StreamRL: Scalable, Heterogeneous, and Elastic RL for LLMs with Disaggregated Stream Generation

Author: Yinmin Zhong[1] Zili Zhang[1] Xiaoniu Song[2] Hanpeng Hu[2]
Chao Jin[1] Bingyang Wu[1] Nuo Chen[2] Yukun Chen[2] Yu Zhou[2]
Changyi Wan[2] Hongyu Zhou[2] Yimin Jiang[3] Yibo Zhu[2] Daxin Jiang[2]

[1] School of Computer Science, Peking University [2] StepFun [3] Unaffiliated

arXiv : 2504.15930

Presented by Muxin Liu

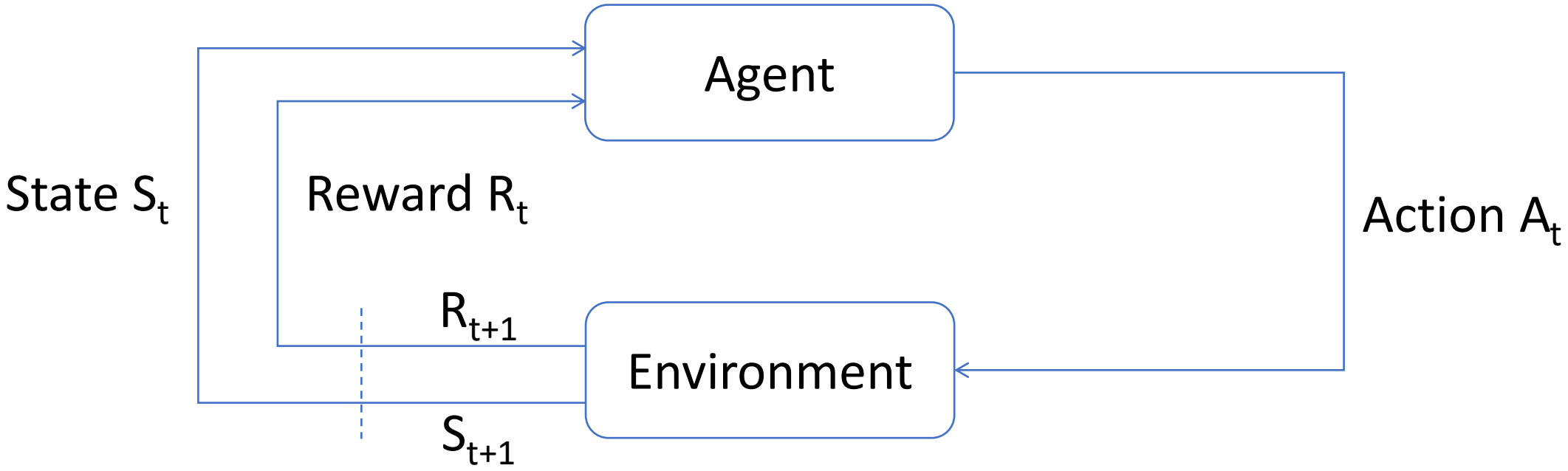
USTC, CHINA ADSLAB 先进数据系统实验室

Outline

- Background
- Challenges for Disaggregation
- Design
- Evaluation
- Conclusion

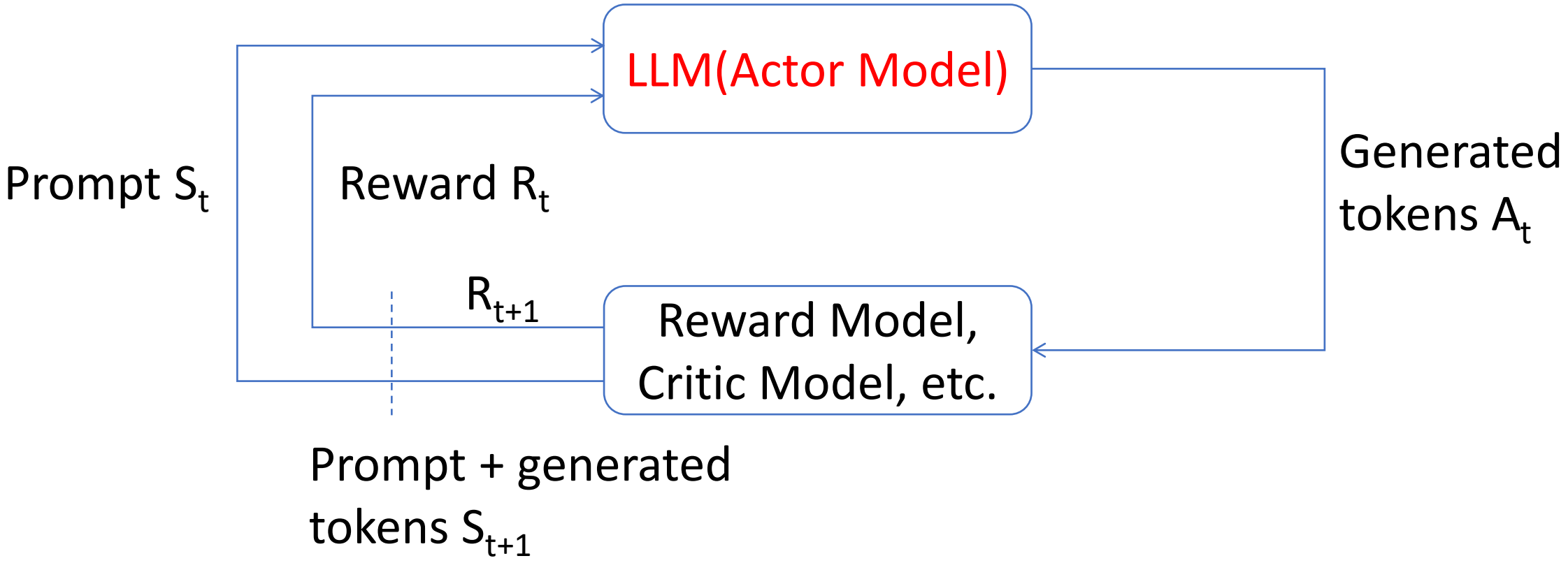
Background

- Reinforcement learning



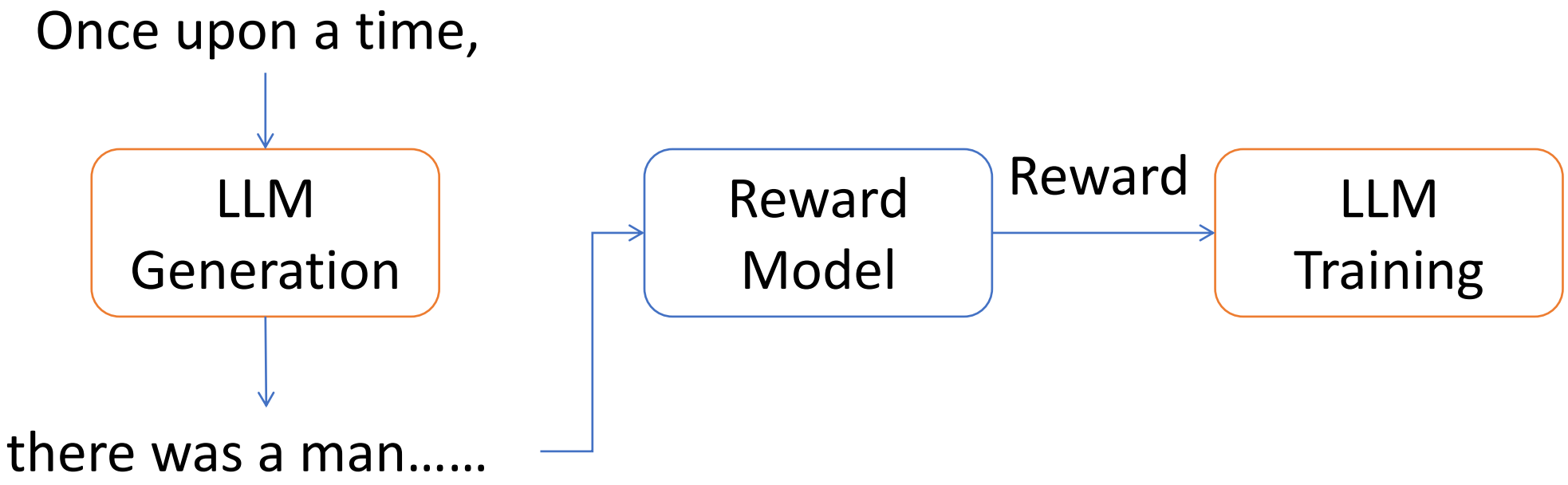
Background

- Reinforcement learning for LLMs



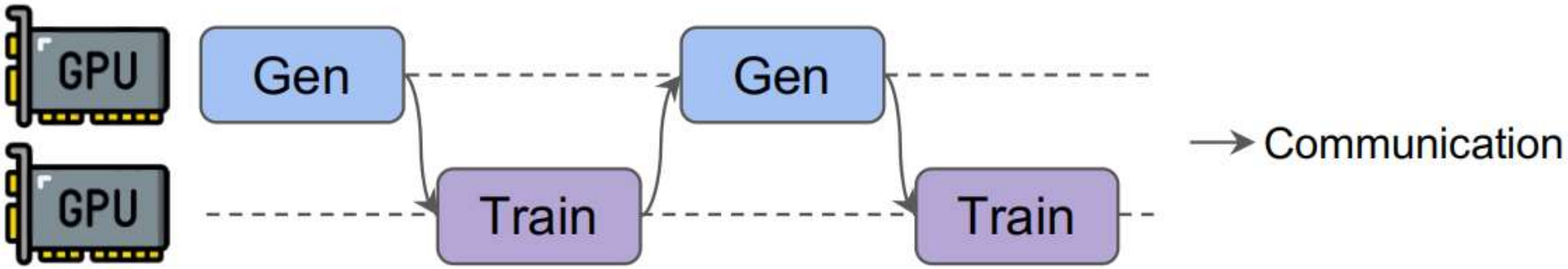
Background

- Two primary stages in RL: generation and training



Background

- Two representative RL framework architectures
  - **Disaggregated architecture**, such as OpenRLHF[1]
    - Reuse existing infrastructures ☺
    - Resource idleness ☹

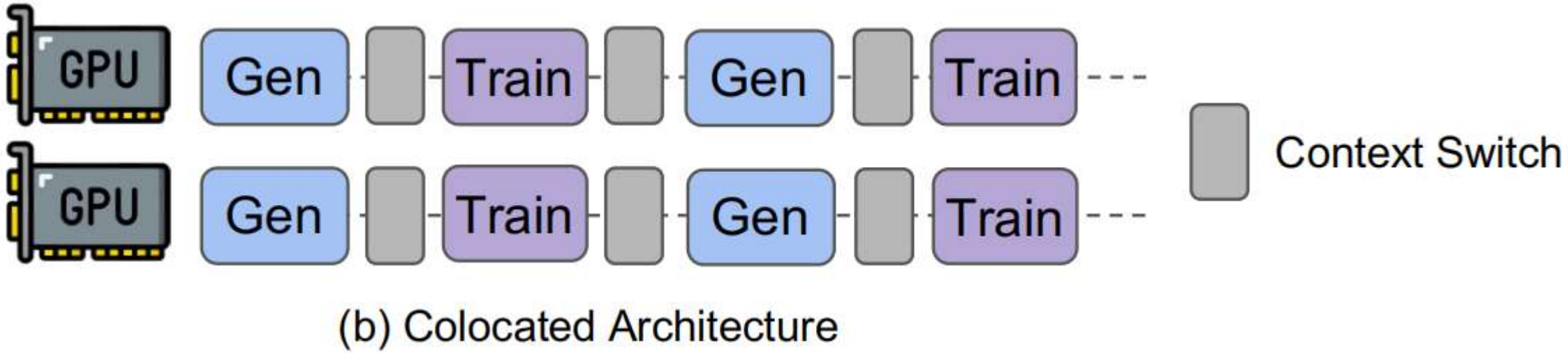


