Bytedance
Seed

Building LLM Training Systems at Scale

Haibin Lin, Bytedance Seed - MLSys

Bytedance Seed

- Established in 2023
- **Mission**

  push the boundaries to un
- **Research directions**

  deep learning, reinforceme

Doubao: personal assistant

	Benchmark	Doubao-1.5-pro	GPT4o-1120	Claude3.5-Sonnet	Gemini-2-flash	Qwen2-VL-72B	InternVL-2.5-78B
College-level Problems	MMMU(val)	**73.8**	70.7	70.4	70.7	64.5	70.1
	MMMU-Pro	**59.3**	54.5	54.7	57.0	46.2	48.6
Mathematical Reasoning	MathVision	**48.6**	30.4	38.3	41.3	25.9	32.2
	OlympiadBench	**48.5**	25.9	27.8	43.6	11.2	25.1
	MathVista	**78.8**	63.8	65.4	73.1	70.5	76.6
Document and Diagrams Reading	TextVQA(val)	84.7	81.4	76.5	75.6	**85.5**	83.4
	ChartQA(test avg.)	88.0	86.7	**90.8**	85.2	88.3	88.3
	InfoVQA(test)	**88.0**	80.7	74.3	77.8	84.5	84.1
	DocVQA(test)	**96.7**	91.1	95.2	92.1	96.5	95.1
	Charxiv(RQ/DQ)	54.4 / 84.3	52.0 / **86.5**	**60.2**/ 84.3	55.2/81.8	43.0 / 81.3	42.4 / 82.3
General Visual Question Answering	RealWorldQA	**78.9**	75.4	66.6	74.5	77.8	78.7
	MMStar	71.9	63.9	65.1	69.4	68.6	**73.1**
	MMBench-en	87.5	83.5	81.7	83.0	85.9	**88.3**
	MMBench-cn	86.0	82.1	83.4	82.9	83.4	**88.5**
Spatial and Counting Understanding	Blink	**68.4**	68.0	59.6	62.6	61.1	63.8
	CountBench	**89.6**	85.1	86.8	88.2	88.6	84.1
Video Understanding	Video-MME	74.1	73.4	61.7	**78.2**	71.2	72.1
	EgoSchema-subest	75.4	74.8	64.4	71.8	**80.6**	78.2

Training systems at Scale

> 01.

Megascale

- **[Stability] Scaling Large Language Model Training to >10,000 GPUs**

> 02.

ByteScale

- [Data heterogeneity] LLM Training with a 2M Context Length

> 03.

HybridFlow (verl)

- [Programming Model] Reinforcement Learning System for LLMs

MegaScale: a production LLM training system

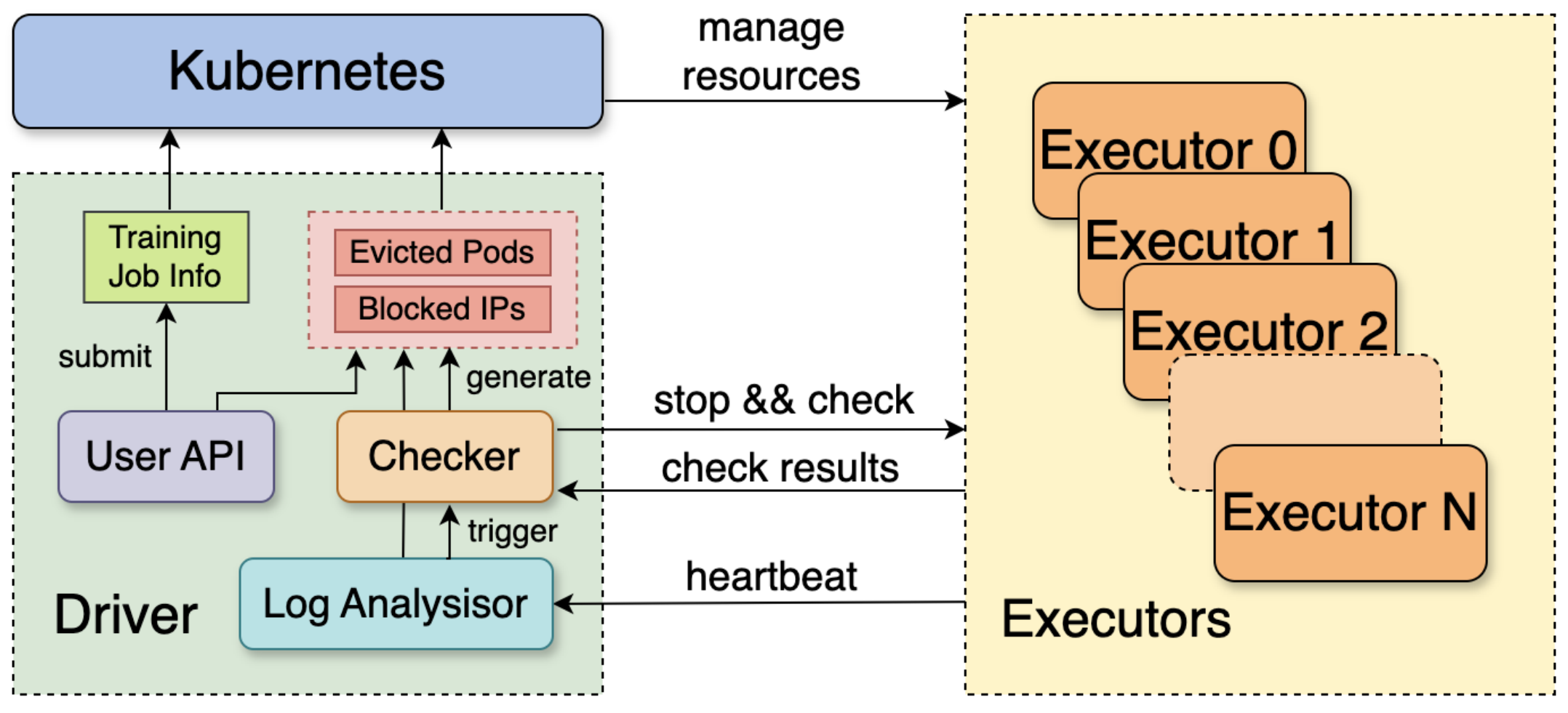
- **Efficiency**
  - nD parallelism (data/tensor/pipeline/expert parallelism) & large batch sizes
  - fused kernels, tile-level computation & communication overlap kernels (**flux / comet**)

- **Stability**
  - Frequent machine failures
  - Efficient checkpoint & restart
  - Root-cause detection

Component	Category	Interruption Count	% of Interruptions
Faulty GPU	GPU	148	30.1%
GPU HBM3 Memory	GPU	72	17.2%
Software Bug	Dependency	54	12.9%
Network Switch/Cable	Network	35	8.4%
Host Maintenance	Unplanned Maintenance	32	7.6%
GPU SRAM Memory	GPU	19	4.5%
GPU System Processor	GPU	17	4.1%
NIC	Host	7	1.7%
NCCL Watchdog Timeouts	Unknown	7	1.7%
Silent Data Corruption	GPU	6	1.4%
GPU Thermal Interface + Sensor	GPU	6	1.4%
SSD	Host	3	0.7%
Power Supply	Host	3	0.7%
Server Chassis	Host	2	0.5%
IO Expansion Board	Host	2	0.5%

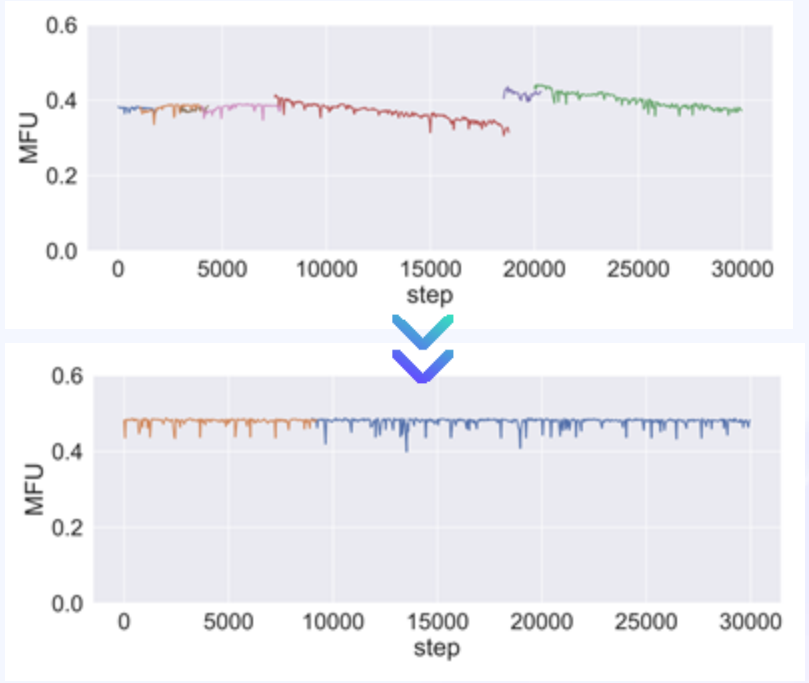
* The Llama 3 Herd of Models, 2024

MegaScale: robust training workflow



* MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs

MegaScale: complicated failures



- **Gray failures & stragglers**

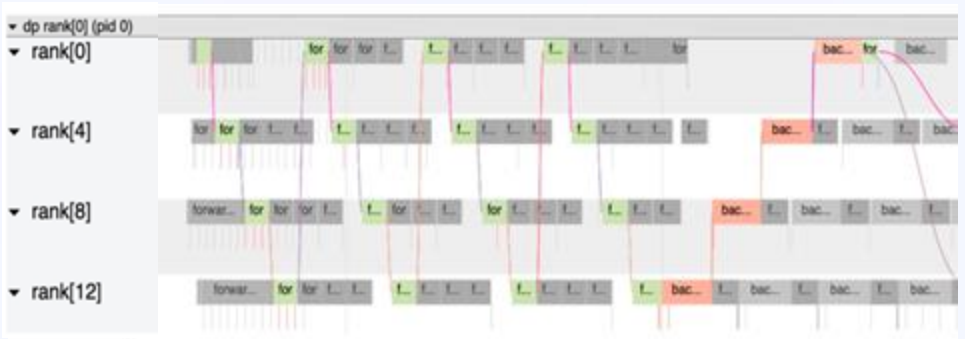
  Slow GPU, python garbage collection, etc

- **Silent data corruptions: long diagnosis cycle**
  - Manufactured with latent silicon defects
  - Sensitive to temperature, voltage, workload
  - Incorrect memory/arithmetic/logic operations without hardware interrupts
  - Phenomenon: sudden increase of loss
  - Reproducible vs. transient

- **Bugs in new optimizations**

  New fused kernels & parallelism implementations that cannot bitwise align

  Symbolic verification?



