UNIVERSAL AND EFFICIENT LOADING BALANCING FOR RL TRAINING OF LARGE MULTIMODAL MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) is crucial for aligning Vision-Language Models (VLMs), but its practical application is hampered by significant system-level bottlenecks. The typical RL pipeline, encompassing data loading, inference-based rollouts, and model updates, suffers from severe inefficiencies when applied to VLMs due to the extreme heterogeneity of multimodal data. Centralized data loading creates I/O bottlenecks with large media files, while variations in sequence length across text, image, and video inputs lead to critical load imbalance during computation, leaving expensive GPU resources underutilized. Existing systems either focus on text-only RL or employ general load-balancing techniques that are incompatible with the small-batch, iterative nature of RL training. To address these challenges, we present FlexRL, a holistic system designed to optimize the end-to-end VLM RL pipeline. FlexRL introduces two core contributions: (1) a **Decentralized Data Pipeline** that parallelizes data fetching and preprocessing across worker nodes, facilitates metadata-only scheduling on the single controller, eliminating the central bottleneck and accelerating data-intensive stages; and (2) a novel **Hybrid Sequence Sharding** mechanism that partitions sequences into fine-grained chunks. This enables sub-sequence level load balancing for both inference and training, effectively mitigating workload skew. Our evaluation on a 128-GPU cluster shows that FlexRL significantly improves training efficiency by up to 4.2$\times$ in long video training scenarios compared to state-of-the-art baselines, enabling more efficient and scalable RL for large multimodal models.

1 INTRODUCTION

Reinforcement learning (RL) has proven to be a powerful paradigm for aligning Vision Language Models (VLMs) with human preferences and enhancing their instruction-following capabilities (Team et al., 2025a;b;d; Wang et al., 2025b). A typical RL workflow for VLMs involves several distinct stages (1) loading diverse multimodal data and prompts, (2) generating responses via policy model inference (i.e., rollouts), and (3) updating the model parameters based on rewards. While conceptually straightforward, scaling this process for VLMs exposes severe system-level bottlenecks across the entire pipeline, hindering training efficiency and scalability.

The challenges are multifaceted and manifest differently in each stage. Firstly, the **data loading** stage becomes a significant bottleneck. VLMs are trained on heterogeneous datasets containing text, high-resolution images, and long video clips. In many RL frameworks (e.g., VeRL (Sheng et al., 2025)), data loading and preprocessing are centralized, causing the master node to become a chokepoint, limited by its memory and compute capacity, especially when handling large media files. Secondly, both the **inference stage** and the **model update stage** suffer from a critical load imbalance problem. In these phases, a single batch can contain a mix of short image-text queries, long text-only reasoning tasks, and video inputs with tens of thousands of tokens. This extreme variation in sequence length and modality leads to a highly skewed distribution of computational and memory loads across GPUs. Consequently, some devices are overwhelmed while others remain underutilized, creating a bottleneck that stalls the entire distributed system.

Existing systems fail to provide a holistic solution for the VLM RL pipeline. On one hand, traditional RL frameworks (Sheng et al., 2025; Hu et al., 2024) are optimized for text-only models and lack sophisticated mechanisms to handle the load imbalance inherent in multimodal data. These

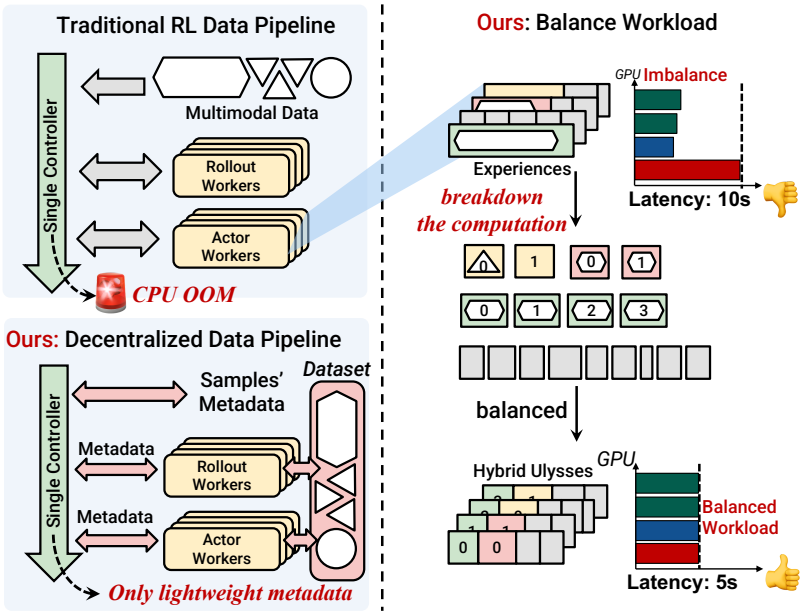

Figure 1: **FlexRL Overview**. We implement a decentralized data pipeline and hybrid sharding method to achieve workload balance for VLM RL Training.

frameworks typically employ naive sequence bucketing and packing strategy, which has limitations due to sequence-level granulation. On the other hand, general-purpose large model training systems (Wang et al., 2025c;d; Li et al., 2024; Ge et al., 2024; 2025) propose heterogeneous parallelism methods over DP instances and gradient steps, which either require large batch sizes and gradient accumulation to be effective. This assumption breaks down in the context of RL, which often utilizes small batch sizes to maximize the utility of dynamically generated samples and maintain training stability. These methods are thus inefficient for the dynamic and iterative nature of the RL rollout-update loop.