(a) Disaggregated Architecture

[1] Jian Hu, Xibin Wu, Weixun Wang, Dehao Zhang, Yu Cao, et al. 2024. OpenRLHF: An Easy-to-use, Scalable and High-performance RLHF Framework. arXiv preprint arXiv:2405.11143 (2024).

Background

- Two representative RL framework architectures
  - **Colocated architecture**, such as verl[2]
    - Resolve the resource idleness



(b) Colocated Architecture

[2] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. arXiv preprint arXiv:2409.19256 (2024)

Background

- Problems with colocation: **resource coupling**
  - **Generation:** memory bandwidth bound
  - **Training:** compute bound
  - Share the same resource quantities and hardware types
    - But they have divergent computational characteristics

Background

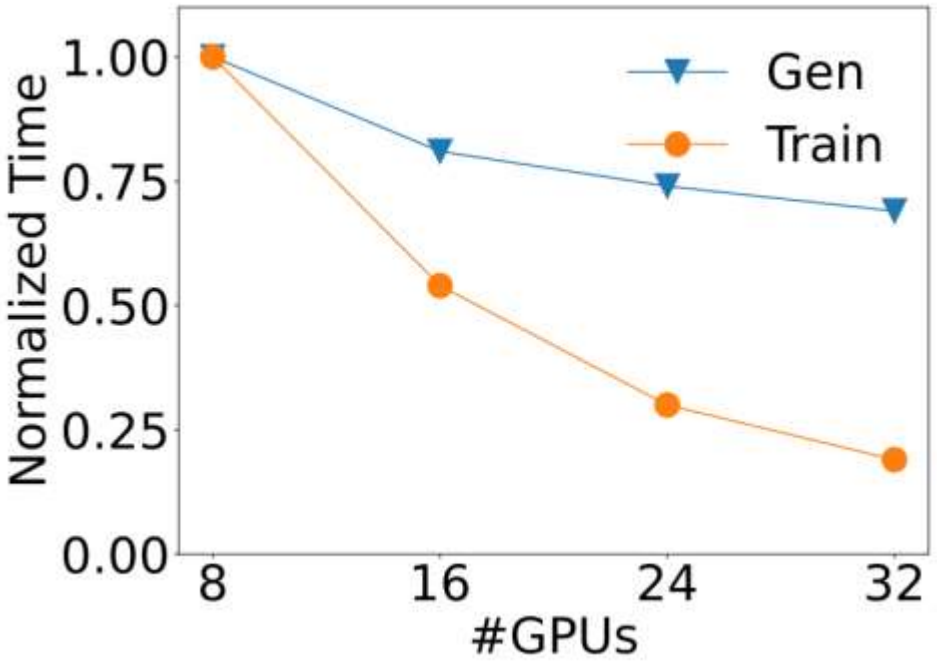
- Problems with colocation: **resource coupling**
  - Prevent selecting the most cost-effectiveness hardware

	H20	H800
BF16 TFLOPS	148	**989.5**
HBM capacity	**96GB**	80GB
HBM bandwidth	**4TB/s**	3.35TB/s
NVLINK bandwidth	**900GB/s**	400GB/s
Cost per machine [60]	1.00	**2.85**