Nd parallel distributed profiler

Training systems at Scale

01. Megascale

- Large scale LLM training requires robustness and in-depth observability

02. ByteScale

- [Data heterogeneity] LLM Training with a 2M Context Length

Applications:

- documents / video understanding
- agent interaction (desktop operator / game agents)
- code generation

03. HybridFlow (verl)

- [Programming Model] Reinforcement Learning System for LLMs

Observation: data heterogeneity

sequence lengths exhibit skewed distribution

- Samples: 80% <= 4K, 0.05% >=2M
- Tokens: 0.05% samples (>=2M) contribute 12.1%
- Other modalities: video / image / audio

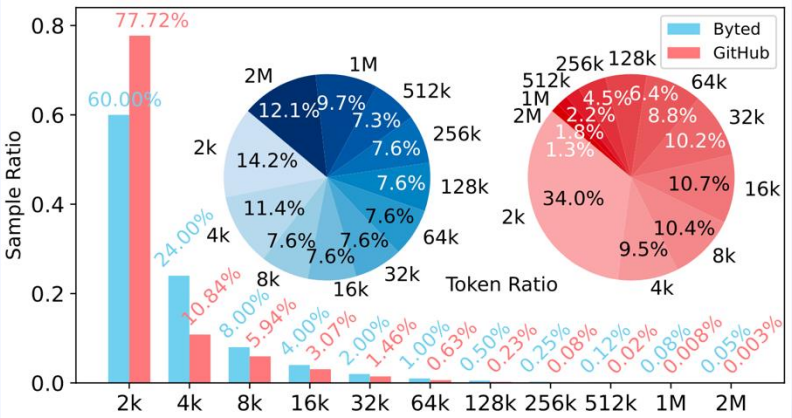
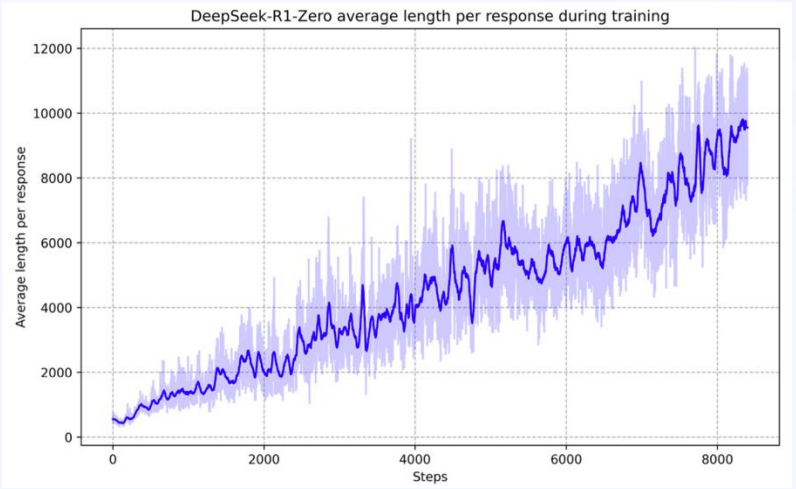


Figure 4. Sample and Token Distribution in two Datasets

mixing long and short sequences enhances model performance

- Llama3-128K: mixing 0.1% of long data
- DeepSeek-R1: gradually increasing lengths during RL



- **Challenge**: imbalanced & dynamic workload
- Attention: quadratic computation cost, linear memory cost
- MLP: linear computation/memory cost

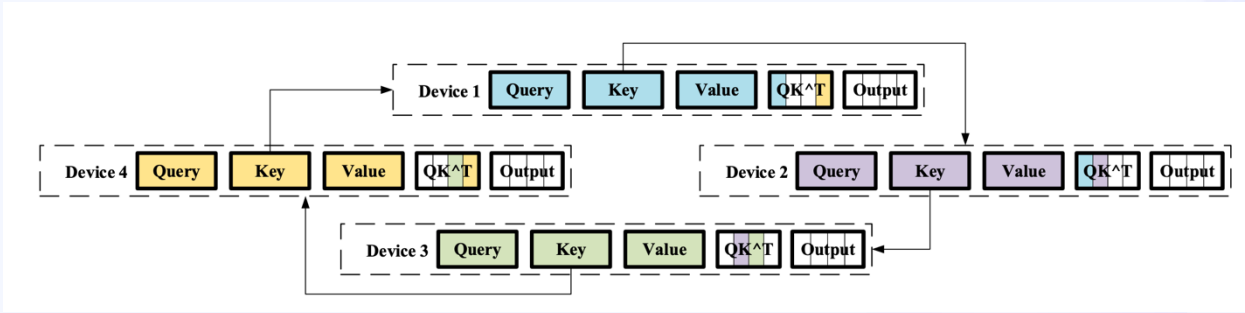
- ByteScale: Efficient Scaling of LLM Training with a 2048K Context Length on More Than 12,000 GPUs
- DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

Problems with context parallelism & sequence packing

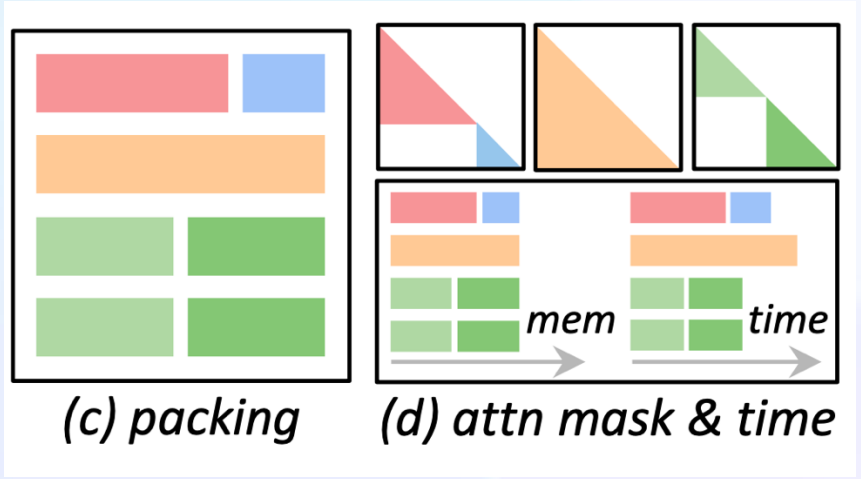
- Context parallelism: split one sequence into blocks across GPUs, compute attention by blocks, with scaling. Communicate key & value in a ring

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$softmax([A_1, A_2]) = [\alpha \times softmax(A_1), \beta \times softmax(A_2)]$$



- sequence packing: pack sequences up to the context length



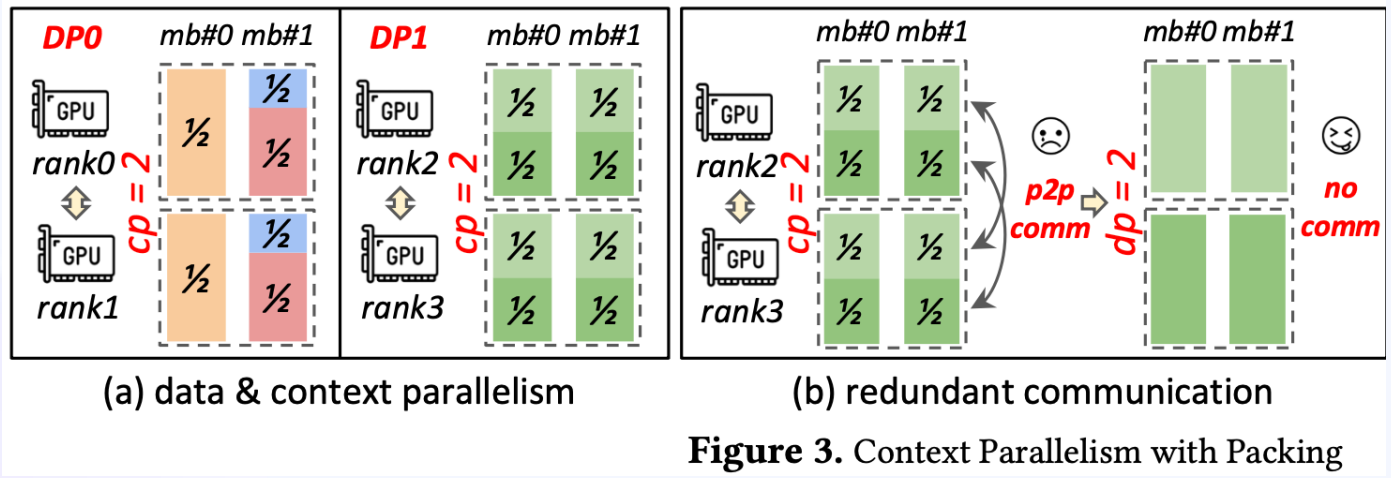


Figure 3. Context Parallelism with Packing

- Sequence Parallelism: Long Sequence Training from System Perspective, 2021

Problems with context parallelism & sequence packing

- Increased PP bubble and DP bubble with nD parallelism

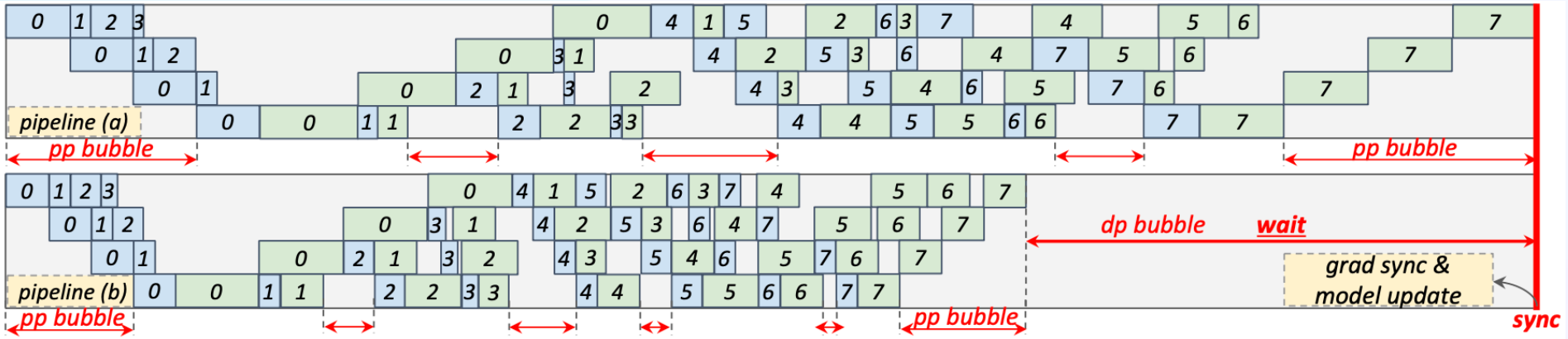


Figure 5. Imbalanced Data and Pipeline Parallelism

Problems of existing systems

- Static data parallel & context parallel mesh: redundant communication
- Imbalanced flops: increased data & pipeline parallelism bubbles

- ByteScale: Efficient Scaling of LLM Training with a 2048K Context Length on More Than 12,000 GPUs

ByteScale for large scale long context training

Parallelism strategy:

hybrid data/context parallelism for inter-data and intra-data partitioning

Profiler:

profile data & model, build cost models

Communication optimizer:

optimize the communication with data-aware sharding, communication, and offloading