To this end, we introduce FlexRL, a system designed to provide a comprehensive, end-to-end optimization for the VLM RL pipeline. FlexRL deconstructs the performance bottlenecks in each stage of the RL process and introduces targeted solutions: **For the Data Loading Bottleneck:** We design a *Decentralized Data Pipeline* that parallelizes the expensive data fetching and preprocessing tasks across all worker nodes and only operates lightweight metadata of samples on the single controller. This decentralizes the workload, eliminating the master node bottleneck and significantly accelerating data throughput for large media files. **For Inference and Update Load Imbalance:** We propose a novel *Hybrid Sequence Sharding* mechanism. Instead of treating sequences as indivisible units, we partition them into fine-grained chunks. This allows FlexRL to balance the load at a sub-sequence level, effectively mitigating the imbalance caused by extreme length variations during both the inference and training steps. This is complemented by a specialized balancing strategy for the vision tower. FlexRL incorporates an efficient decision algorithm to determine the optimal sharding strategy for each sequence and a dynamic execution engine to orchestrate the complex computation and communication patterns, maximizing hardware utilization.

We implement FlexRL on top of the veRL framework and conduct extensive experiments on a cluster of 128 H800 GPUs. Our evaluation demonstrates that by optimizing the entire RL workflow, FlexRL achieves significant end-to-end performance gains. For instance, when training on a diverse mix of multimodal data, FlexRL improves training throughput by up to 4.2× and forward computation by up to 3× compared to existing baseline systems.

In summary, our contributions are:

- We provide a systematic analysis of the performance bottlenecks across the entire VLM RL pipeline, from data loading to inference and model updates.

- We propose FlexRL, a *Hybrid Sequence Sharding* mechanism for more general and flexible load balancing to address these stage-specific challenges and a holistic system that integrates a lightweight metadata-based dataloader to eliminate the storage bottleneck.
- We design an efficient algorithm and a dynamic execution engine to solve the complex scheduling problem introduced by our fine-grained, hybrid sharding approach.
- Our evaluation shows that FlexRLsignificantly improves training efficiency in VLM training and achieves workload balancing in various scenarios compared to traditional methods.

2 Related Work

2.1 RL Training Frameworks

Reinforcement learning has become a central paradigm for aligning and enhancing large language models (LLMs). Recent frameworks such as VeRL (Sheng et al., 2025), siiRL (Wang et al., 2025e), AReal (Fu et al., 2025), StreamRL (Zhong et al., 2025), MiroRL (Team & Team, 2025), ROLL (Wang et al., 2025a), and OpenRLHF (Hu et al., 2024) provide system-level support for distributed RL training. These frameworks focus on issues such as asynchronous rollout-update decoupling, scalable data pipelines, and integration with model-parallel training backends. While these systems improve throughput and modularity, they are largely developed for text-only LLMs and short-context RLHF settings. Their load balancing strategies often assume relatively homogeneous workloads, leaving open challenges in multi-modal and long-context reinforcement learning.

2.2 VLM Training Frameworks

The emergence of multi-modal LLMs has motivated training frameworks such as DistTrain (Zhang et al., 2025), DistMM (Huang et al., 2024), LongVILA (Chen et al., 2025), and VeOmni (Ma et al., 2025). These systems propose optimizations for heterogeneous architectures combining vision towers and language backbones. For instance, they disaggregate model components, employ hybrid parallelism, or develop scheduling algorithms to reduce communication overheads. Despite these advances, existing VLM frameworks primarily target pretraining or supervised fine-tuning. They do not explicitly address the unique challenges of RL training, such as small batch sizes, dynamically generated trajectories, and highly variable sequence lengths across modalities.

2.3 Loading Balancing for Large Model Training

A line of work focuses on load balancing techniques for efficient large model training. Classic methods rely on sequence bucketing and packing, which sort samples by length and allocate them across GPUs to minimize padding (Team et al., 2025b;c;d; Wang et al., 2025b). More recent approaches introduce heterogeneous parallelism and dynamic reconfiguration, as in FlexSP (Wang et al., 2025c), HotSPa (Ge et al., 2024), Hydraulis (Li et al., 2024), ByteScale (Ge et al., 2025), and WLB-LLM (Wang et al., 2025d), which adjust parallelism configuration across DP(Data Parallel) ranks depending on sequence characteristics. Although effective for load balance of large-scale pretraining, these approaches typically operate at coarse sequence-level granularity, employing different parallelism configurations across sequence buckets or gradient steps. These methods rely on a large batch size that requires gradient accumulation, providing opportunities for load balancing.