NVIDIA GPU specifications

Background

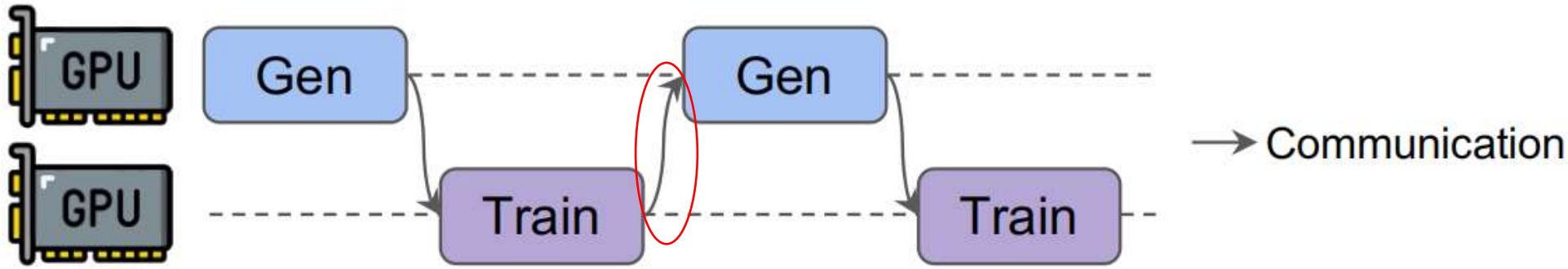
- Problems with colocation: **resource coupling**
  - Profile the execution latency of each stage
    - 7B LLM with 8192 sequence length



generation time quickly reaches a plateau as resources increase

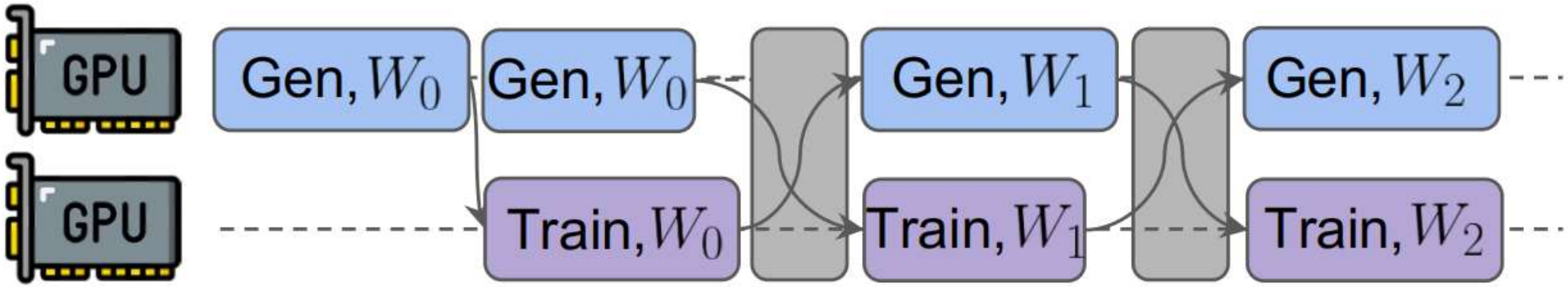
Background

- Synchronous RL vs. asynchronous RL
  - **Synchronous RL**: need to use the latest weights to generate
    - Make it impossible to achieve perfect overlapping



Background

- Synchronous RL vs. asynchronous RL
  - **Asynchronous RL**: allow weights for some extent of staleness
    - Achieve better overlapping
    - Not compromise model performance or convergence[3, 4]



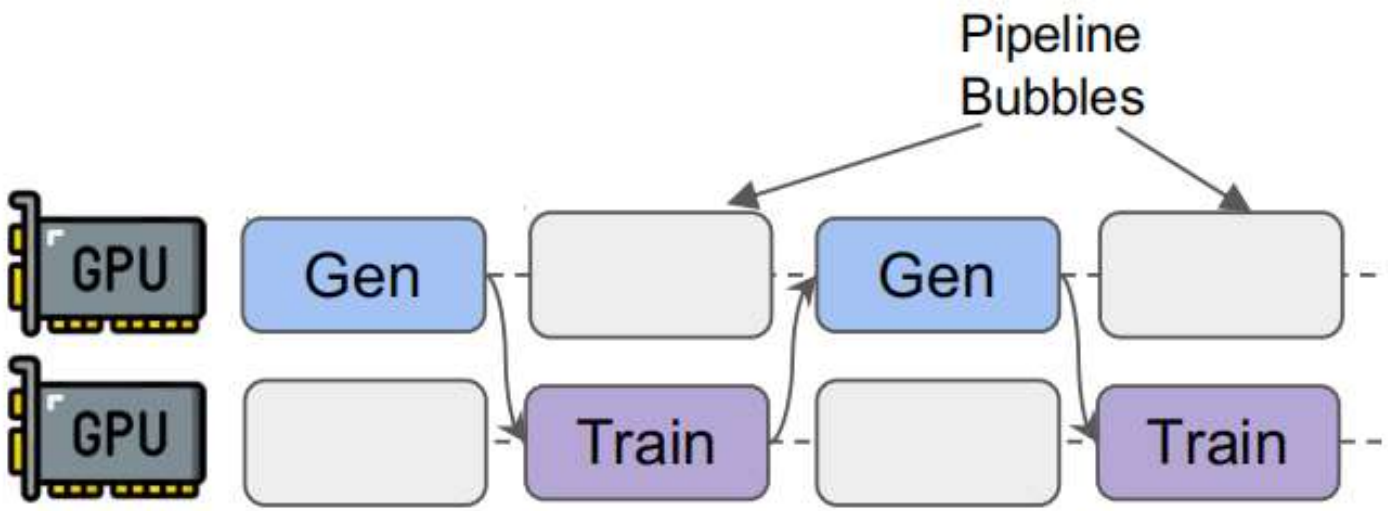
[3] Michael Noukhovitch et.al. 2025. Asynchronous RLHF: Faster and More Efficient Off-Policy RL for Language Models. International Conference on Learning Representations (ICLR)
[4] Taiyi Wang et.al. 2025. DistRL: An Asynchronous Distributed Reinforcement Learning Framework for On-Device Control Agents.

Outline

- Background
- Challenges for Disaggregation
- Design
- Evaluation
- Conclusion

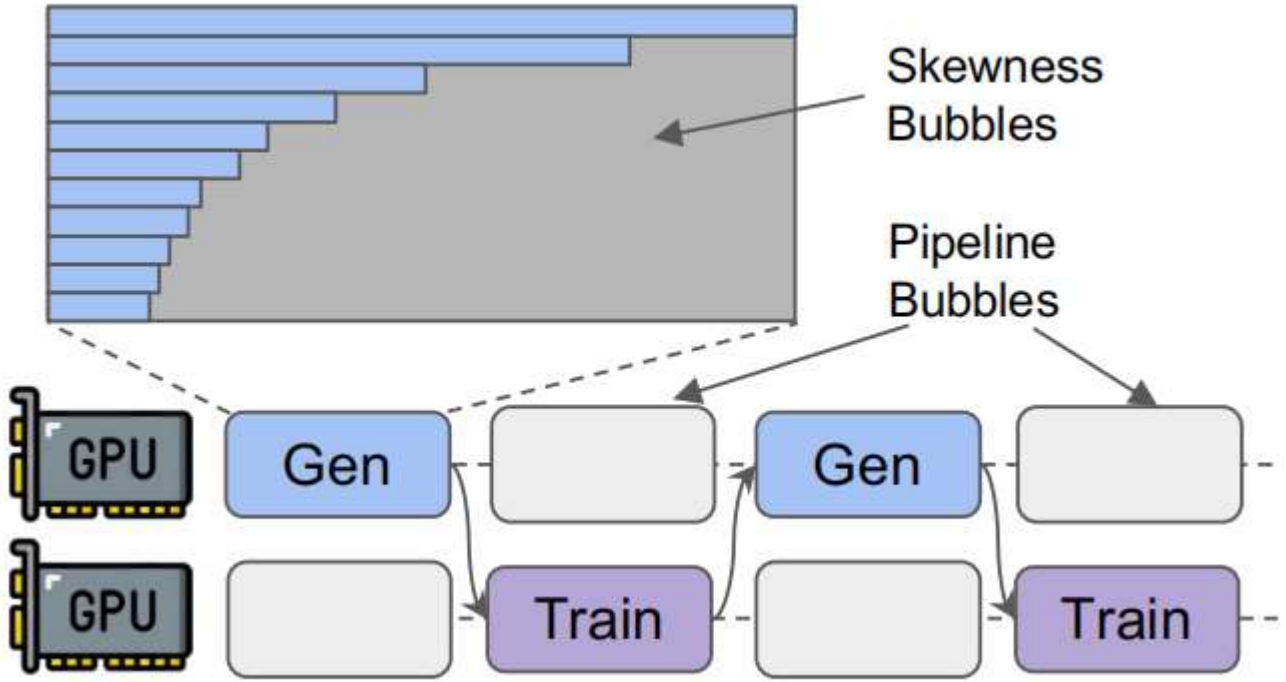
Challenge 1: Pipeline bubbles

- The primary source of inefficiency
  - When one stage is active, the other stage is idle