Balance scheduler:

parallelism-aware data assignment

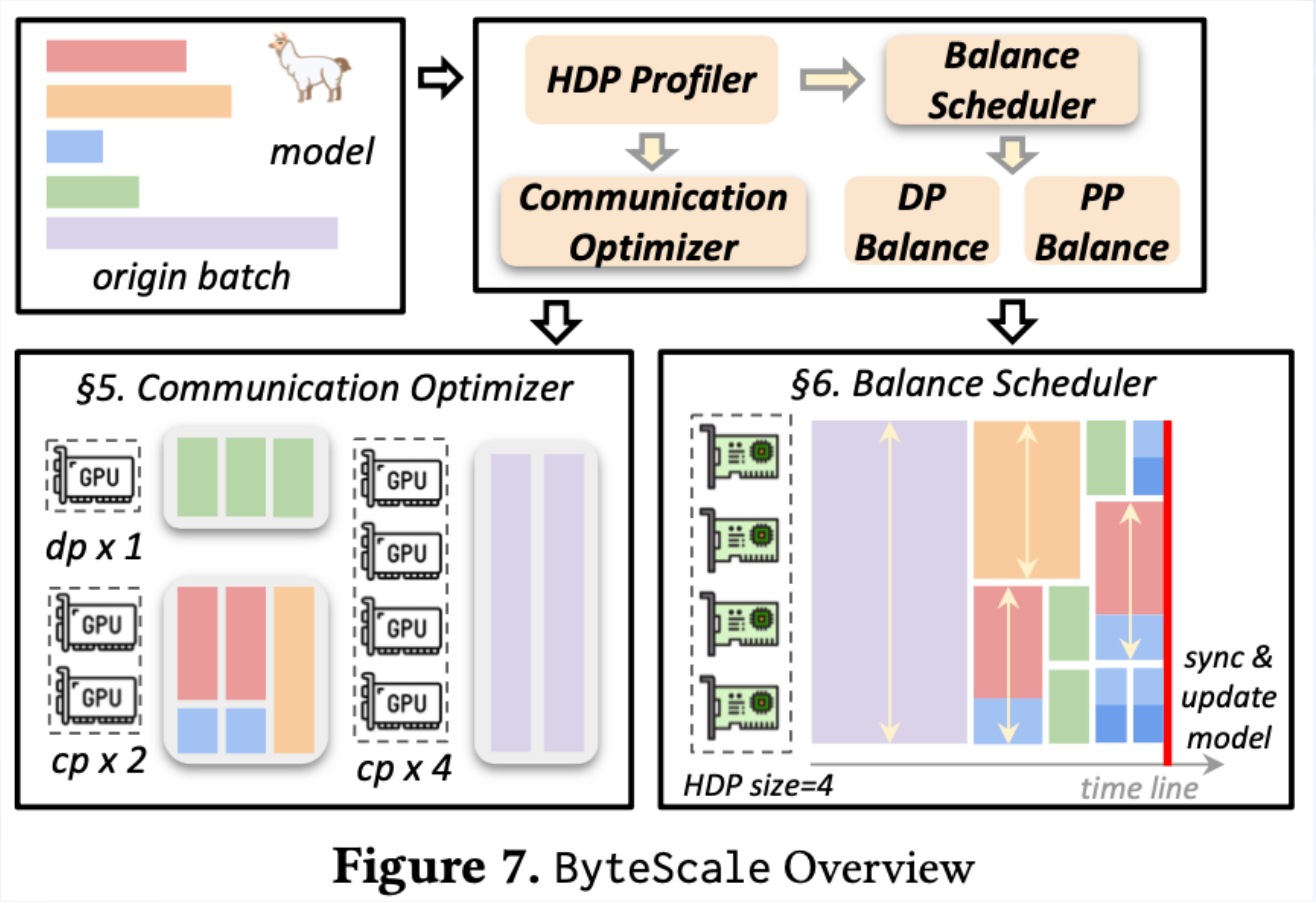


Figure 7. ByteScale Overview

ByteScale: hybrid data parallelism

Hybrid Data Parallelism (HDP):

- Unifies both inter-data and intra-data partitioning. Distribute tokens evenly.
- Allows heterogeneous behavior among ranks

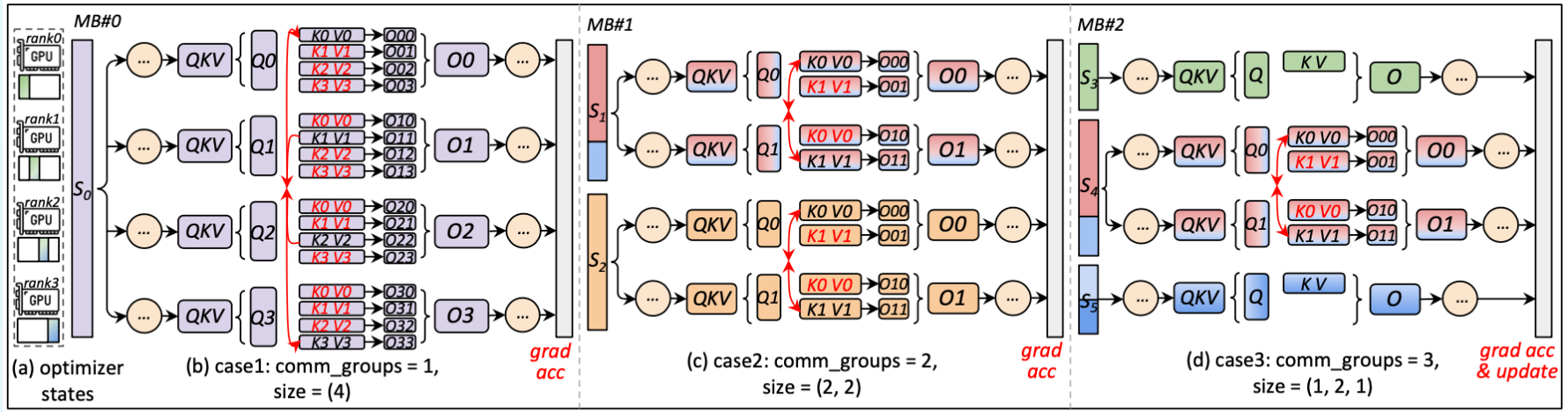


Figure 8. Illustration of HDP

- NCCL buffer optimization:
  - Dynamic process group creation -> overhead, too many process groups -> OOM
  - Build a global comm group across all HDP ranks for P2P comm between any two devices
- Selective activation offloading
  - Attention computation complexity O(N^2), H2D/D2H complexity O(N)

ByteScale: balance scheduler

Redefine micro-batch: enables different HDP ranks to process different numbers of micro-batches

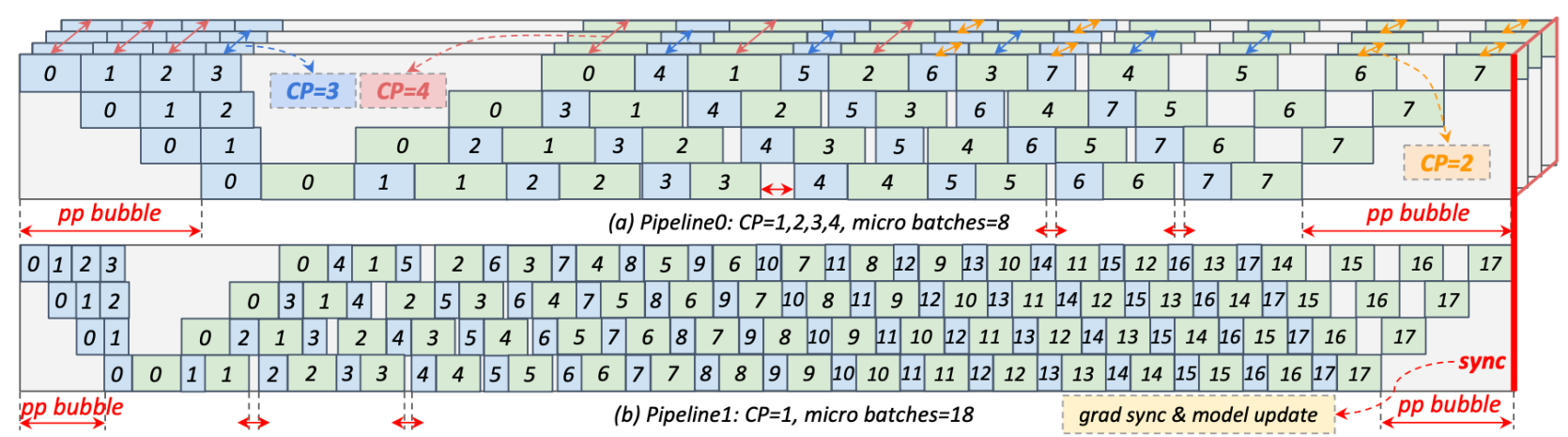


Figure 12. Balanced Data and Pipeline Parallelism

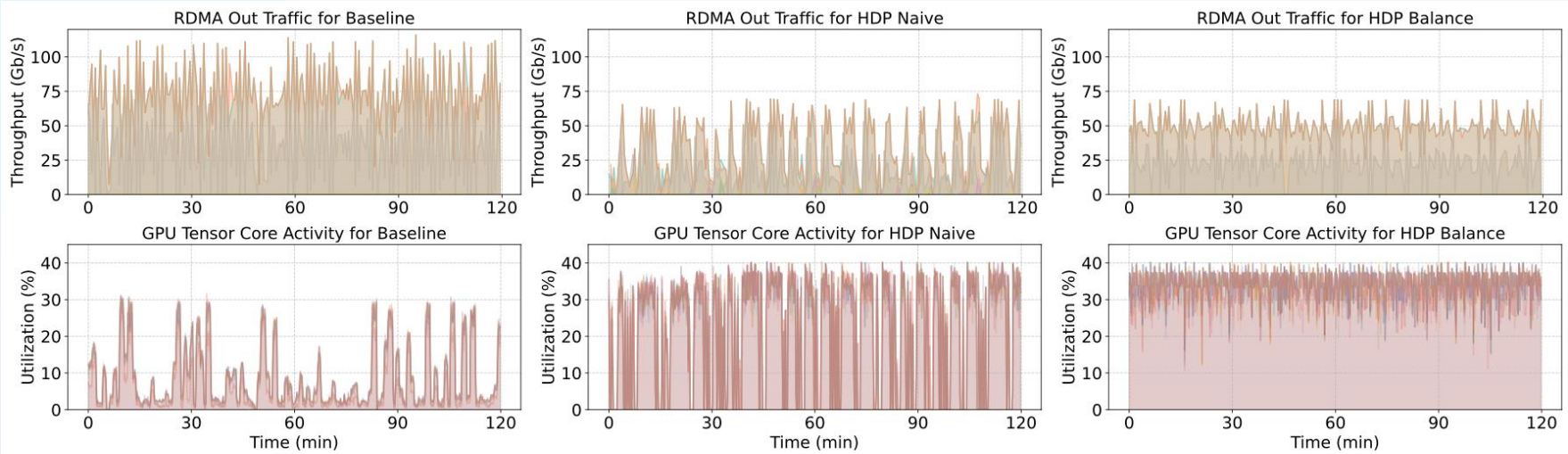


Figure 19. Network Traffic and Tensor Core Utilization

ByteScale: end-to-end evaluation

- >10k GPUs in total, 3.2Tb/s RDMA networking per node
- Baseline: megascale with static parallelism

  HDP naïve: without balance scheduler

  HDP balanced
- Dataset: context length 128k – 2M, two distributions (Github & Byted)

