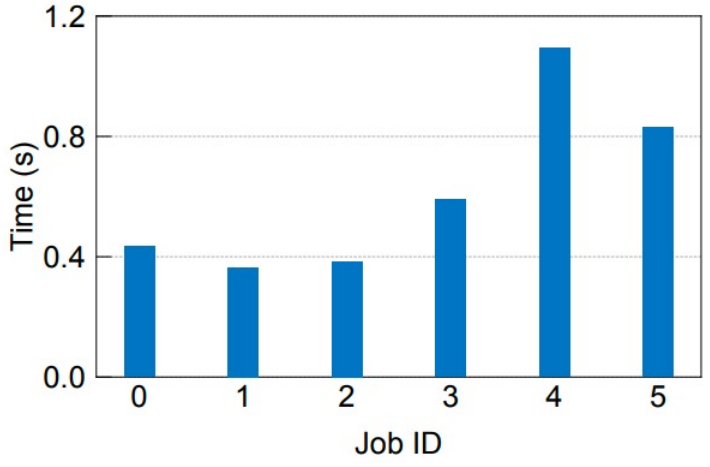
- DistRS reduces the GPU time by **3.79x** and CPU time by **1.98x** under colocated RL training
- The extra delay is **0.62s** in average



(a) Number of GPU nodes.



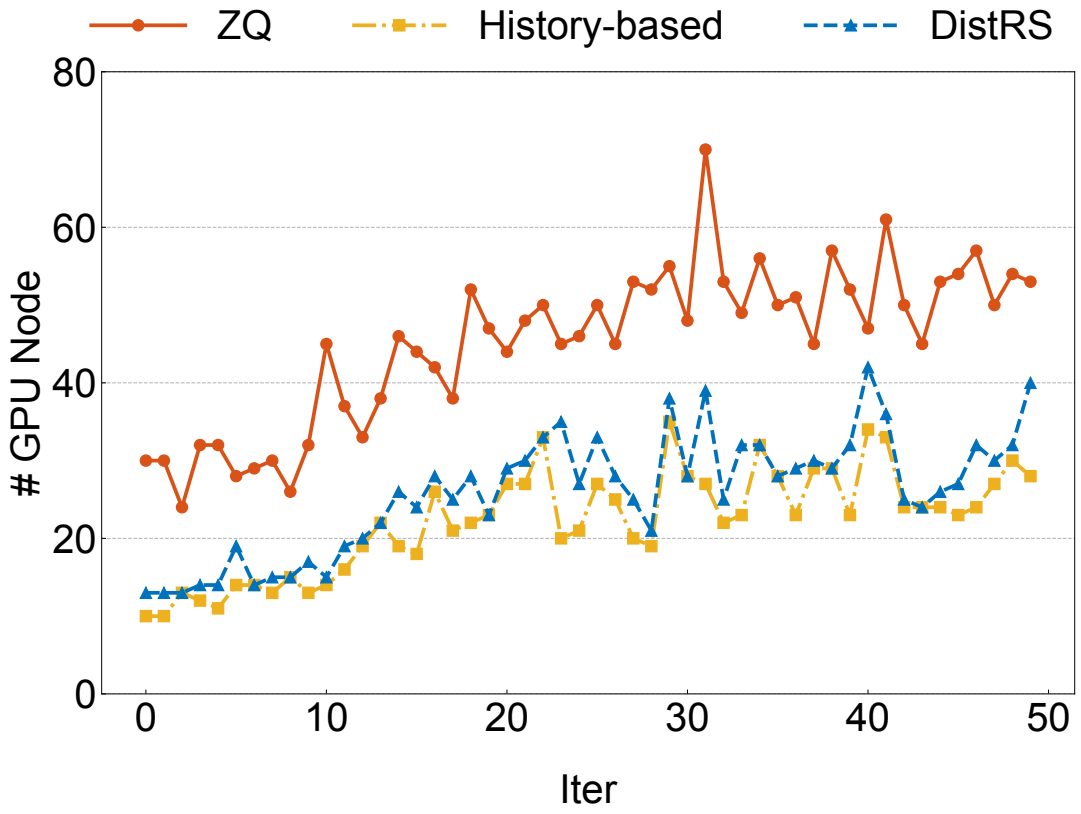
(b) Number of CPU nodes.