3 RL Training of VLMs

Existing loading balancing works mainly focus on LLM pretraining. They usually involve sequence reordering and grouping (also known as bucketing). However, these methods have limitations when applied to RL training, especially for VLMs.

3.1 Challenges of VLM RL Training

The RL training of VLMs presents unique challenges due to the extreme variations in sequence lengths and computational requirements arising from different data modalities, such as text, images,

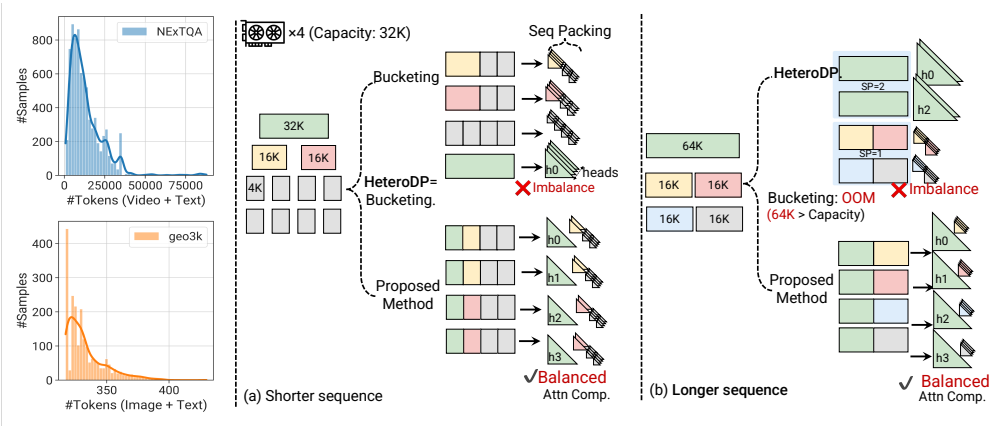

Figure 2: Motivation of Hybrid Sharding. **Left**: Distribution of token counts in typical video-text (NExTQA (Xiao et al., 2021)) and image-text (Geo3K(Lu et al., 2021)) datasets, showing extreme variation in sequence lengths. **Middle**: An imbalance with shorter sequences, where conventional bucketing and Heterogeneous DP (e.g. FlexSP (Wang et al., 2025c)) lead to load imbalance. **Right**. For longer sequences, bucketing can cause out-of-memory (OOM) errors and further imbalance, while our approach enables fine-grained sharding and balanced attention computation across GPUs.

and videos, and model heterogeneity. For example, in a single training batch, we may have image-text sequences of a few hundred tokens, long reasoning sequences with thousands of tokens, and long video sequences of tens of thousands of tokens. Firstly, due to the variations in the number of visual tokens, the vision tower can have significantly different computational requirements across different samples. In Figure 2, we show the distribution of two typical video-text and image-text datasets. We observe that video data contains far more tokens than typical multi-image samples and varies significantly. Secondly, the backbone LLM also faces extreme variations in sequence lengths, leading to a significant imbalance in both memory and compute workloads across different samples. Besides, unlike text data, multimodal data contains pixel data of videos and images, which requires a huge amount of space. Existing RL frameworks like veRL use a central controller to preprocess, store, and transfer multimodal data, leading to both high latency and pressure on CPU memory.

Sequence Bucketing and Packing. Some works (Team et al., 2025b;c; Wang et al., 2025b; Team et al., 2025d) employ a bucketing algorithm that iteratively traverses the sequences in descending order of their lengths and assigns each sequence to the bucket with the least computation load. In each GPU, the sequences in the same bucket are packed together and padded to the GPU's token capacity. The parallelism configuration is fixed for all buckets. This method is simple and efficient, as it only requires a single pass through the sequences and does not involve modifying either the parallelism configuration or the training algorithm. However, in this method, on single longest sequence can easily lead to the worst case, as shown in Figure 2(middle). In this example, we have 11 sequences (1×32K, 2×16K, and 8×8K) and four GPUs. No sequence surpasses the GPU capacity, so we can simply use data parallelism (e.g., FSDP (FSD, 2023)) with a bucketing algorithm to balance the workload. No matter how we bucket the sequences, one GPU will always get the longest sequence (32K), leading to $4\times$ slowdown. This problem can be extended to more general 3D/4D parallelism, as they employ a homogeneous parallelism configuration for all DP ranks.