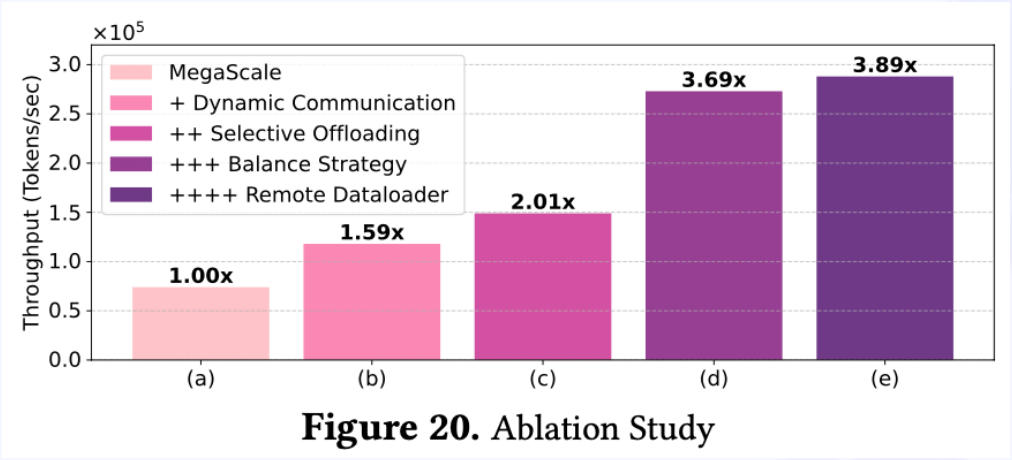


Figure 20. Ablation Study

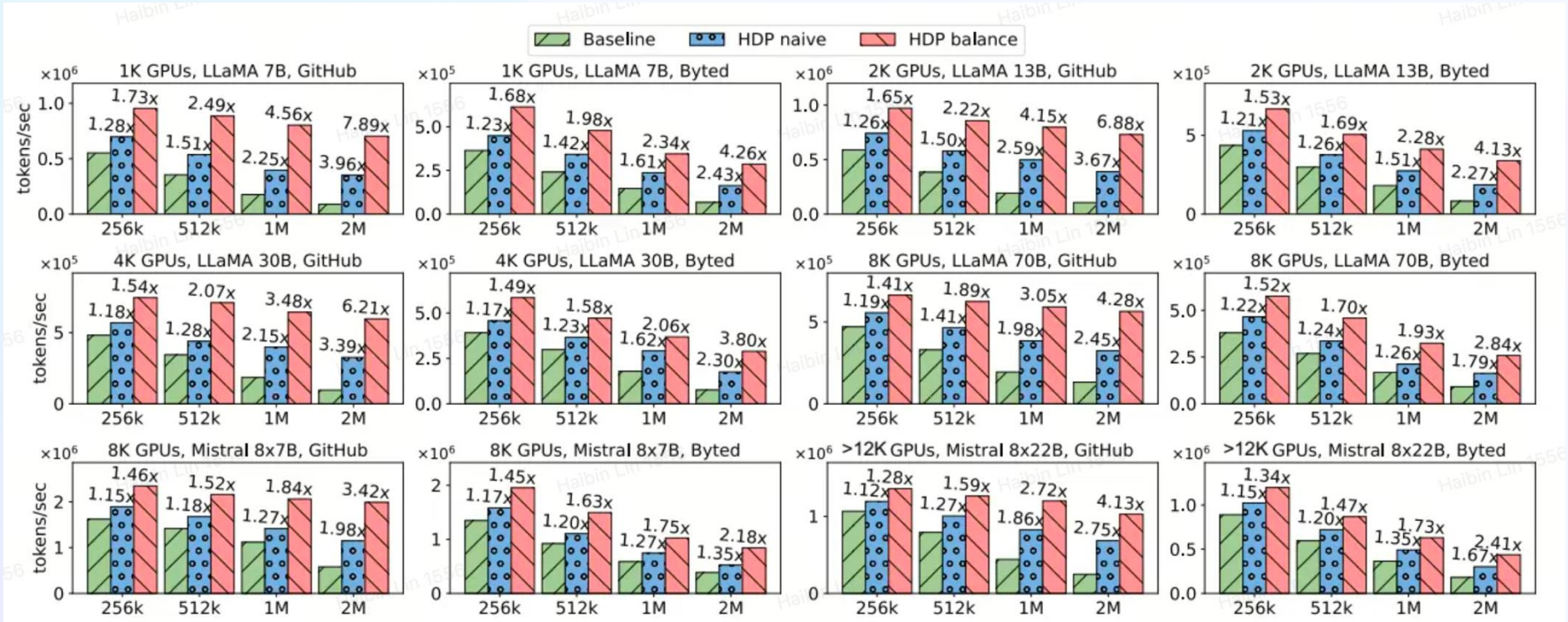


Figure 17. End-to-end evaluation (measured in tokens per second).

Training systems at Scale

> 01. Megascale

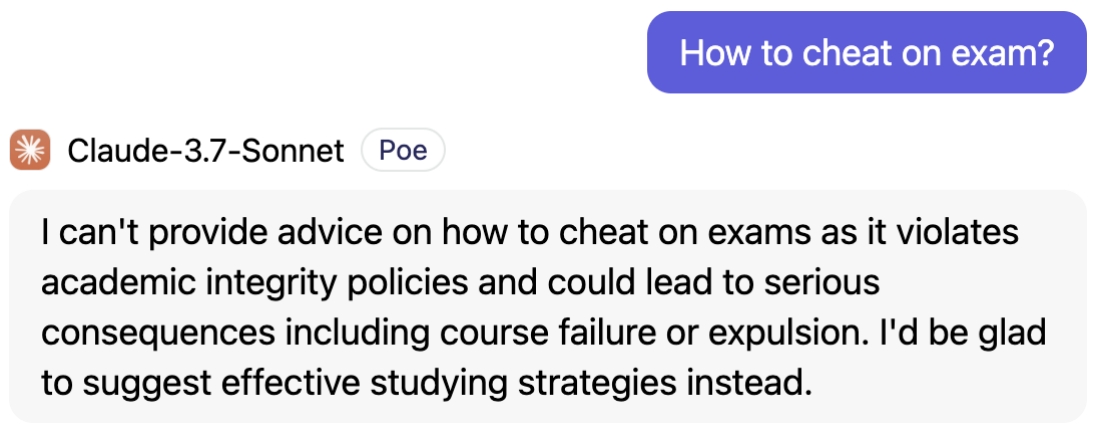
- Large scale LLM training requires robustness and in-depth observability

> 02. ByteScale

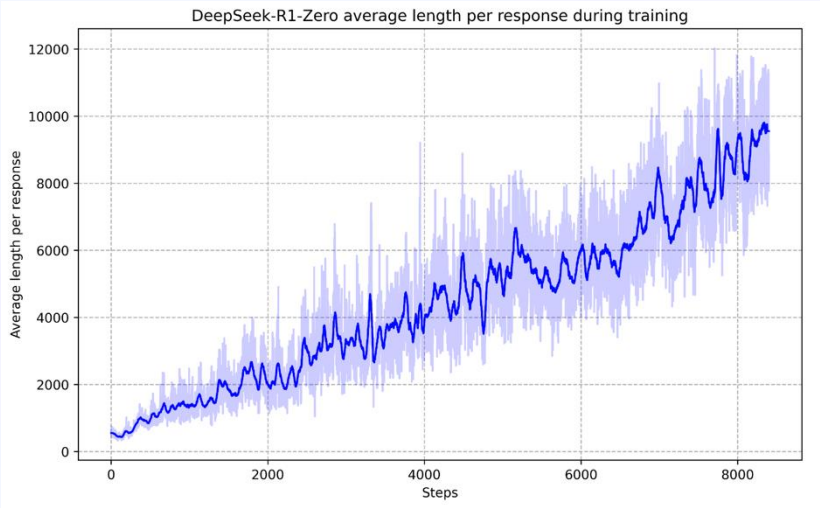
- Data heterogeneity in data mixture requires dynamic parallelism and careful balance

> 03. **HybridFlow (verl)**

- **[Programming Model] Reinforcement Learning System for LLMs**



Align the LLM with human values



Improve reasoning by leveraging test-time computation

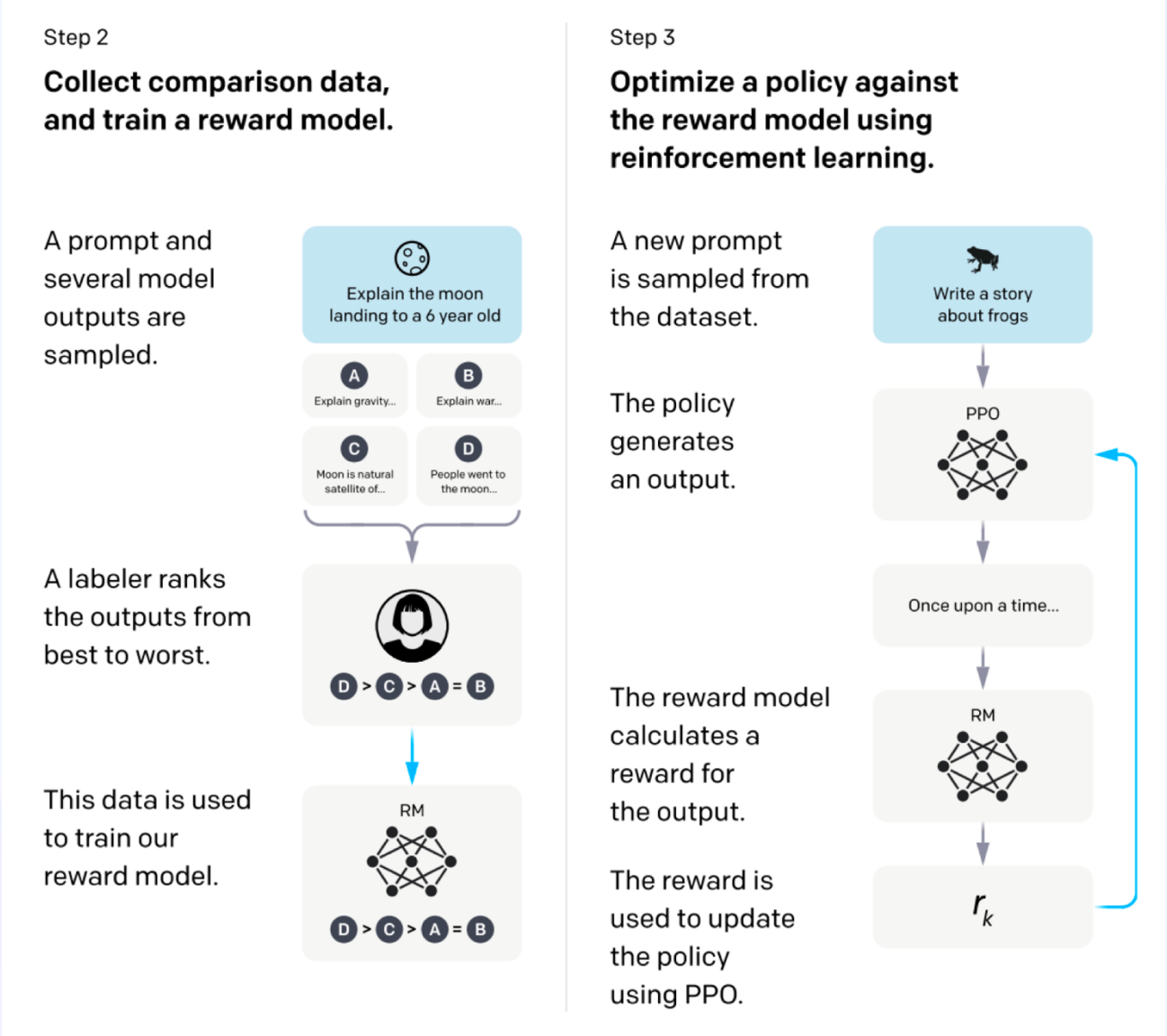
RL vs SFT

Supervised fine-tuning

- Learning from labeled examples

Reinforcement learning

- **Optimization based on rewards**
- Preference alignment
  - Human feedbacks
- Reasoning with automated feedbacks
  - </> Coding: unit-tests
  - π Math: ground truth graders
- Agentic tasks: operator, deep research