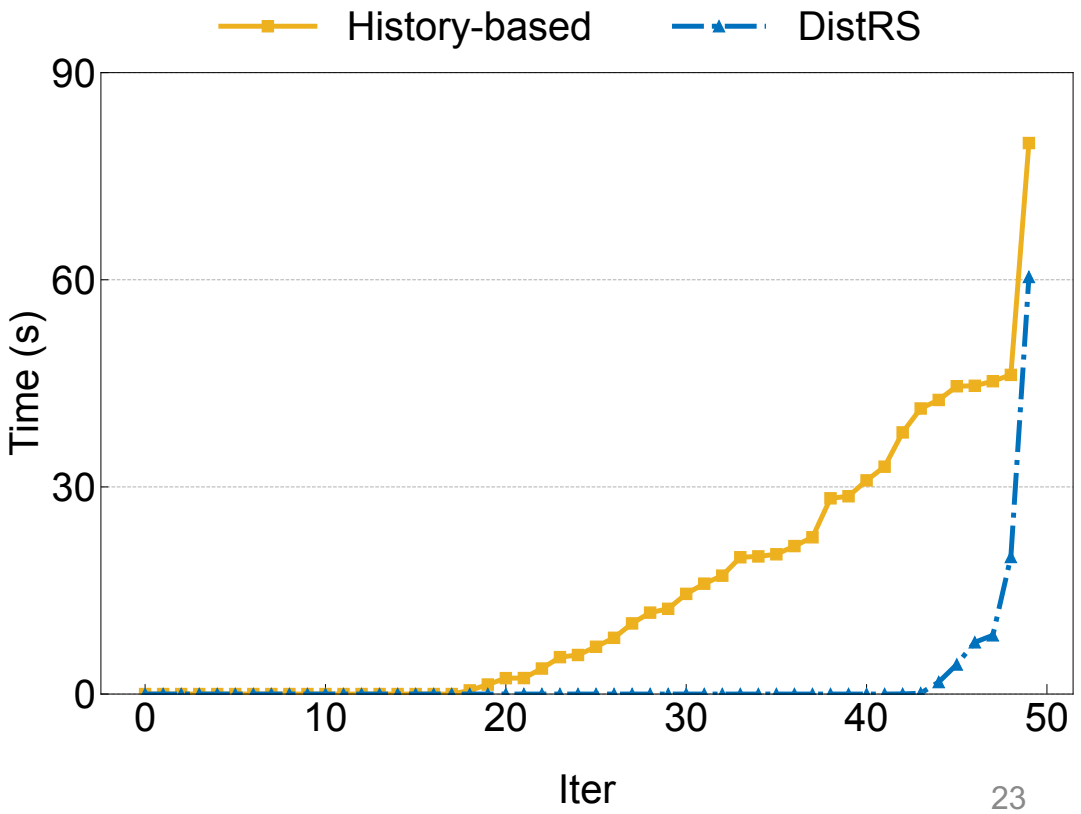


(c) Extra delay.

Effectiveness of Resource Scaling

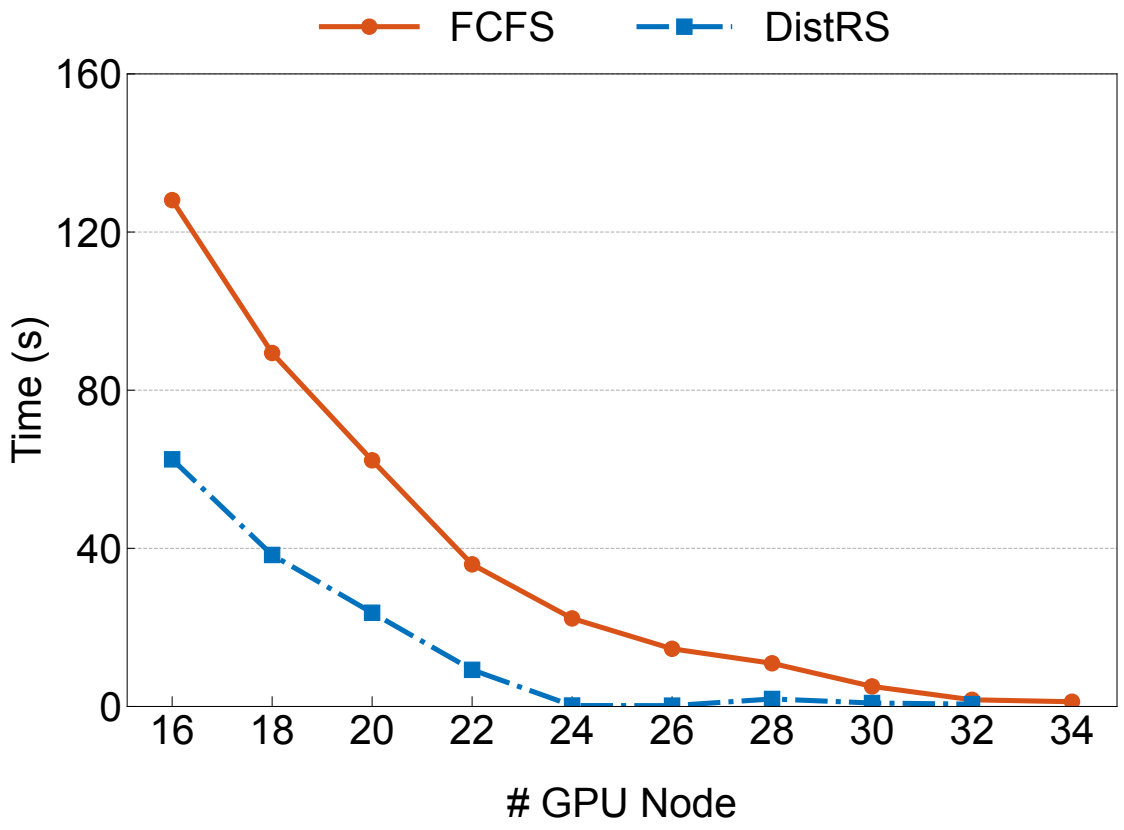
- DistRS reduces the resource consumption while minimizing the influence on training