3.2 Understanding the Issue of Existing Loading Balancing Methods

Existing methods fall short in the RL regime for three reasons: (a) Small-batch RL leaves little split-batch freedom. With only a handful of sequences per step, sequence-level bucketing/packing (sort-by-length, pack-to-capacity with fixed parallelism) cannot hide outliers; the step time is dominated by the longest sequence (e.g., 1×32K, 2×16K, 8×8K on 4 GPUs inevitably yields a 32K straggler and $\sim4\times$ slowdown; Fig. 2(Middle)). Moreover, attention compute scales as $O(L^2)$ while activation memory is $O(L)$, so no single per-sequence placement can simultaneously equalize compute and memory across GPUs, leading to both padding waste and stragglers. (b) Heterogeneous DP across buckets (e.g., FlexSP (Wang et al., 2025c), HotSPa (Ge et al., 2024), Hydraulis (Li et al.,

2024), ByteScale (Ge et al., 2025), relies on gradient accumulation and step-wise reconfiguration. In RL, frequent rollout–update alternation and small batches make re-sharding, extra synchronization, and optimizer-state movement non-trivial overheads. When sequence/context parallelism is enabled, many short sequences pay redundant all-to-all communication; choosing a single SP/CP degree per bucket mismatches mixed-length samples and reintroduces imbalance. (c) Multimodality introduces heterogeneity. Vision-tower cost scales roughly with the number of images/frames, not text tokens; bucketing by text length alone ignores visual workload, so mixing videos and images within a bucket yields highly skewed per-GPU compute/memory even if token counts look balanced. These limitations motivate a granularity finer for better and more universal load balancing (see Sec. 4).

4 FLEXRL

FlexRL is designed to accelerate the end-to-end reinforcement learning pipeline for Large Multimodal Models (VLMs). A typical VLM RL training loop, such as PPO, consists of three main phases: (1) **Inference Phase**, where the actor/reward model performs forward computation on the generated trajectories; (2) **Data Preparation Phase**, where experiences are sampled and preprocessed for training, often bottlenecked by I/O and CPU processing for large media data; and (3) **Update Phase**, where the policy and value models are trained on the collected trajectories. that these sequences should fill all GPUs memory for the purpose of maximizing utilization.

4.1 PRELIMINARIES: SEQUENCE PARTITIONING FOR LOAD BALANCING

To address the load imbalance caused by highly skewed sequence lengths, a foundational strategy is to partition original sequences into smaller, more manageable units. By creating smaller, equal-sized chunks, we establish uniform computational and memory footprints, which provides a crucial opportunity for effective load balancing.

A prominent technique that implements this principle is Ulysses Sequence Parallelism (Jacobs et al., 2023). While Ulysses was originally proposed to enable the training of exceptionally long sequences that would otherwise exceed single-GPU memory, we observe that its underlying partitioning mechanism can be repurposed as a powerful tool for load balancing. This shifts the perspective on sequence parallelism: instead of scaling sequence length, it becomes a flexible strategy for balancing workloads of varied-length sequences, such as the one depicted in Figure 2(Middle), which actually doesn't necessarily require sequence parallelism.

In the Ulysses approach, a sequence $\mathbf{x}$ of length L is split along the sequence dimension across N devices. Each device $i \in \{0, \ldots, N-1\}$ receives an equal-sized chunk $\mathbf{x}_i$ of length L/N. During the forward pass of a layer, each device computes its local Query ($\mathbf{Q}_i$), Key ($\mathbf{K}_i$), and Value ($\mathbf{V}_i$) tensors from its chunk $\mathbf{x}_i$. To compute the full attention scores, the $\mathbf{K}_i$ and $\mathbf{V}_i$ tensors must be shared among all N devices. This is achieved via an `all-to-all` communication operation. After the `all-to-all`, each device possesses the complete Key and Value tensors for a subset of attention heads, allowing it to compute its shard of the attention output. Another `all-to-all` operation is then performed to gather the output, which is then passed to the subsequent layers.

4.2 HYBRID SEQUENCE SHARDING FOR WORKLOAD BALANCING

Strawman Solution: Greedy Sharding. When moving from a single sequence to a batch, it is evident that we can achieve perfect load balancing by sharding every sequence to all GPUs. While theoretically leading to an equal distribution of computation and memory, this strategy is impractical as it presents two significant drawbacks. First, Ulysses' scalability is capped by the number of attention heads, limiting the degree of parallelism. Second, it incurs substantial communication overhead. While the computational complexity for a batch of packed sequences is approximately $O(\sum L_i^2)$, the communication volume is proportional to the total sequence length, leading to an increased communication-to-computation ratio and a higher GPU idle ratio.

Our Solution: Hybrid Sharding. As shown in Figure 2(Middle), a more effective strategy is to adopt a hybrid strategy by assigning a tailored sharding degree to each sequence. Firstly, this approach can resolve the redundant communication issue while maintaining a near-optimal load bal-

Algorithm 1 Training Step with FlexRL.