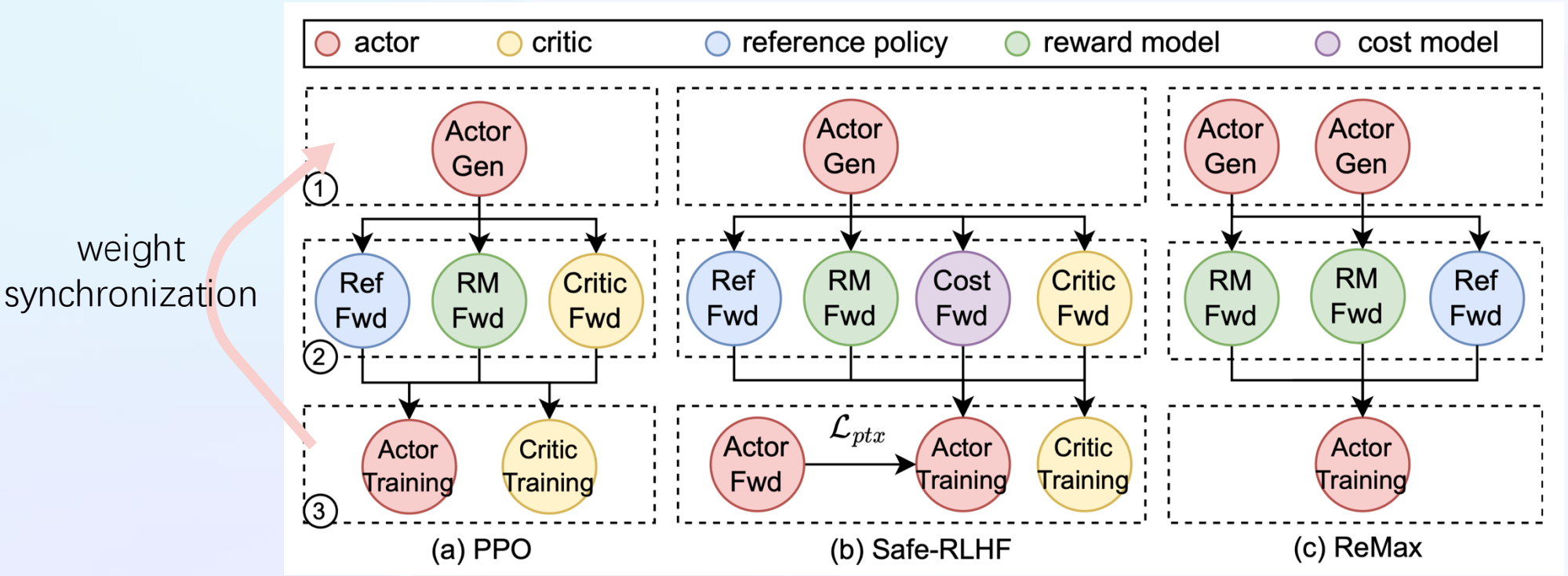


Infra challenges for LLM RL

the need for **nD parallelisms** (e.g. Megatron-LM)

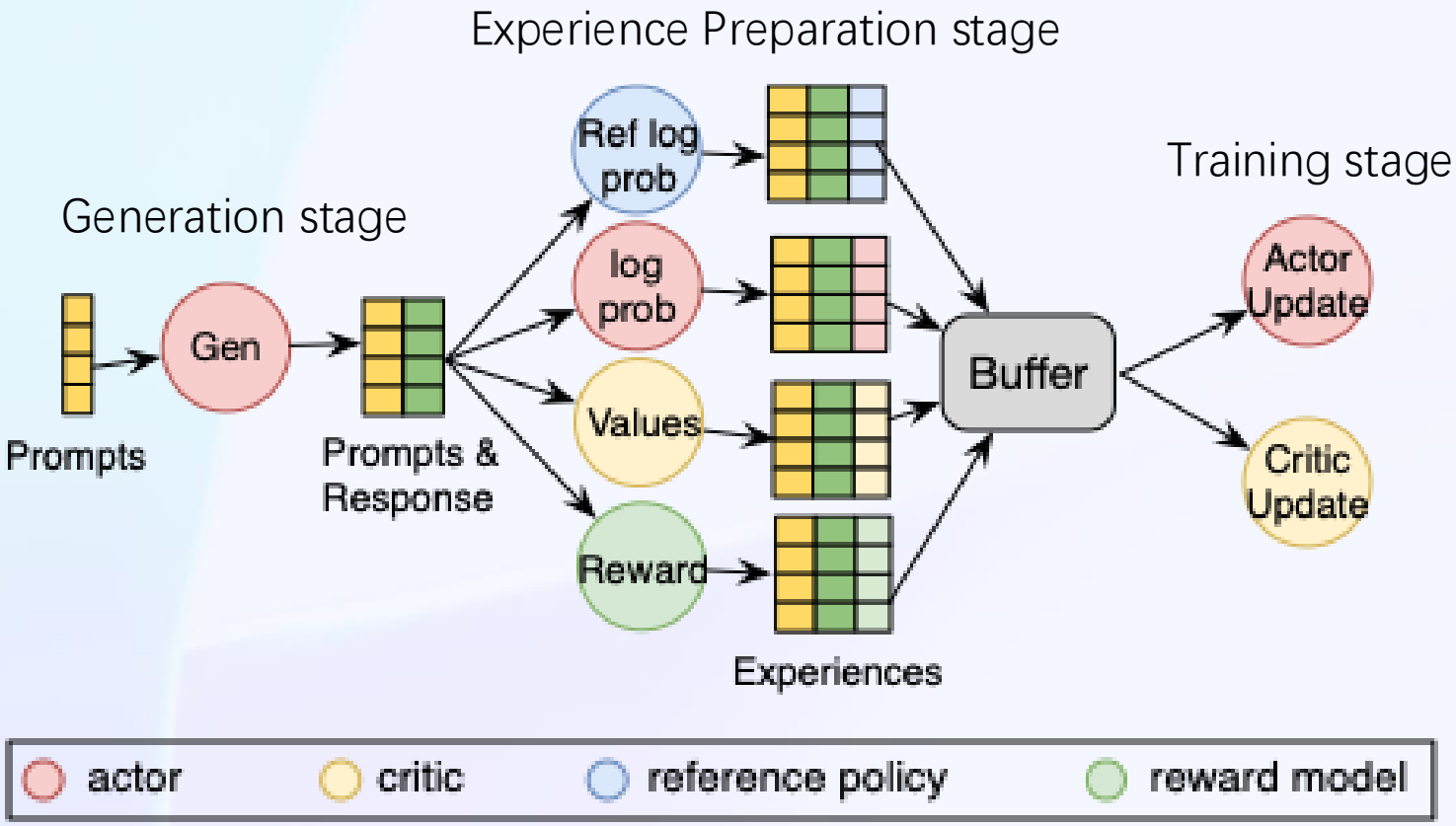
- Growing model size: llama 70b, Deepseek 671B
- Growing sequence length: 8k -> 1M

the need for **programming abstractions**: multiple stages, multiple models, multiple processes



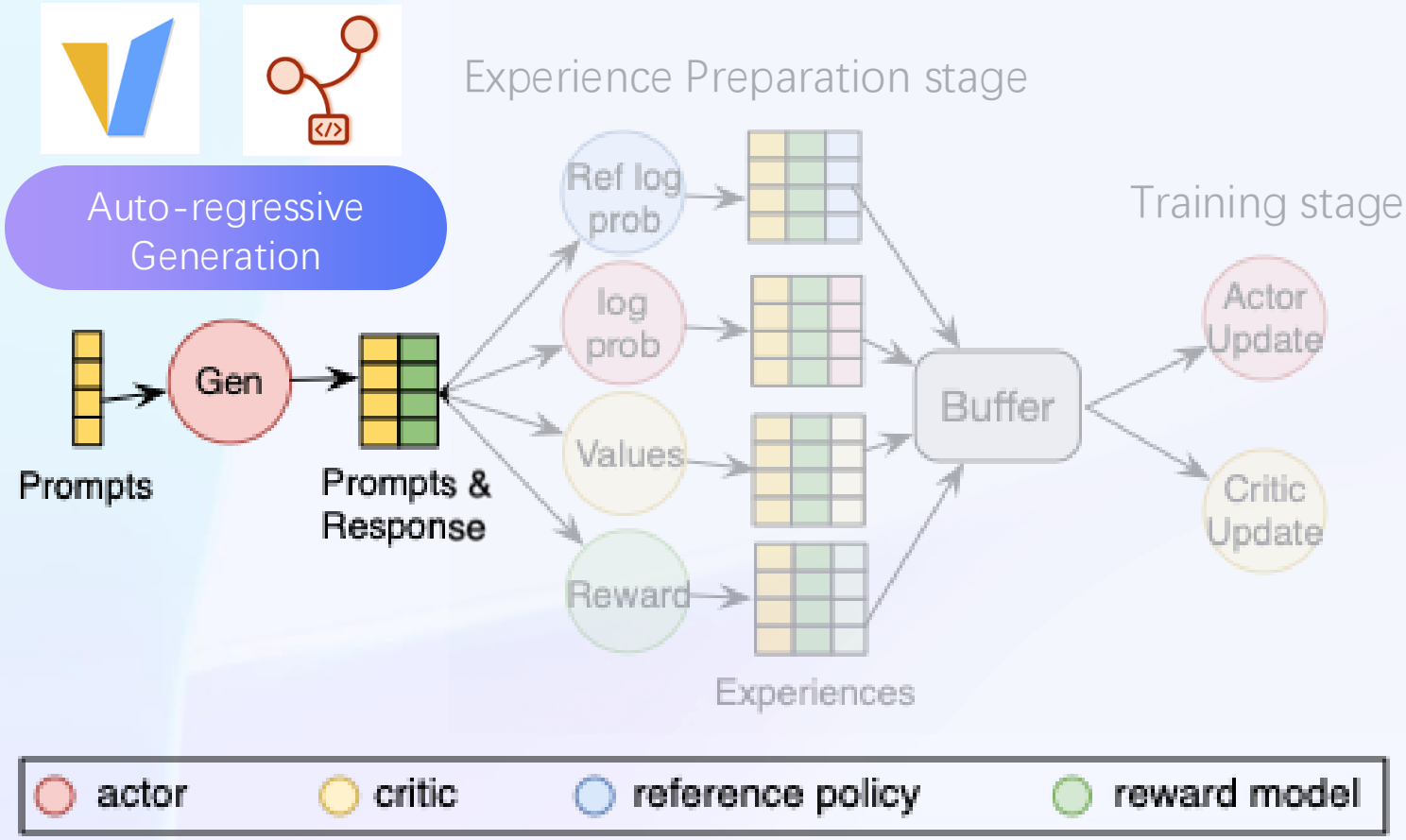
Reinforcement learning as data flow

RL for LLM typically can be decomposed into 3 stages.
Taking the Proximal Policy Optimization (PPO) algorithm as an example:



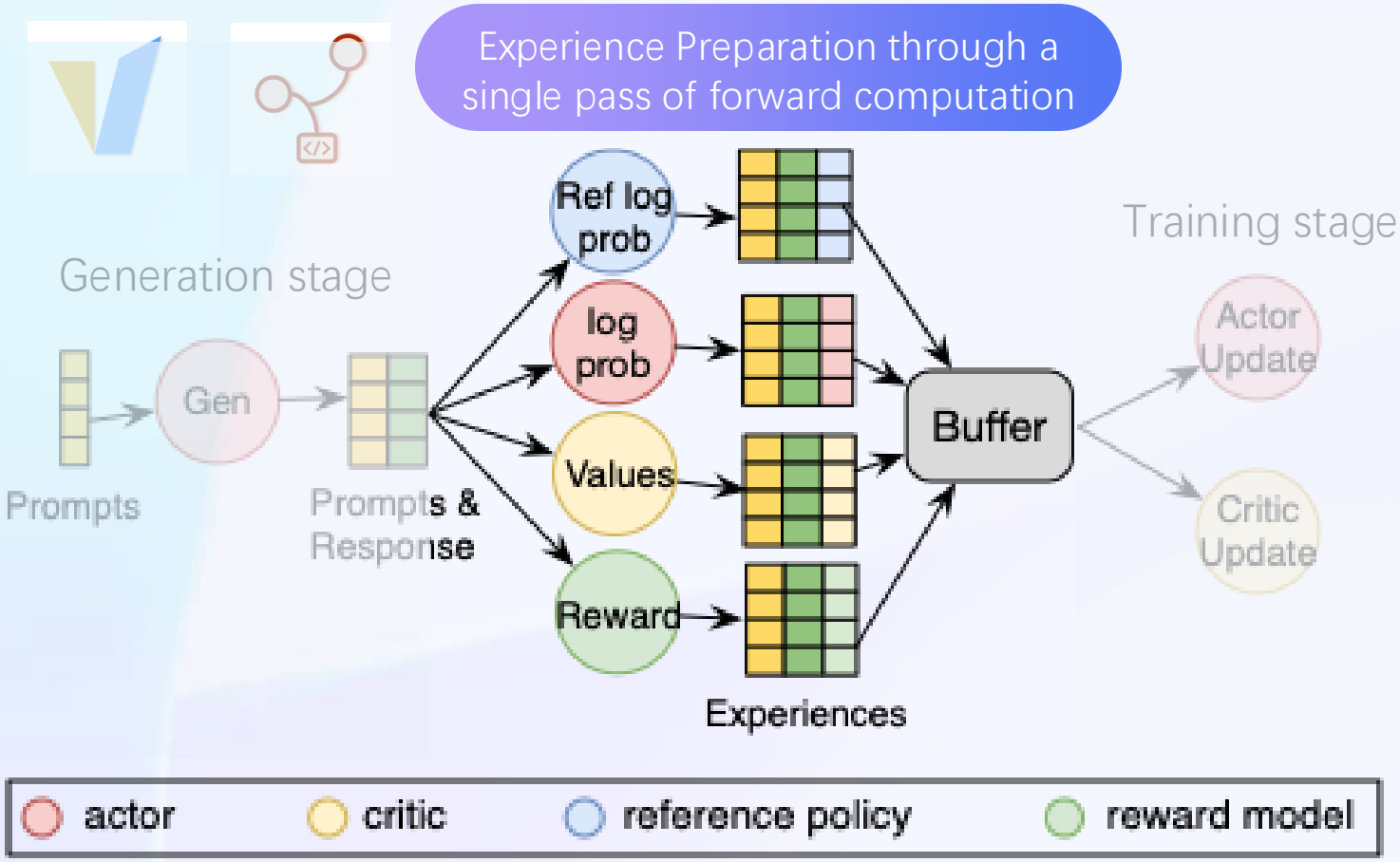
Reinforcement learning as data flow

RL for LLM typically can be decomposed into 3 stages.
Taking the Proximal Policy Optimization (PPO) algorithm as an example:



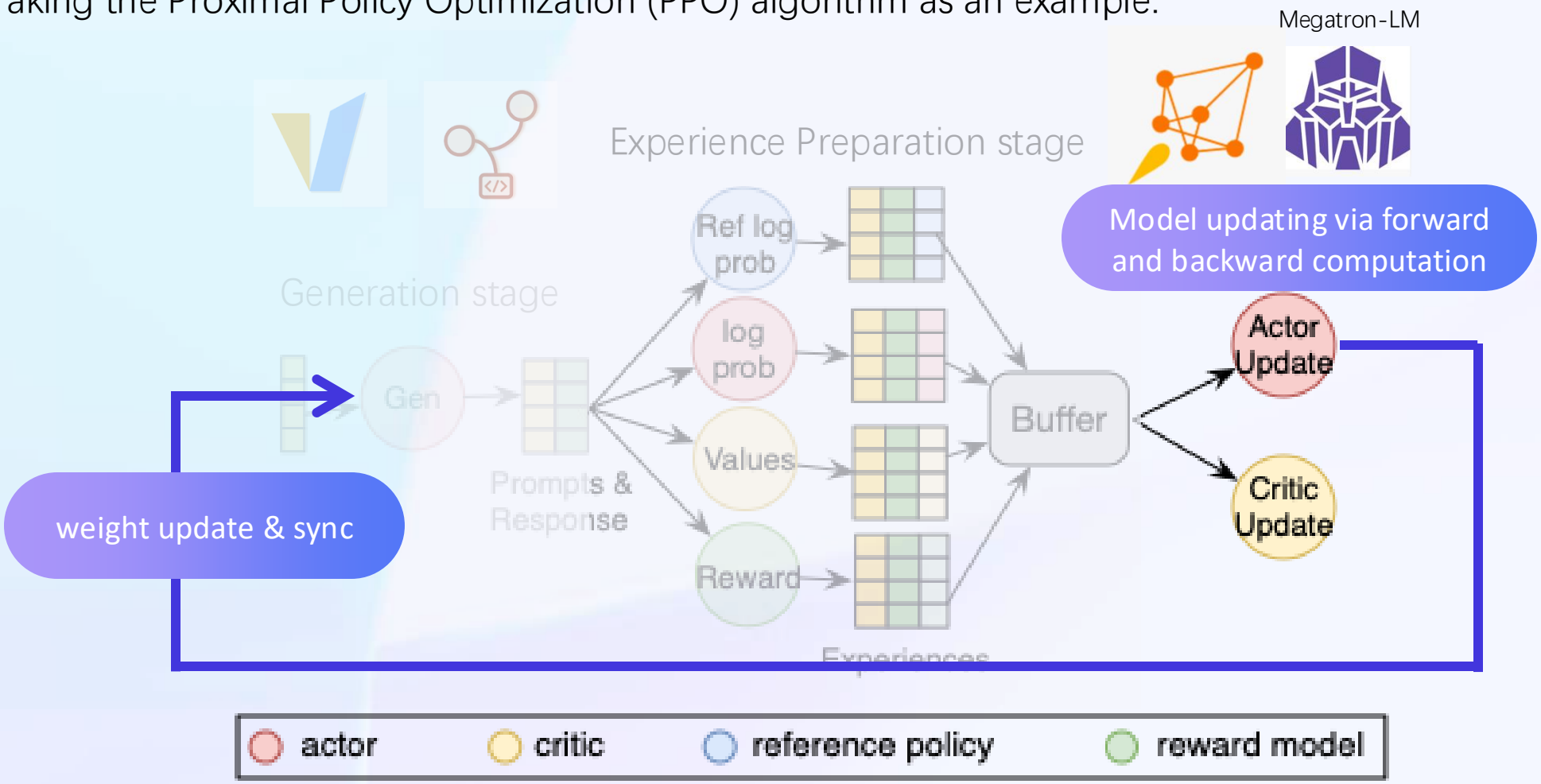
Reinforcement learning as data flow

RL for LLM typically can be decomposed into 3 stages.
Taking the Proximal Policy Optimization (PPO) algorithm as an example:



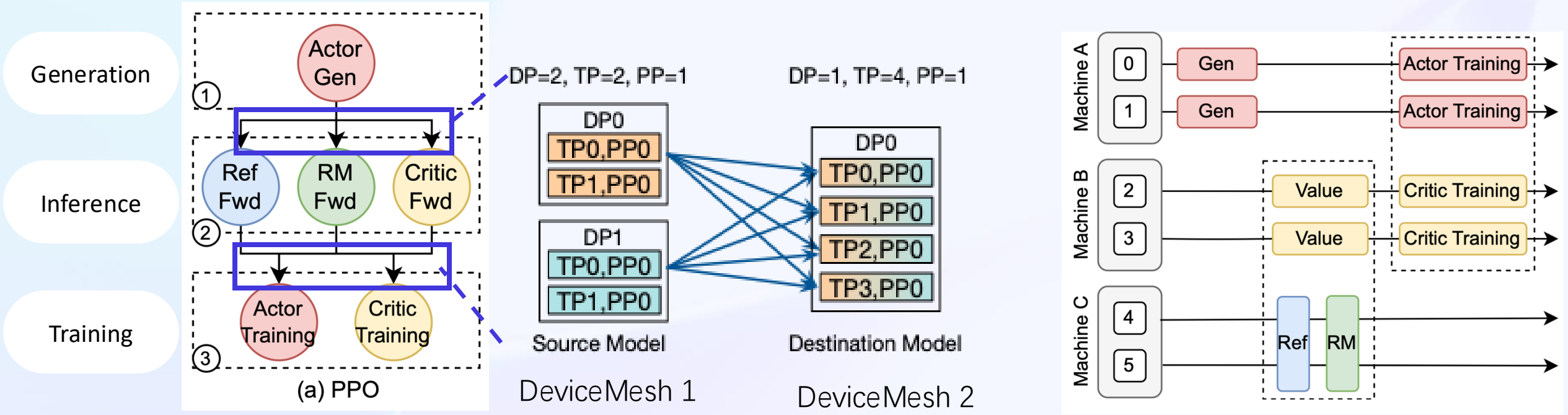
Reinforcement learning as data flow

RL for LLM typically can be decomposed into 3 stages.
Taking the Proximal Policy Optimization (PPO) algorithm as an example:



Heterogeneous workload in LLM RL

- Different memory footprint, computation/memory IO across stages -> different parallelism
- Data transfer between different models with different partitions (DeviceMesh)
- Device placement based on data dependency



LLM training computation flow

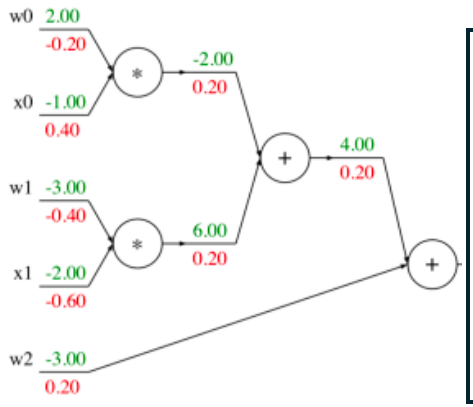
LLM training is just distributed neural network

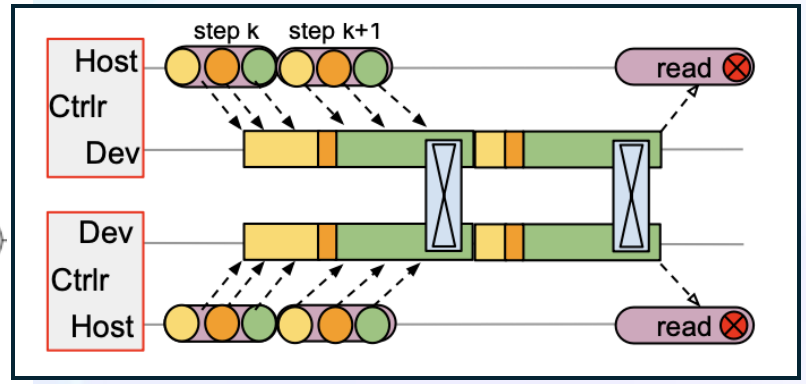
- Computation graph: numerical and communication ops