Challenge 2: Skewness bubbles

- Output length has a skewed distribution in gen stage
- As gen proceeds, only long-tail samples remain
  - undermine GPU utilization

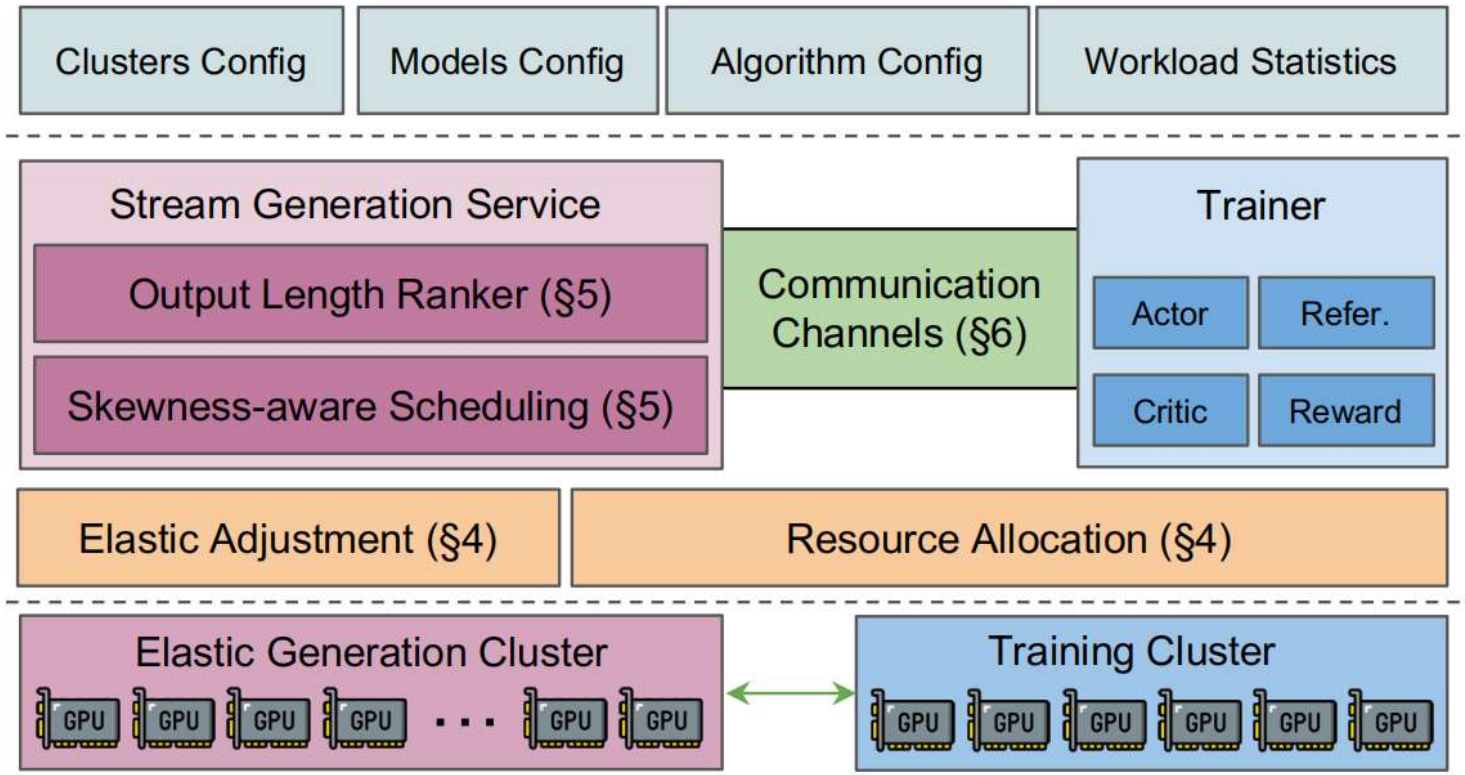


Outline

- Background
- Challenges for Disaggregation
- Design
- Evaluation
- Conclusion

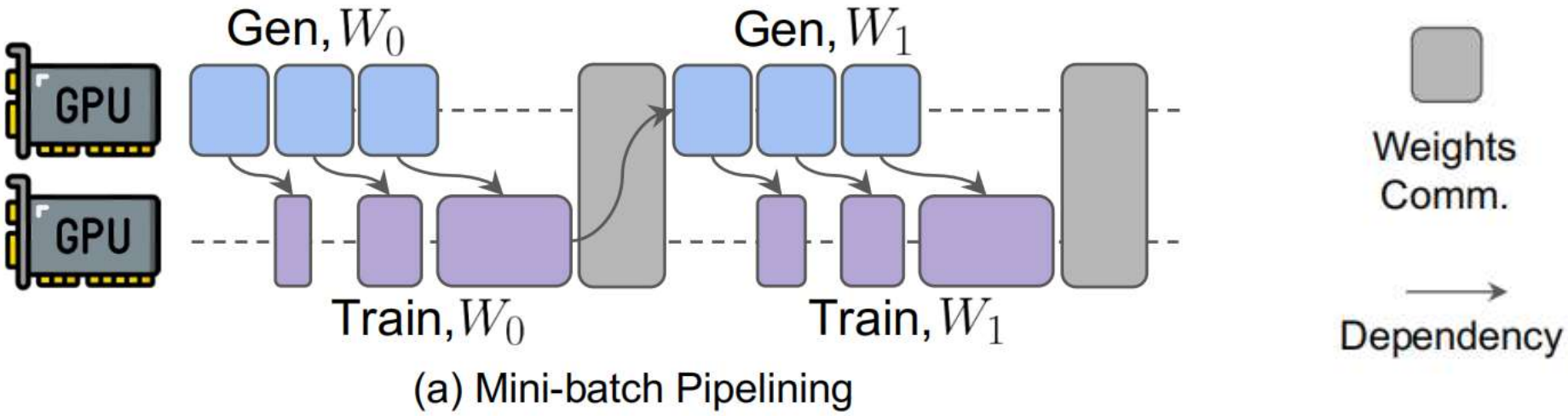
StreamRL Overview

- Generation stage: Stream Generation Service (SGS)
- Training stage: Trainer
- Goal: address pipeline bubbles and skewness bubbles



Tackle Pipeline Bubbles: Overlapping Design

- Strawman solution 1: **Mini-batch pipelining**[5]
  - Evenly divide the samples into several mini-batches
  - Require manually setting the mini-batch size