Input: Global batch of sequences $S_{\text{global}} = \{s_1, s_2, \ldots, s_B\}$
Phase 1: Decision (on the single controller)
Determine an assignment map M by solving the per-sequence parallelism optimization problem.
for each sequence $s_i \in S_{\text{global}}$ **do**
    $(N_i, G_i) \leftarrow M[s_i]$ ▷*N_i is SP size, G_i is the device group for s_i. Utilizing a tailored bucketing algorithm for efficiency.*
Phase 2: Dynamic Execution (on all ranks in parallel)
for rank $k \in \{0, \ldots, \text{world\_size} - 1\}$ **in parallel do do**
    Initialize local packed sequence $x_k^{\text{local}} \leftarrow \emptyset$. Pack the sequences with the same N_i into sequence group SG_j.
    Perform `all2all1` of SG_0. ▷*Overlapping communication*
    for each sequence group SG_j **do**
        Let j be the local rank of device k within process group G_i.
        Launch `all2all1` of $SG_j + 1$. ▷*Overlap Communication*
        Launch `all2all2` of SG_j.
    $\text{loss}_k \leftarrow$ ComputeLoss
    loss_k.backward()
Synchronize gradients and update model parameters.

ancing. Secondly, both the `all-to-all` communications and attention computations of different sequences are independent, providing opportunities for communication computation overlapping.

Key Challenges. However, this hybrid approach introduces two significant implementation challenges. **Firstly, the search space is prohibitively large.** Finding an optimal configuration requires solving a two-level combinatorial problem: (a) *Sharding Degree Selection*, which involves choosing a sharding degree for each of the B sequences, creating a search space that grows exponentially with batch size B. (b) *Device Group Placement* requires assigning a concrete GPU group to each sharded sequence. This is a constrained task analogous to an NP-hard bin-packing problem, as placements for different sequences are coupled and must collectively satisfy per-GPU resource limits. **Secondly, the resulting configuration poses a complex scheduling challenge.** The solution to the placement problem is a heterogeneous plan where sequences are processed by different and potentially overlapping device groups. This breaks the conventional SPMD (Single Program, Multiple Data) paradigm. Since `all-to-all` operations are collective and require synchronization, a naive implementation that serializes the communication for each group would introduce significant GPU idle time (bubbles), diminishing the benefits of hybrid sharding. Therefore, a sophisticated scheduling mechanism is required to manage these diverse computation and communication patterns efficiently.

4.3 Solving the Hybrid Sharding Challenge

Structured device grouping and decoupled assignment. We adopt a simple yet restrictive device-grouping scheme that jointly shrinks the search space and eases scheduling. Concretely, we partition GPUs into disjoint device groups such that: (i) each group size is a power of two, (ii) groups are preferentially formed within a single node to maximize locality, and (iii) groups of different sizes never overlap (a GPU participates in at most one group across all sizes). On 8 GPUs, for example, 2-way groups are uniquely determined as [0,1], [2,3], [4,5], [6,7]. These constraints essentially induce a unique partition, drastically curbing combinatorics and simplifying downstream orchestration. To further lower complexity, we decouple grouping from placement: we first instantiate all admissible device groups, then shard sequences and assign them to groups using a lightweight bucketing heuristic that packs by sharding degree and estimated cost while meeting per-GPU memory limits and balancing both compute and memory across groups.

Deadlock-free overlapped execution. On each GPU, we schedule the communication and computation of assigned sequences to maximize overlap (Fig. 3). For sequences with sharding degree > 1, we process them in descending sharding degree; sequences with the same degree communicate within independent process groups, and this global descending-order discipline eliminates deadlocks

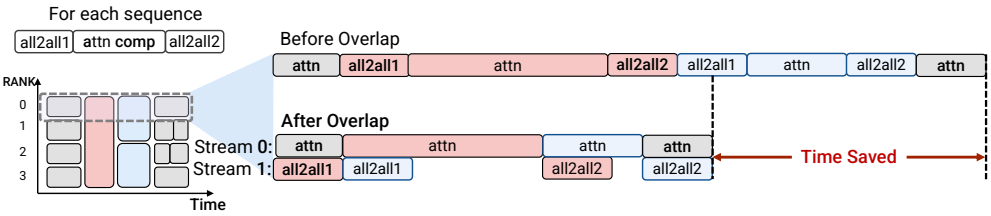

Figure 3: Sliced sequence computation pattern of hybrid sharding and communication overlapping strategies.

and busy waiting. Unsharded sequences (degree $= 1$) are executed first, so their compute overlaps with the first issued communication, which comes from the largest-degree shard, thereby maximizing overlap. We further pipeline communication across sequences: within a sequence the dependency is `all2all1` $\rightarrow$ compute $\rightarrow$ `all2all2`; since sequences are independent, we prelaunch the next sequence's `all2all1` as soon as the current sequence enters compute, overlapping the next sequence's communication with the current sequence's computation.