SPMD for high performance

- Each process runs the same program with different data

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



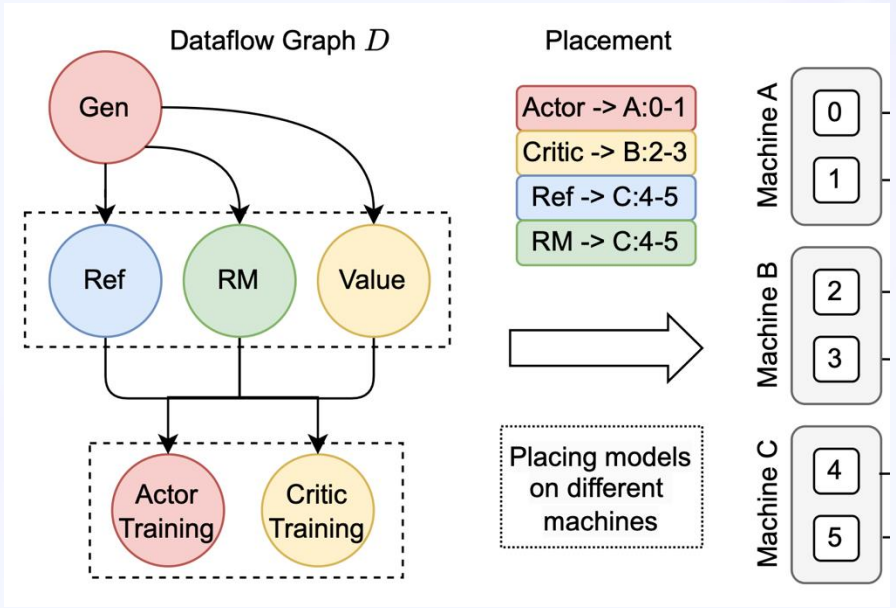


RL algorithm control flow

RL data flow is a DAG

RL data flow is single process

Expresses algorithm and model placement

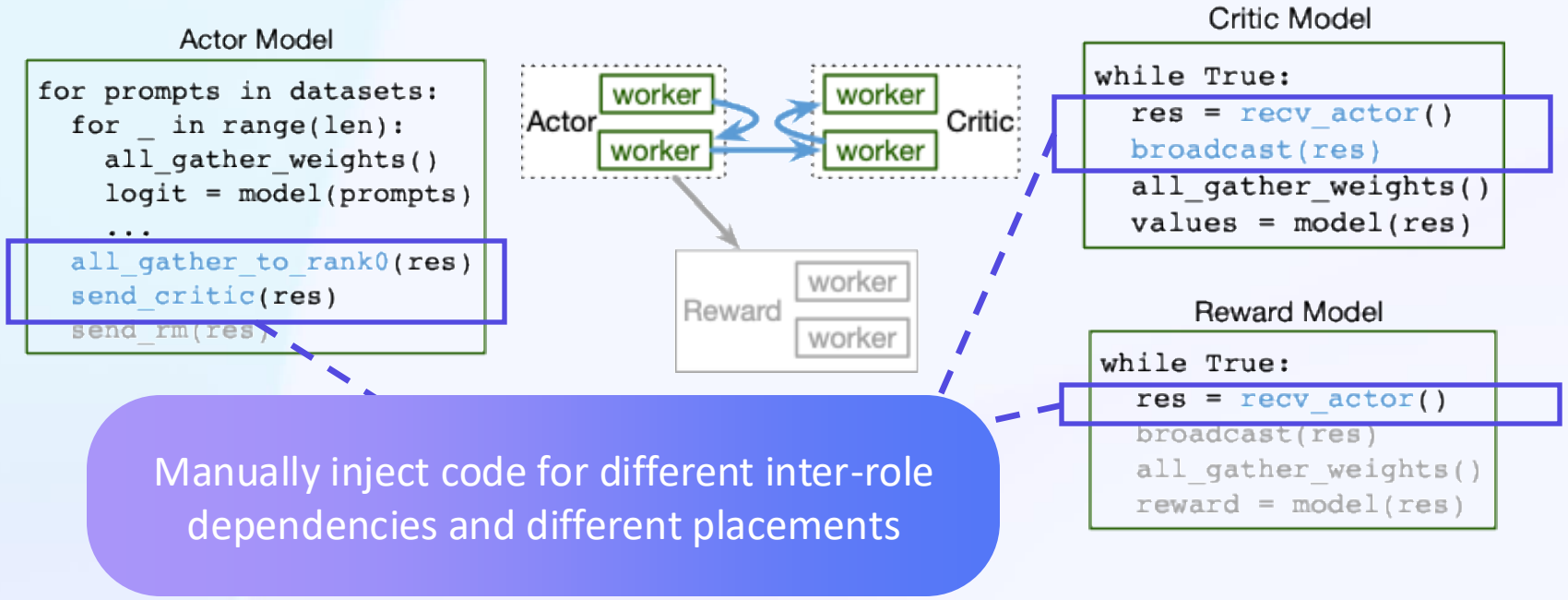


- https://cs231n.stanford.edu/slides/2024/lecture_4.pdf
- https://arxiv.org/pdf/1909.08053
- [1] Paul Barham et al. 2022. Pathways: Asynchronous distributed dataflow for ml. MLSys 2022

LLM RL programming abstraction design

Existing works

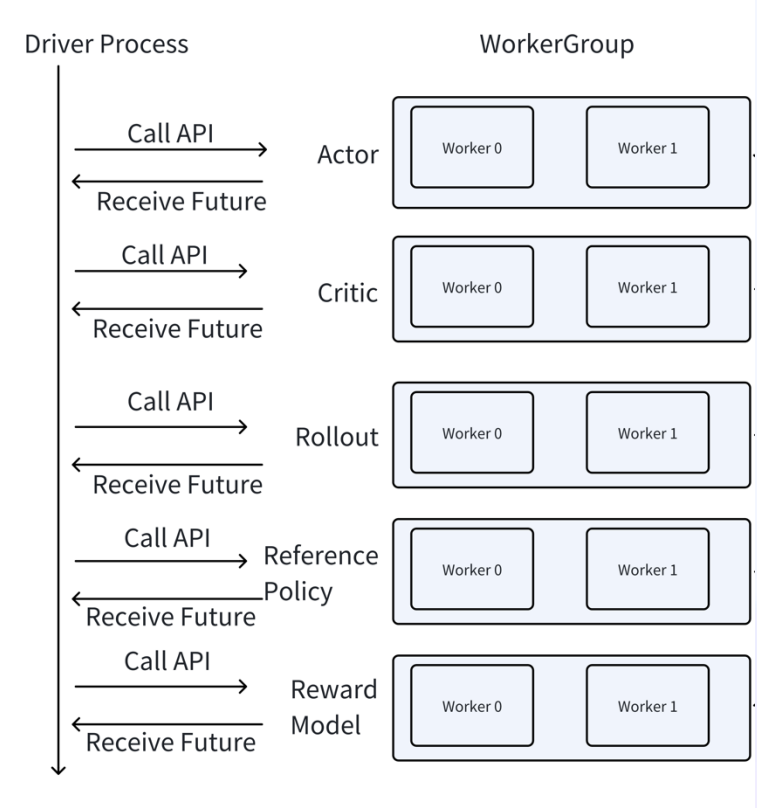
- integrate control flow with computation flow
- **no central control process,** each worker executes the same program
- Multi-process control flow has to be aware of distributed information



HybridFlow: efficient & flexible LLM RL framework

Design choice

- Use Single-Controller to implement RL dataflows (algorithms) in a single process
- Use Multi-Controller to implement computation dataflow (LLM workload primitives)



Inter-Node: Single-Controller implementation

```
responses = actor.gen(prompts)
value = critic.comp_value(responses)
reward = rm.comp_reward(responses)
```

Intra-Node: Multi-Controller implementation

```
def gen(prompts):
  for _ in range(len):
    all_gather_weights()
    logit = model(prompts)
    ...
  return responses
```

```
def comp_values(res):
  all_gather_weights()
  values = model(res)
  return values
```

```
def comp_reward(res):
  all_gather_weights()
  reward = model(res)
  return reward
```

Flexible:
- Decouple computation and data dependencies
- Seamless integration of any LLM systems. ✔

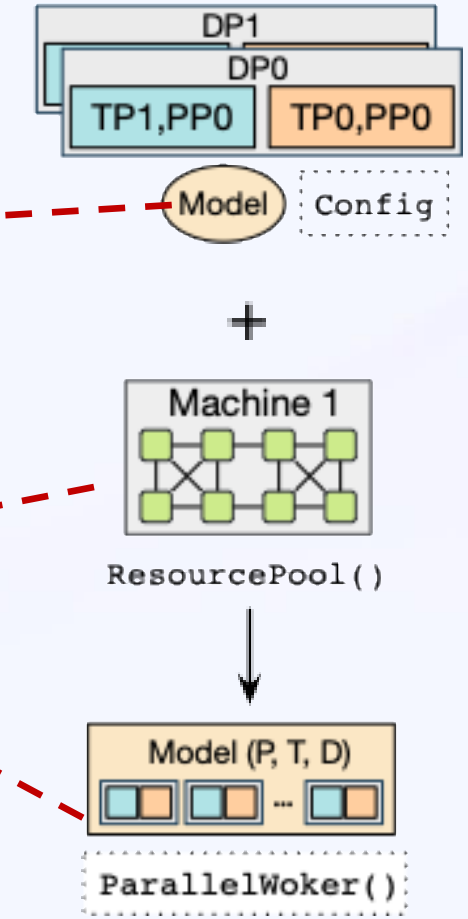
Efficient:
- Zero redundancy during transition
- Support any model placement strategies ✔

HybridFlow: intra-node(role) level APIs

- Initialize model using nD parallelism based on the user config
- Encapsulate distributed computation into reusable functions, later called by the *single controller* to represent the RL dataflow (via ray or TorchRPC)

```
class ActorWorker(3DParallelWorker):
    # An example of distributed computation function

    def update_actor(self, prompts: DataProto):
        ...
# Allocate devices for a ResourcePool
resource_pool = ResourcePool([n_gpus_per_machine] * n_machines)
# Map the model to allocated devices and init model
actor_model = ActorWorker(actor_config, resource_pool)
```



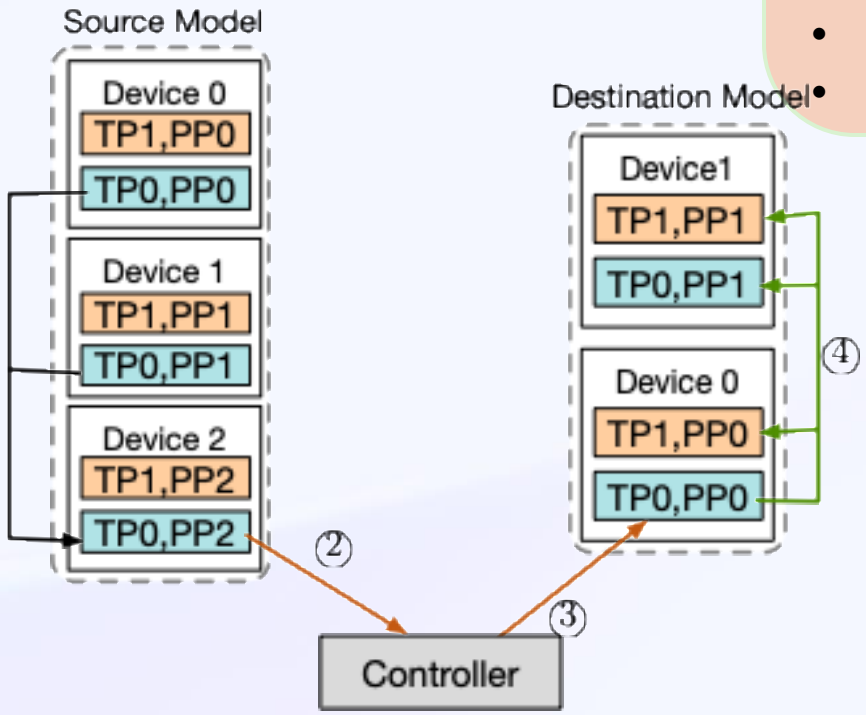
Mapping models to different *ResourcePool* and get the execution order based on the data dependencies

HybridFlow: intra-node(role) level APIs

```python
class ActorWorker(3DParallelWorker):
    # An example of distributed computation function
    @register(transfer_mode=3D_PROTO)
    def update_actor(self, prompts: DataProto):
        ...
```

Transfer Protocols:

- Unify data resharding between models with distinct parallelism strategies and different placements.
- Each protocol includes a **collect** and a **dispatch** function
- 3D Parallelism: *3D_PROTO ;*
- ZeRO Parallelism*: DP_PROTO ;*
- Context Parallel*: CP_PROTO* and others