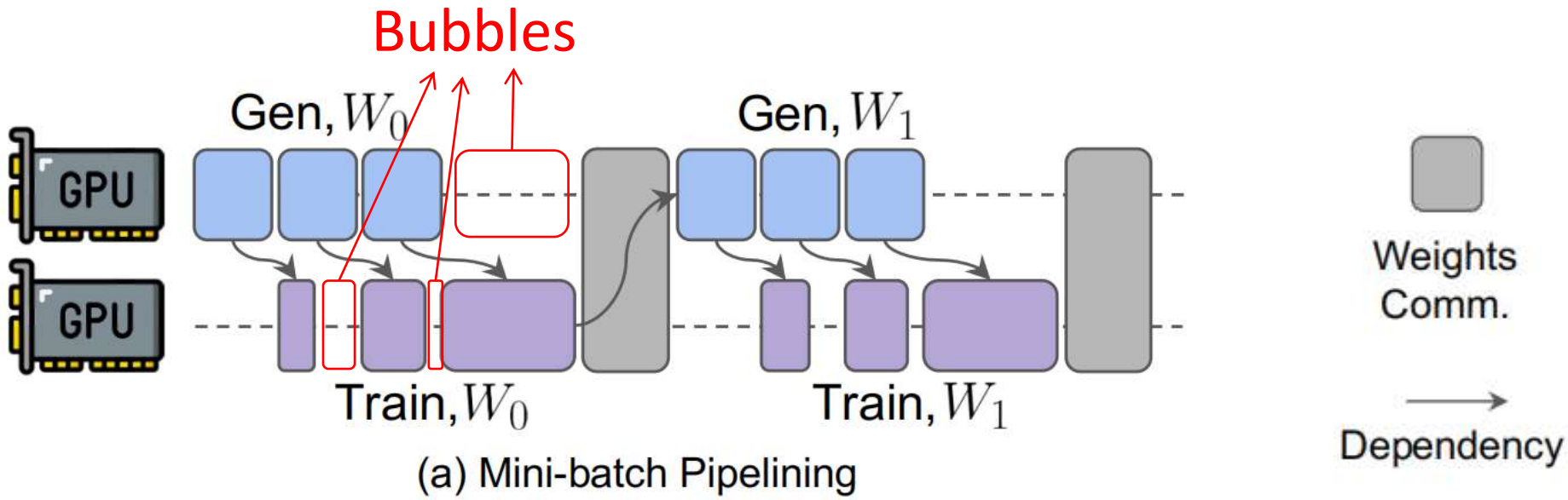


(a) Mini-batch Pipelining

[5] Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. DeepCoder: A Fully Open-Source 14B Coder at O3-mini Level

Tackle Pipeline Bubbles: Overlapping Design

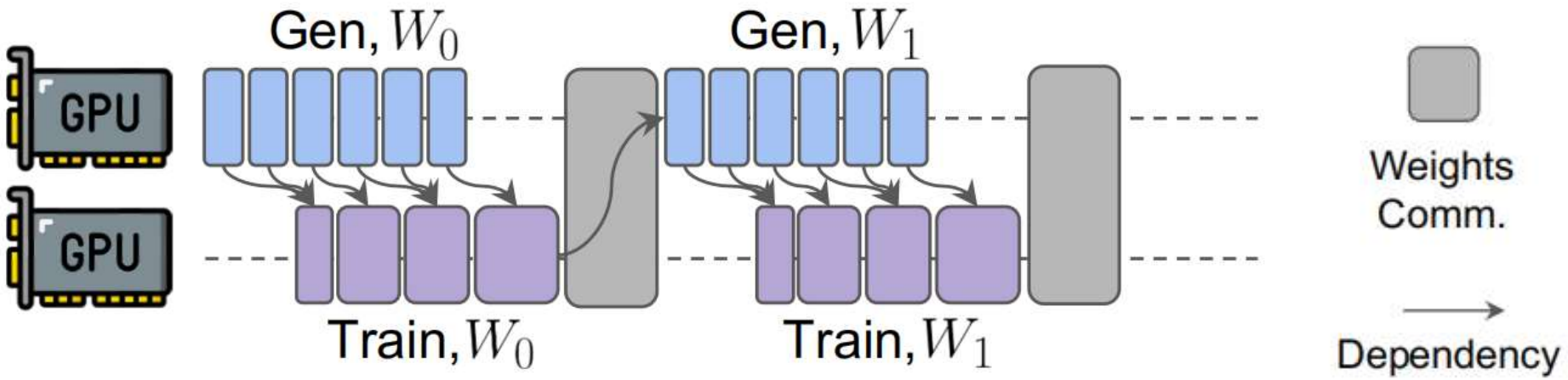
- Strawman solution 1: **Mini-batch pipelining**
  - Problem: The seqlen of the later mini-batches gradually increase
    - The last few mini-batches training often spill over
    - Hard to avoid idle time in the training stage



(a) Mini-batch Pipelining

Tackle Pipeline Bubbles: Overlapping Design

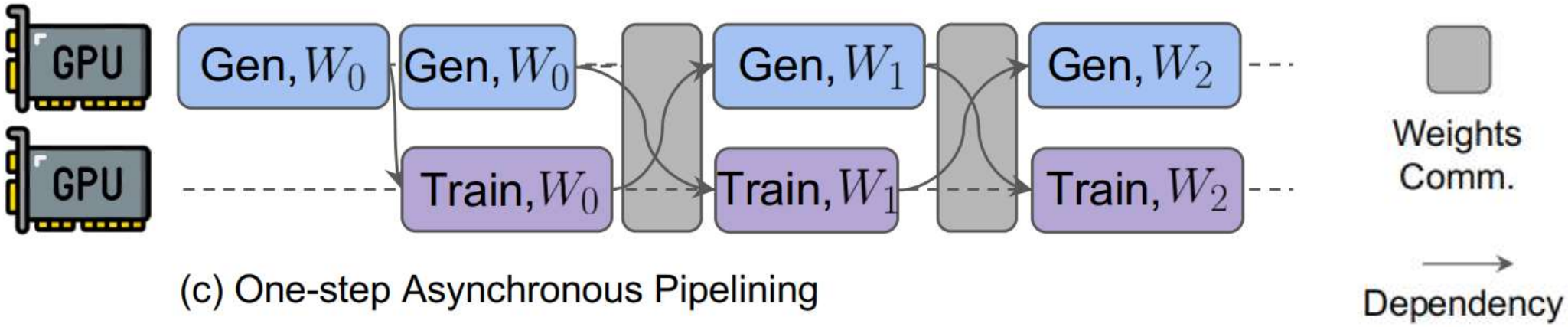
- Our solution: **Dynamic-batch pipelining**
  - Replace the batched generation with stream generation
  - The training stage can start earlier
    - As soon as it receives enough samples to saturate the GPUs



(b) Dynamic-batch Pipelining

Tackle Pipeline Bubbles: Overlapping Design

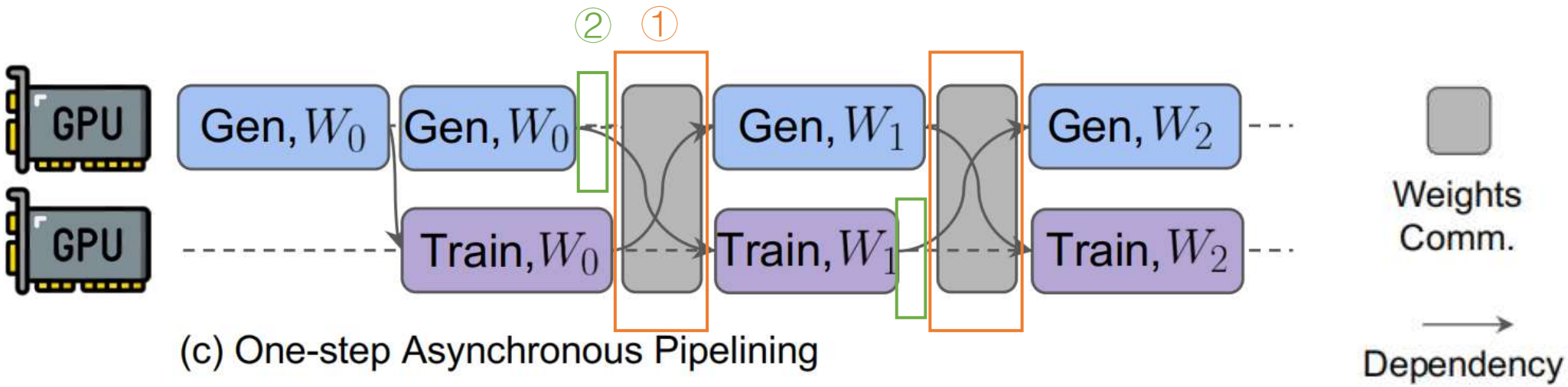
- Strawman solution 2: **One-step asynchronous pipelining**
  - Method: generate one additional batch
    - While the training stage processes samples from the previous iteration