Vision Tower Balancing. Unlike LLM backbone, the vision tower's workload is inherently parallelizable at the sequence length level. For multi-image inputs, vision encoders typically process images independently by stacking them along the batch dimension. Techniques like dynamic resolution may further tile high-resolution images into smaller, independent images (Wang et al., 2025b). For video inputs, frames are sampled and processed with intra-frame attention, making computation independent across frames. Consequently, both the computational and memory costs of the vision tower scale near-linearly with the number of images or frames. We leverage this property by distributing images and video frames evenly across available GPUs, thereby balancing both compute and memory loads.

4.4 Solving the Dataloading Bottleneck

Implementing Hybrid Sharding requires solving a two-level optimization problem and scheduling the resulting heterogeneous plan.

In frameworks like veRL that employ a hybrid-controller architecture, a single controller is responsible for data loading. This encounters a bottleneck with large data modalities like videos. As the batch size increases to scale up distributed training, the master node's memory becomes a limiting factor, leading to potential CPU out-of-memory. Furthermore, the preprocessing of large video files, which includes decoding and frame sampling, is computationally intensive and exacerbates the bottleneck.

Decentralized Data Pipeline. To address this, we design a distributed dataloader and only transfer **lightweight metadata of multimodal data** through the single controller. At initialization, a lightweight *Proxy Dataloader* is launched on the master node, and a *Local Dataloader* is instantiated on each worker node. Both hold only the **dataset's metadata**, consuming minimal memory. When a batch is requested, the Proxy Dataloader partitions the global batch into shards and distributes these data-loading tasks to the Local Dataloaders. Each Local Dataloader then independently performs the heavy preprocessing tasks—such as decoding, frame sampling, and data augmentation—on its assigned data shard. This distributes both memory and computational loads across the cluster. Once complete, the Local Dataloaders send the metadata of other multimodal data and materialized data of text tensors to the Proxy Dataloader. Then, the scheduler on the single controller operates on the metadata to determine the optimal data placement across GPUs for vision tower balancing. Finally, the single controller transfer of the metadata of multimodal data to their designated GPUs. Each GPU fetches the desired data on the fly from the corresponding node for load balancing

This design ensures that the master node's memory is only used for the lightweight text tensor and metadata of visual data, while the expensive preprocessing is parallelized and the memory bottleneck is alleviated, significantly improving data throughput and scalability.

Table 1: Model and dataset configurations used in our evaluation. In the short video setting, we set `max_frame_per_sample` to 128; In the long video setting, we set it to 512.

Model	ViT	Layers	Hidden Size	Attention Heads	KV Heads	FSDP	Dataset
7B	600M	36	4096	32	8	8	Short/Long Video
32B	600M	64	5120	64	8	16	Short/Long Video

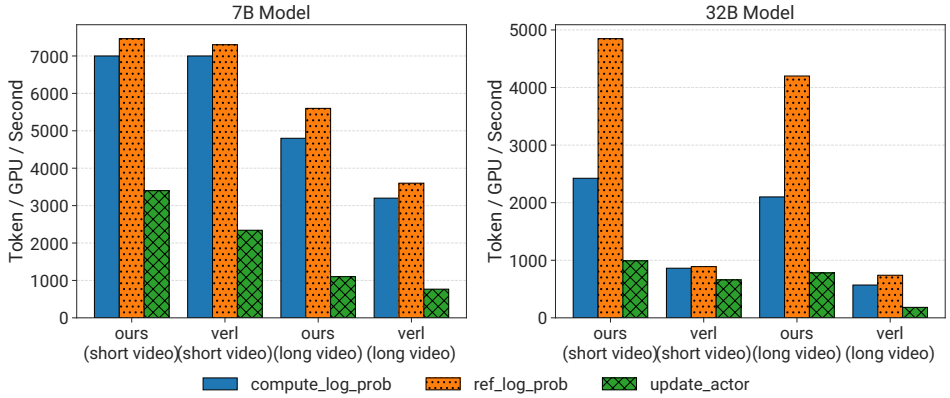

Figure 4: Comparison of token/GPU/second across different datasets, models, and methods.

5 Evaluation

5.1 Evaluation Setup

Implementation. We implement FlexRL in Python on top of veRL (Sheng et al., 2025) and RAY (Moritz et al., 2018) framework, leveraging their distributed computing capabilities for scalable deployment. The core system components are built using PyTorch (Paszke et al., 2019) for tensor operations and automatic differentiation, while communication primitives are implemented using NCCL (ncc, 2023) for efficient GPU-to-GPU communication. Our implementation consists of approximately 8K lines of Python code.

Testbed. We evaluate our system on a high-performance computing cluster comprising 128 NVIDIA H800 GPUs distributed across 16 nodes. Each node is equipped with 8 H800 GPUs (80GB HBM3 memory each) interconnected via high-bandwidth 900GB/s NVLink fabric. Inter-node connectivity is a 3200Gb/s RoCEv2 RDMA network.