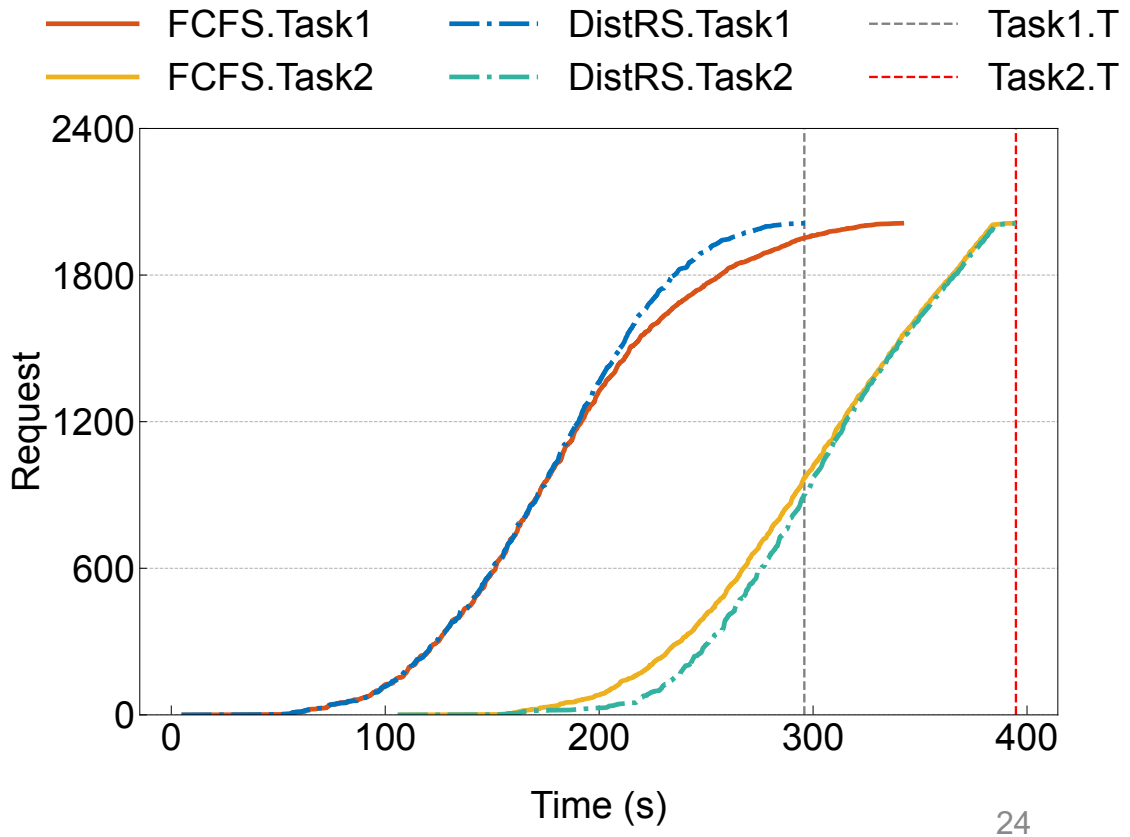




Effectiveness of Request Scheduling

- DistRS reduces extra delay under the same resource allocation compared to FCFS





Conclusion

- We present DistRS, a disaggregated reward service for agentic RL training
  - Elastic and multi-tenant
  - Leverage the request-in, batch-out characteristic
  - Batch-level constraint and request-level flexibility
  - Provide efficient reward computation while keeping training non-blocking

Thanks!

zhurd@pku.edu.cn