(c) One-step Asynchronous Pipelining

Tackle Pipeline Bubbles: Overlapping Design

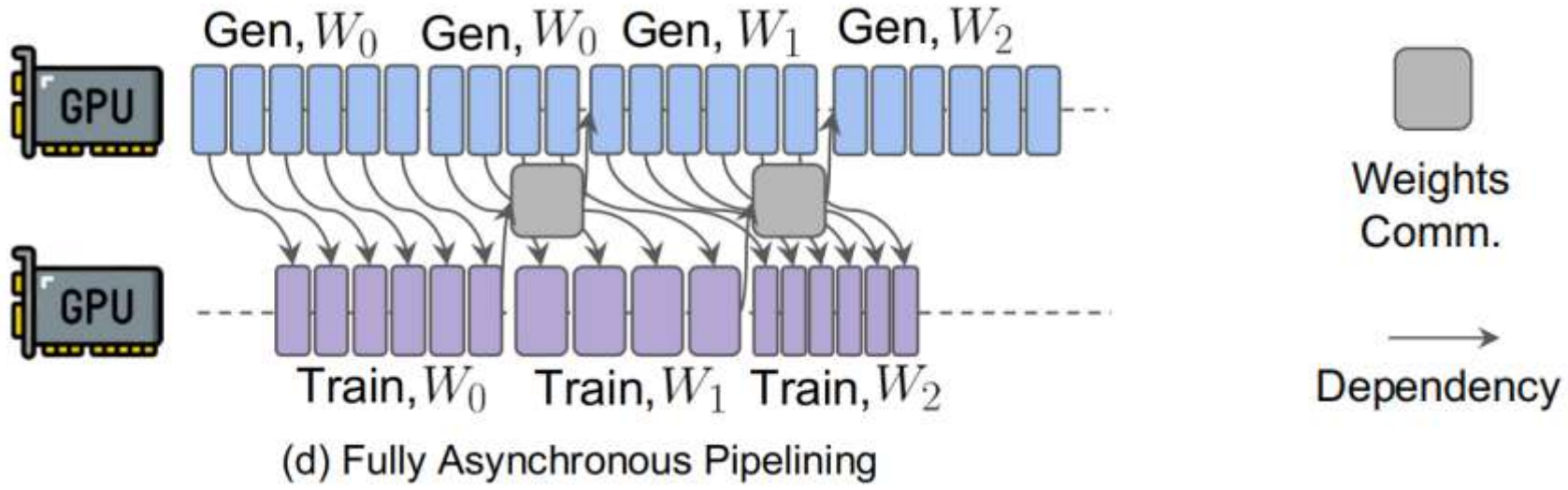
- Strawman solution 2: **One-step asynchronous pipelining**
  - Problems:
    ① Each iter ends with a global synchronization to transmit the weights
    ② Fluctuations in generation and training time across iterations



(c) One-step Asynchronous Pipelining

Tackle Pipeline Bubbles: Overlapping Design

- Our solution: **Fully asynchronous pipelining**
  - Remove weight transmission completely from the critical path
  - No new bubbles will emerge
    - As long as the fluctuation across iterations is limited



(d) Fully Asynchronous Pipelining

Tackle Pipeline Bubbles: Stage Balancing

- Need to balance the execution times of the two stages
  - To achieve better overlapping between SGS and Trainer
- Part1: determine the optimal execution time of SGS and Trainer under a given workload and GPU budget
- Part2: determine the resource allocation for SGS and Trainer

Tackle Pipeline Bubbles: Stage Balancing

- Parallel configuration
  - Trainer: profiler-based approach
    - Inspired by automated parallelism[6]
  - SGS: skewness-aware scheduling

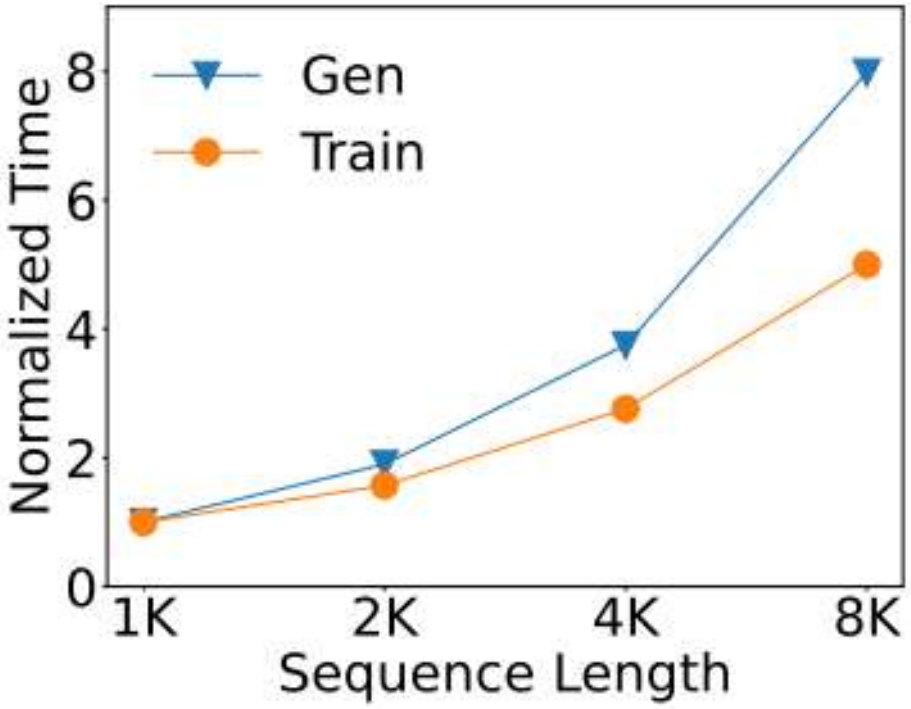
[6]: Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In USENIX OSDI.

Tackle Pipeline Bubbles: Stage Balancing

- Resource allocation for SGS (x) and Trainer (y)
  - Single-datacenter deployment: $x + y \leq n$
    - n is the total GPU budget
  - Cross-datacenter deployment: $x \leq m, y \leq n$
    - m and n denote the GPUs in SGS and Trainer

Tackle Pipeline Bubbles: Dynamic adjustment

- Another problem: the generation length of LLMs increases progressively during RL training
  - Generation time grows more significantly than training
  - Cause imbalance between two stages

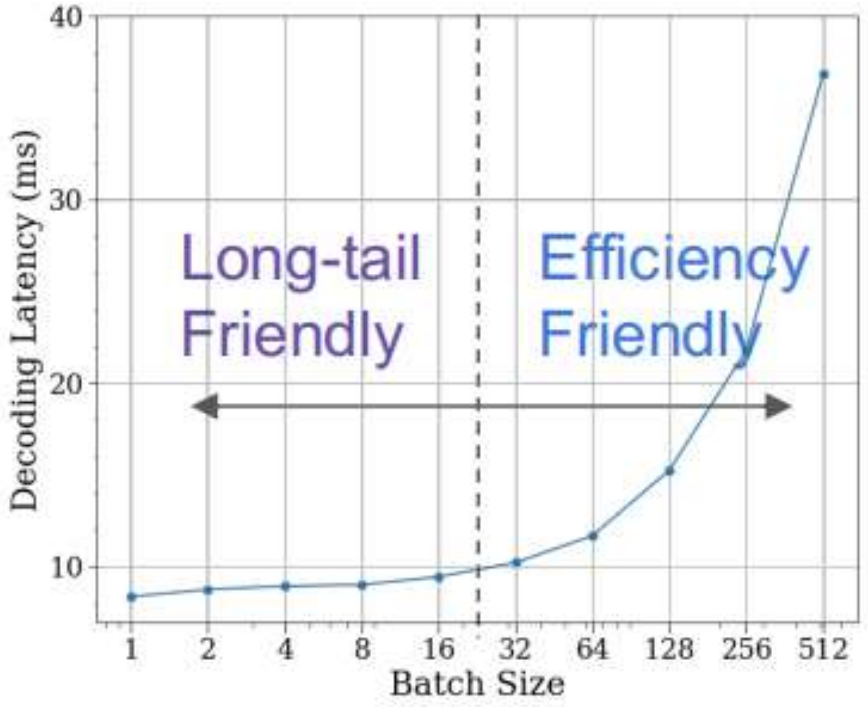


Tackle Pipeline Bubbles: Dynamic adjustment

- Another problem: the generation length of LLMs increases progressively during RL training
- Solution:
  - Monitor the execution time gap(δ) between gen and train
  - Estimate the reduction in generation time(δ')
    - By adding one DP unit to SGS
  - When $\delta \geq \delta'$, then add one more DP unit to SGS