Models and Datasets. As shown in Table 1, we evaluate FlexRL on two Qwen-2.5-VL-like VLM variants (7B, 32B) that share the same 600M vision tower using different FSDP sizes. All models are trained on a unified mixture of short-video, long-video, and image-only datasets, representing different real-world scenarios. The 7B model follows Mimo-VL(Team et al., 2025a), because Qwen-2.5-VL-7B uses 28 attention heads, which is unfriendly to head-level sharding in the attention components.

We compare the following methods: (1) **veRL+Bucketing**: The original veRL system without any optimization for short videos.; (2) **veRL+DS Ulysses**: veRL with sequence parallelism for long videos, $sp\_size = 8$; (3) **FlexRL**: automatically decides the sharding degree and computation pattern of each sequence for workload balancing.

5.2 Main Results

We present the training throughput of our methods and baselines in Figure 4. The results clearly demonstrate that FlexRL consistently and significantly outperforms the veRL baseline across all evaluated scenarios, including both 7B and 32B models on short and long video datasets.

The most substantial gains are observed in the forward computation phases. As shown by the ref_log_prob bars, our system dramatically accelerates the forward pass, achieving a speedup of

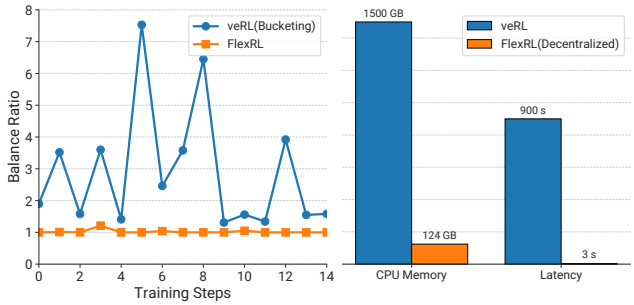

Figure 5: **Left**: Balance ratio of the attention computation across all GPUs. A high balance ratio indicates severe load imbalance, while 1.0 means perfect load balancing. **Right:** CPU memory usage on master node and model inputs transfer latency comparison.

up to 3x compared to veRL. This highlights the efficiency of our workload balancing and sharding strategy in inference-heavy computations.

While the end-to-end training throughput is ultimately bottlenecked by the training phase, FlexRL still delivers a remarkable overall performance improvement. By optimizing the entire pipeline, our system boosts the total training throughput by up to 4.2x, with the peak acceleration observed for the 32B model on the long video dataset. These results validate the effectiveness of our proposed optimizations in enhancing the training efficiency of large-scale video language models.

Workload Balance Study. We further conduct experiments to show how our methods balance the workload. For 7B model with *short videos*, we record the theoretical computation of the attention components of all sequences in each training step. Then, we calculate the balance ratio of each GPU by dividing the total computation of all sequences in the GPU by the average value of all GPUs. The result is shown in Figure 5. Our result demonstrates that our method achieves good load balance through the training steps, while the bucketing algorithm leads to at most 7.5$\times$ imbalance.

6 Conclusion And Discussion

In this work, we present FlexRL, a holistic system that addresses the unique system-level challenges of reinforcement learning for large Vision-Language Models. By systematically analyzing the bottlenecks across the RL pipeline, we identify critical inefficiencies in both data loading and workload balancing that hinder scalability and hardware utilization. FlexRL introduces a decentralized data pipeline to eliminate I/O and memory bottlenecks on the controller, and a novel hybrid sequence sharding mechanism to achieve fine-grained, sub-sequence level load balancing across GPUs. Our efficient scheduling algorithm and dynamic execution engine further maximize overlap between computation and communication, ensuring high throughput even under extreme data heterogeneity.

REFERENCES

Pytorch fullyshardeddataparallel. https://pytorch.org/docs/stable/fsdp, 2023.

Nccl. https://developer.nvidia.com/nccl, 2023.

Yukang Chen, Fuzhao Xue, Dacheng Li, Qinghao Hu, Ligeng Zhu, Xiuyu Li, Yunhao Fang, Haotian Tang, Shang Yang, Zhijian Liu, Yihui He, Hongxu Yin, Pavlo Molchanov, Jan Kautz, Linxi Fan, Yuke Zhu, Yao Lu, and Song Han. LongVILA: Scaling long-context visual language models for long videos. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=wCXAlfvCy6.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025. URL https://arxiv.org/abs/2505.24298.

Hao Ge, Fangcheng Fu, Haoyang Li, Xuanyu Wang, Sheng Lin, Yujie Wang, Xiaonan Nie, Hailin Zhang, Xupeng Miao, and Bin Cui. Enabling parallelism hot switching for efficient training of large language models. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, pp. 178–194, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400712517. doi: 10.1145/3694715.3695969. URL https://doi.org/10.1145/3694715.3695969.

Hao Ge, Junda Feng, Qi Huang, Fangcheng Fu, Xiaonan Nie, Lei Zuo, Haibin Lin, Bin Cui, and Xin Liu. Bytescale: Communication-efficient scaling of llm training with a 2048k context length on 16384 gpus. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, pp. 963–978. ACM, August 2025. doi: 10.1145/3718958.3754352. URL http://dx.doi.org/10.1145/3718958.3754352.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.