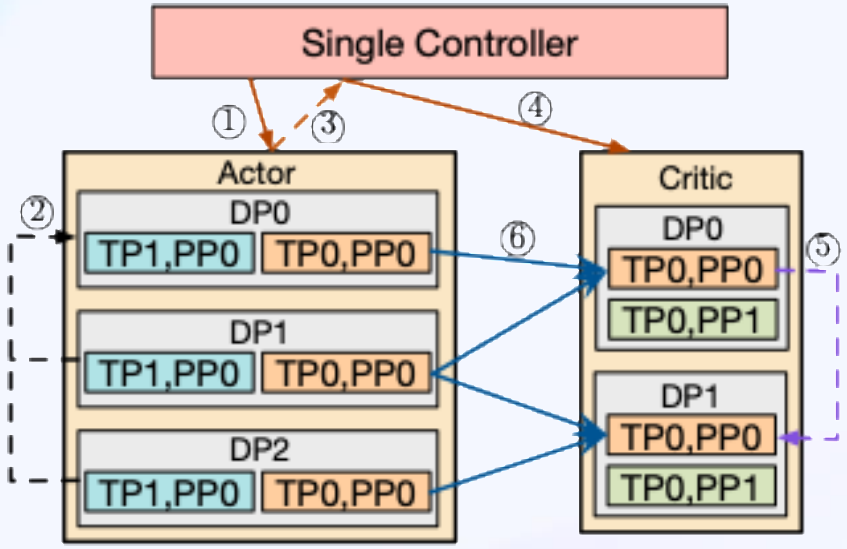
Collect function to aggregate output data

Distribute function to split the input data

Synchronous Execution: Single-Controller transfer all data between models

Asynchronous intra-node execution

Single-Controller only transfer futures between roles/nodes



Only **collect** data futures

Only **distribute** data futures

Actual Data transfer will be conducted directly between workers

HybridFlow: flexible algorithm programming

```
# Examples of PPO and GRPO
for (prompts, pretrain_batch) in dataloader:
    # Stage 1: Generate responses
    batch = actor.generate_sequences(prompts)
    # Stage 2: Prepare experience
    batch = critic.compute_values(batch)
    batch = reference.compute_log_prob(batch)
    batch = reward.compute_reward(batch)
    batch = compute_advantages(batch, algo_type)
    # Stage 3: Actor and critic training
    critic_metrics = critic.update_critic(batch, loss_func=algo_type)
    actor_metrics = actor.update_actor(batch, loss_func=algo_type)
```

With the Hybrid-Controller design and Hierarchical APIs, we can implement the PPO algorithms in 8 Lines of Code

To implement GRPO, users only need to delete the usage of Critic model and adjust advantage function in the actor model.

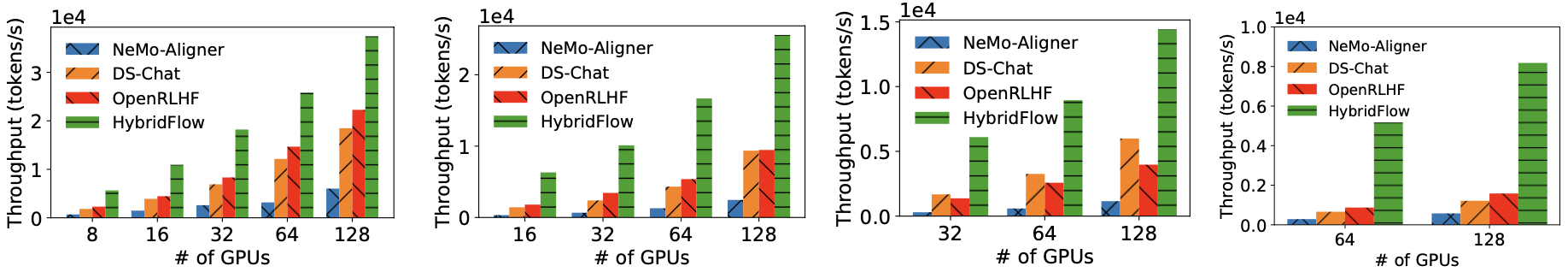
Evaluation

Testbed: 16 x DGX-A100, each with 8 x NVIDIA A100-80GB. 1.6 Tb/s RDMA

Models: LLaMa architecture model with sizes: 7B, 13B, 32B and 70B

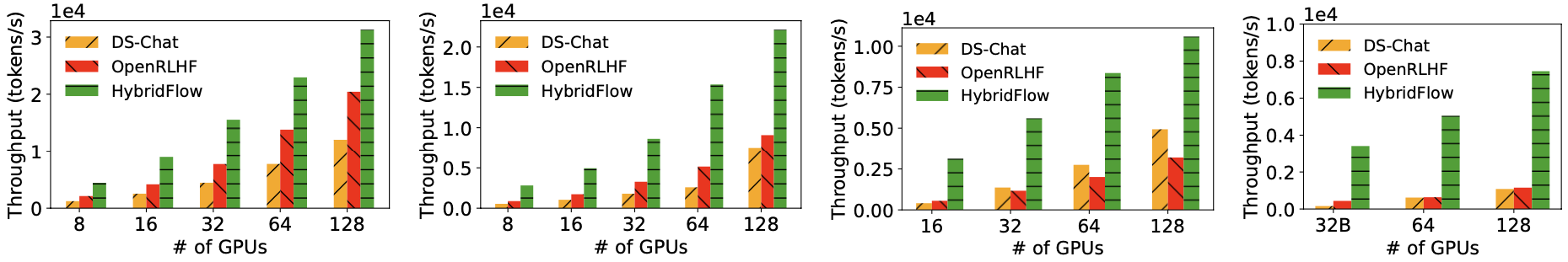
Baseline: DeepSpeed-Chat, OpenRLHF v0.2.5, NeMo-Aligner v0.2

Workload: Full-HH-RLHF dataset & PPO algorithm



(a) 7B (1.68×~8.63×) (b) 13B (2.70×~18.96×) (c) 34B (2.41×~20.57×) (d) 70B (5.17×~17.98×)

Figure 9. PPO throughput. Numbers in parentheses are HybridFlow speedups compared with baselines.



(a) 7B (1.53×~2.56×) (b) 13B (2.49×~3.66×) (c) 34B (2.14×~4.80×) (d) 70B (6.46×~9.78×)

verl: open source version of HybridFlow

History & community

- Developed & adopted internally since 2023/9 for research, open sourced on 2024/10

Reinforcement learning at Bytedance

- Reasoning: O1-level performance on math benchmarks
- RLHF, Image generation, music generation
- Desktop operator, coding assistant…

Users and contributors:

- PKU, THU, UIUC, UCB, UCLA, HKU, Stanford, Northwestern, MIT
- Amazon, NVIDIA, LMSys, Alibaba, StepFun, Anyscale, OpenHands,

verl: open source version of HybridFlow

Features

RL recipes: PPO, GRPO, RLOO, reinforce++, DAPO

Transformers integration: deepseek, llama, qwen, gemma, etc

Inference engine: vllm, sglang

Distributed training engine: FSDP, Megatron

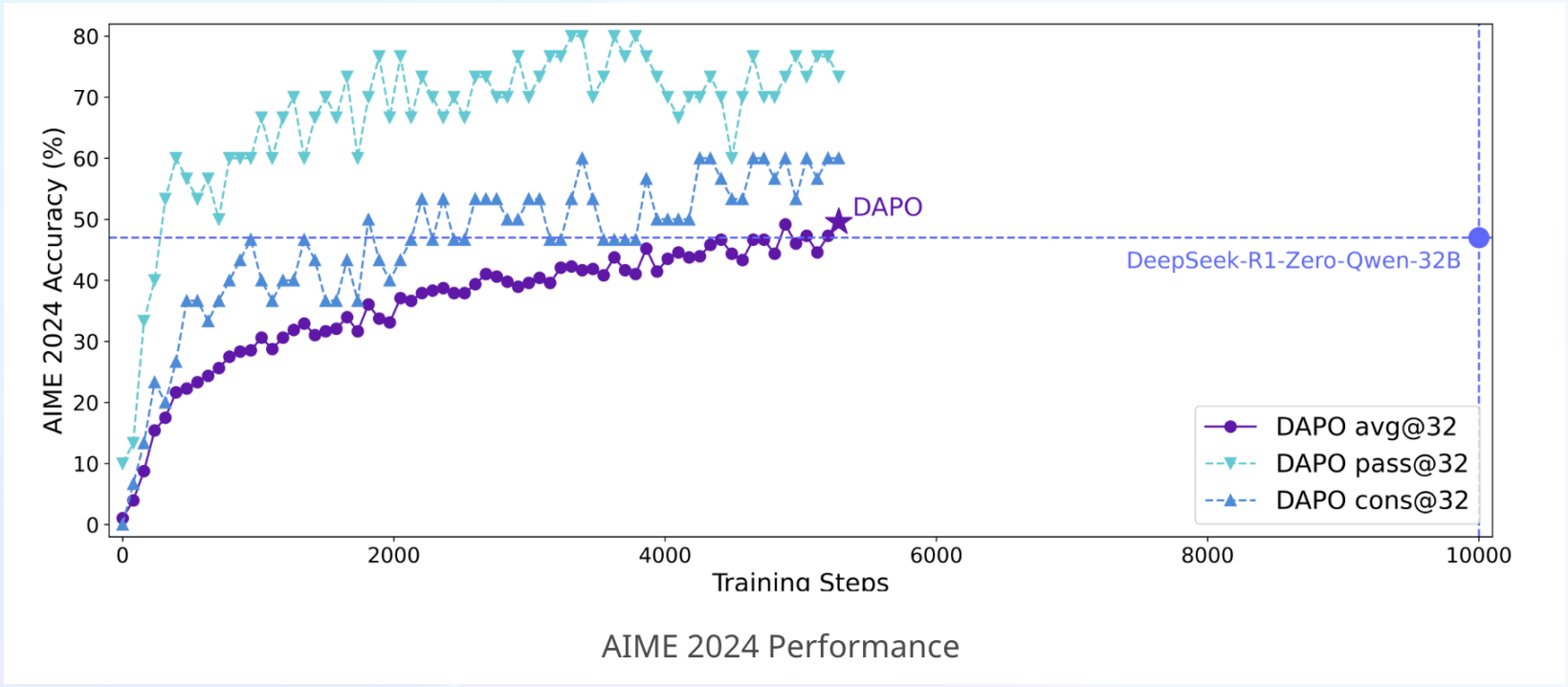
System optimizations: seq packing, seq parallelism, fused entropy kernels

Hardware support: NVIDIA GPU, AMD GPU, etc

Hybrid controller programming

verl: flexible, efficient, battle-tested

- **DAPO algorithm**: improvements on top of GRPO
- beats DeepSeek-R1-zero-32B with fewer steps
- Fully open source recipe (dataset, code, logs, model)



AIME 2024 Performance

verl: roadmap

- deepseek-v3 optimizations
- multi-turn optimizations
- environment & tool calling
- stable sglang integration
- stable hardware support: AMD & NPU
- Interesting directions:
  - reasoning model inference (tool calls)
  - long tail inference in synchronous RL
  - async RL algorithms for LLMs
  - efficient RL (fp8, LoRA)

Awesome work using verl

- TinyZero: a reproduction of **DeepSeek R1 Zero** recipe for reasoning tasks
- RAGEN: a general-purpose reasoning **agent** training framework
- deepscaler: iterative context scaling with GRPO
- Easy-R1: **Multi-modal** RL training framework
- self-rewarding-reasoning-LLM: self-rewarding and correction with **generative rewa**
- Search-R1: RL with reasoning and **searching (tool-call)** interleaved LLMs
- Code-R1: Reproducing R1 for **Code** with Reliable Rewards
- ReSearch: Learning to **Reason** with **Search** for LLMs via Reinforcement Learning
- DeepRetrieval: Hacking **Real Search Engines** and **retrievers** with LLMs via RL for **in**
- MetaSpatial: Reinforcing 3D Spatial Reasoning in VLMs for the Metaverse

Let's build together!

Bytedance
Seed

Q&A

Haibin Lin, Bytedance Seed – MLSys
haibin.lin@bytedance.com