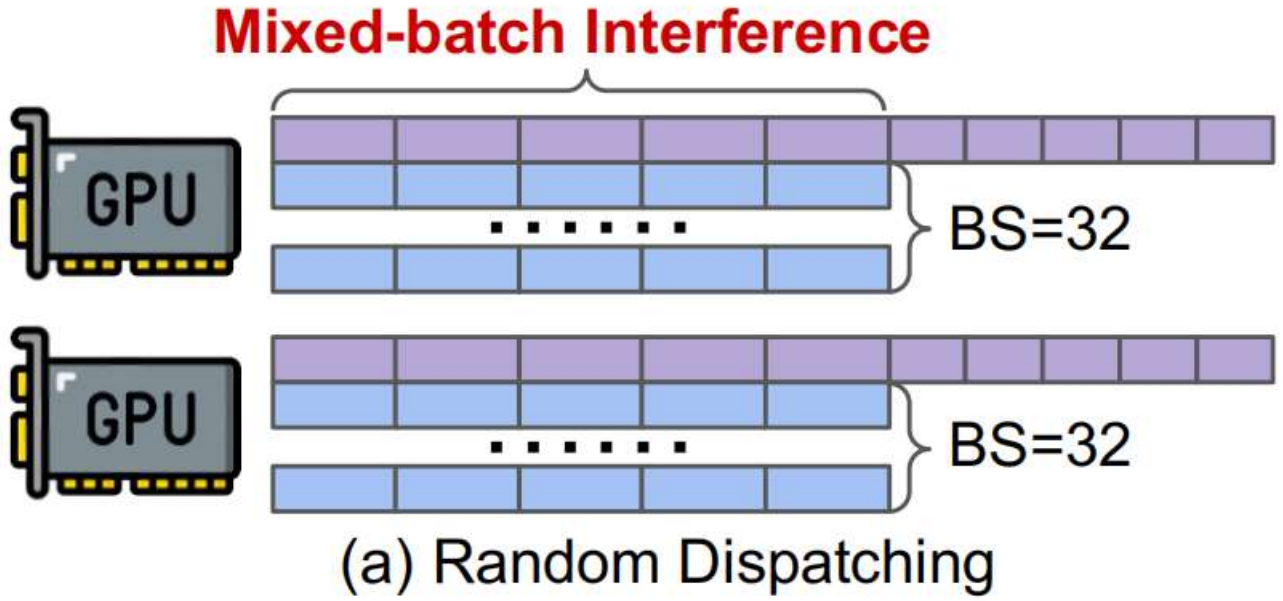
Tackle Skewness Bubbles

- The trend of per-token decoding latency for a 7B model on an NVIDIA A100 GPU as the batch size increases
  - Latency grows slowly before reaching compute-bound
  - Then increases almost linearly after that



Tackle Skewness Bubbles

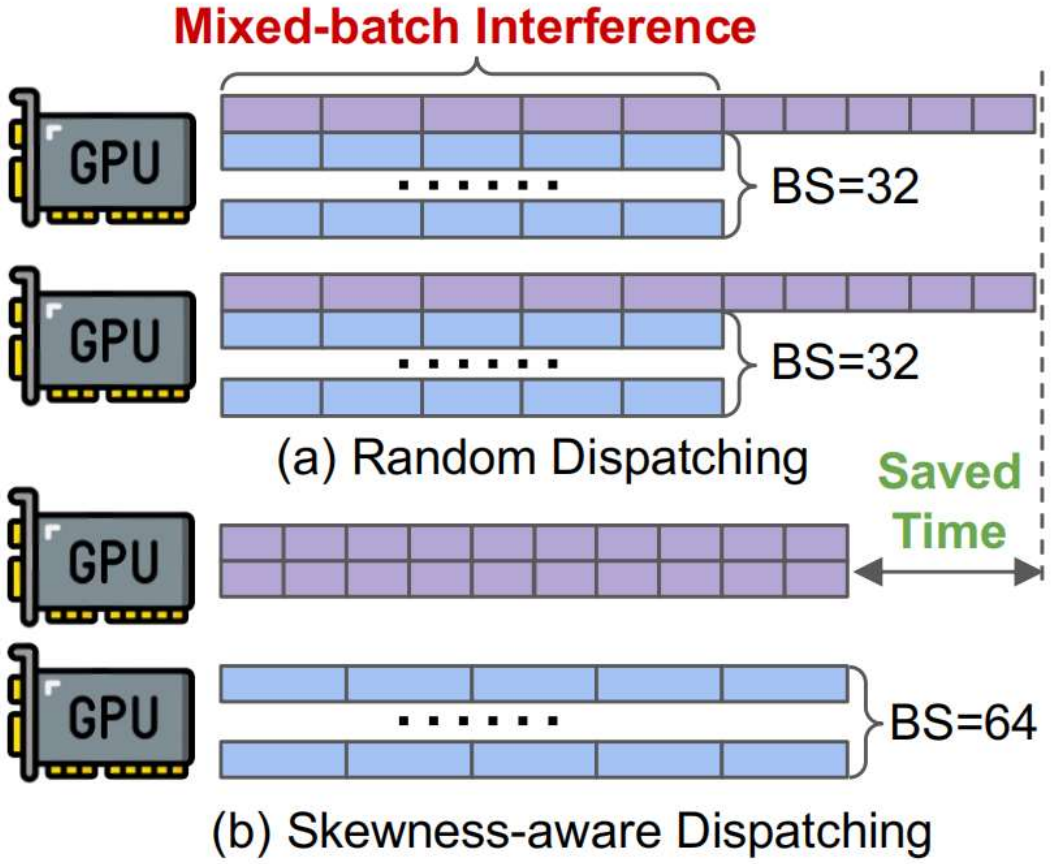
- Existing systems: prompts are assigned randomly
  - Example: DP size 2, 64 samples with 2 long-tail samples
  - Significant interference for the long-tail samples

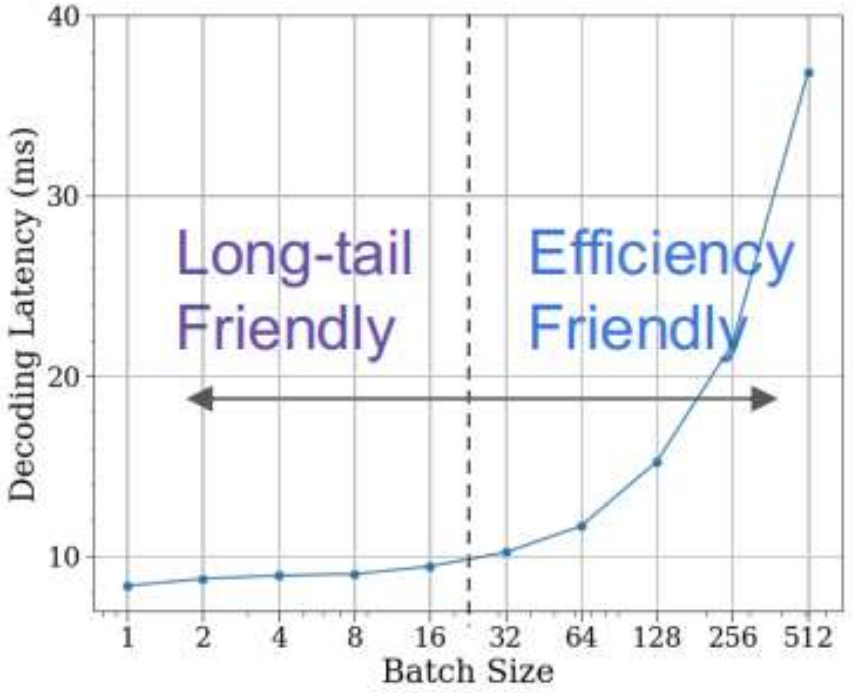


(a) Random Dispatching

Tackle Skewness Bubbles

- Solution: extract the long-tail samples
  - Long-tail samples: smaller batch size
  - Regular samples: large batches





Tackle Skewness Bubbles

- How to predict the output length of LLM generation?

Tackle Skewness Bubbles

- How to predict the output length of LLM generation?
  - Estimate the relative ranks of output lengths with another model
  - Classifies prompts based on difficulty
    - a property inherent to the prompts themselves

Tackle Skewness Bubbles

- Skewness-aware Scheduling Algorithm
  - Sort the prompts by their estimated output lengths
  - Mark the longest α% of them as long-tail samples
    - α = 20 yields good results
  - Estimate average output length for regular and long-tail samples
    - P50 and P90 of output length distribution
  - Iterate all configurations and minimize the generation time

Outline

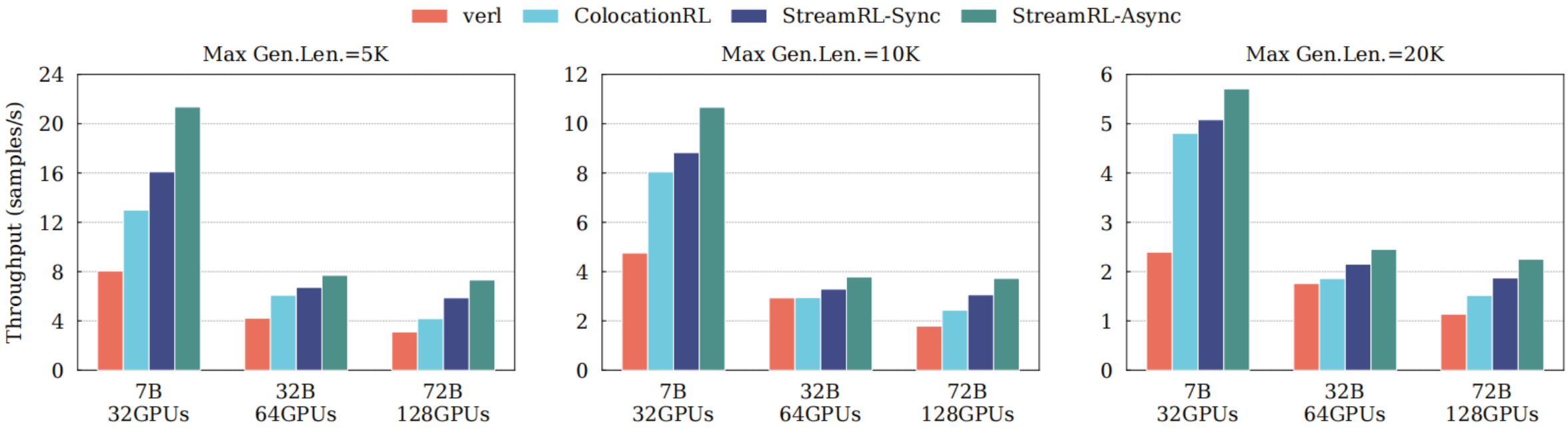
- Background
- Challenges for Disaggregation
- Design
- Evaluation
- Conclusion

Evaluation

- Experimental Setup
  - Testbed:
    - 16 nodes * 8 NVIDIA H800-80GB GPUs, 8 * 200 Gbps RDMA network
    - 4 nodes * 8 NVIDIA H20-96GB GPUs, 100Gbps TCP network
      - For cross-datacenter experiments
    - 80Gbps dedicated link between two clusters
  - Models: Qwen2.5 models(7B, 32B, 72B)
  - Dataset: internal CodeMath prompts dataset
  - Settings: PPO algorithm
- Baselines: verl, ColocationRL

Evaluation

- End-to-end throughput (samples/s) under different sequence length and model size settings



Compared to verl, StreamRL-Sync achieves a 1.12×–2.12× speedup, StreamRL-Async achieves 1.30×–2.66× throughput improvement.

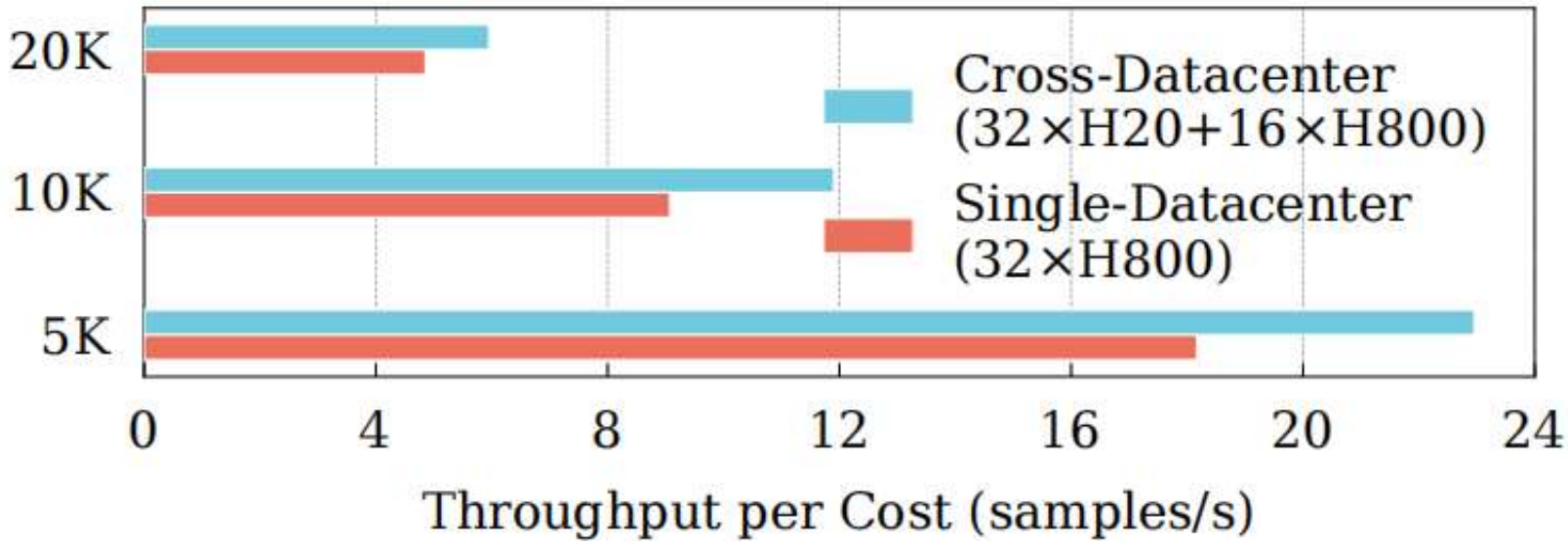
Evaluation

- Ablation Studies: throughput improvement breakdown when training 72B model on the 20K maximum length
- Baseline: ColocationRL

Idx	Method	Normalized Throughput
1	Colocation Baseline	1.00
2	(1) with skewness-aware scheduling	1.08 (+8%)
3	(2) with disaggregation + streaming	1.23 (+15%)
4	(3) with asynchronous	1.48 (+25%)

Evaluation

- Cross-Datacenter and Heterogeneity: end-to-end experiment with the 7B model
  - Cross-Datacenter: SGS 32*H20, Trainer 16*H800



The throughput normalized by the hardware cost between cross- and single-datacenter deployment. StreamRL achieves a 1.23×–1.31× higher throughput normalized by hardware cost.

Outline

- Background
- Challenges for Disaggregation
- Design
- Evaluation
- Conclusion

Conclusion

- We revisit the disaggregated architecture for RL training
- We present StreamRL to address the pipeline bubbles and skewness bubbles
- StreamRL achieves up to a 2.66x speedup compared to the current state-of-the-art RL framework

Thanks!