Jun Huang, Zhen Zhang, Shuai Zheng, Feng Qin, and Yida Wang. DISTMM: Accelerating distributed multimodal model training. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 1157–1171, Santa Clara, CA, April 2024. USENIX Association. ISBN 978-1-939133-39-7. URL https://www.usenix.org/conference/nsdi24/presentation/huang.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models, 2023. URL https://arxiv.org/abs/2309.14509.

Haoyang Li, Fangcheng Fu, Sheng Lin, Hao Ge, Xuanyu Wang, Jiawen Niu, Jie Jiang, and Bin Cui. Demystifying workload imbalances in large transformer model training over variable-length sequences, 2024. URL https://arxiv.org/abs/2412.07894.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. In *The Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2021)*, 2021.

Qianli Ma, Yaowei Zheng, Zhelun Shi, Zhongkai Zhao, Bin Jia, Ziyue Huang, Zhiqi Lin, Youjie Li, Jiacheng Yang, Yanghua Peng, et al. Veomni: Scaling any modality model training with model-centric distributed recipe zoo. *arXiv preprint arXiv:2508.02317*, 2025.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, pp. 561–577. USENIX Association, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32 of *NeurIPS '19*. Curran Associates, Inc., 2019.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Core Team, Zihao Yue, Zhenru Lin, Yifan Song, Weikun Wang, Shuhuai Ren, Shuhao Gu, Shicheng Li, Peidian Li, Liang Zhao, Lei Li, Kainan Bao, Hao Tian, Hailin Zhang, Gang Wang, Dawei Zhu, Cici, Chenhong He, Bowen Ye, Bowen Shen, Zihan Zhang, Zihan Jiang, Zhixian Zheng, Zhichao Song, Zhenbo Luo, Yue Yu, Yudong Wang, Yuanyuan Tian, Yu Tu, Yihan Yan, Yi Huang, Xu Wang, Xinzhe Xu, Xingchen Song, Xing Zhang, Xing Yong, Xin Zhang, Xiangwei Deng, Wenyu Yang, Wenhan Ma, Weiwei Lv, Weiji Zhuang, Wei Liu, Sirui Deng, Shuo Liu, Shimao Chen, Shihua Yu, Shaohui Liu, Shande Wang, Rui Ma, Qiantong Wang, Peng Wang, Nuo Chen, Menghang Zhu, Kangyang Zhou, Kang Zhou, Kai Fang, Jun Shi, Jinhao Dong, Jiebao Xiao, Jiaming Xu, Huaqiu Liu, Hongshen Xu, Heng Qu, Haochen Zhao, Hanglong Lv, Guoan Wang, Duo Zhang, Dong Zhang, Di Zhang, Chong Ma, Chang Liu, Can Cai, and Bingquan Xia. Mimo-vl technical report, 2025a. URL https://arxiv.org/abs/2506.03569.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, Congcong Wang, Dehao Zhang, Dikang Du, Dongliang Wang, Enming Yuan, Enzhe Lu, Fang Li, Flood Sung, Guangda Wei, Guokun Lai, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haoning Wu, Haotian Yao, Haoyu Lu, Heng Wang, Hongcheng Gao, Huabin Zheng, Jiaming Li, Jianlin Su, Jianzhou Wang, Jiaqi Deng, Jiezhong Qiu, Jin Xie, Jinhong Wang, Jingyuan Liu, Junjie Yan, Kun Ouyang, Liang Chen, Lin Sui, Longhui Yu, Mengfan Dong, Mengnan Dong, Nuo Xu, Pengyu Cheng, Qizheng Gu, Runjie Zhou, Shaowei Liu, Sihan Cao, Tao Yu, Tianhui Song, Tongtong Bai, Wei Song, Weiran He, Weixiao Huang, Weixin Xu, Xiaokun Yuan, Xingcheng Yao, Xingzhe Wu, Xinhao Li, Xinxing Zu, Xinyu Zhou, Xinyuan Wang, Y. Charles, Yan Zhong, Yang Li, Yangyang Hu, Yanru Chen, Yejie Wang, Yibo Liu, Yibo Miao, Yidao Qin, Yimin Chen, Yiping Bao, Yiqin Wang, Yongsheng Kang, Yuanxin Liu, Yuhao Dong, Yulun Du, Yuxin Wu, Yuzhi Wang, Yuzi Yan, Zaida Zhou, Zhaowei Li, Zhejun Jiang, Zheng Zhang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Zijia Zhao, Ziwei Chen, and Zongyu Lin. Kimi-vl technical report, 2025b. URL https://arxiv.org/abs/2504.07491.

Kwai Keye Team, Biao Yang, Bin Wen, Changyi Liu, Chenglong Chu, Chengru Song, Chongling Rao, Chuan Yi, Da Li, Dunju Zang, Fan Yang, Guorui Zhou, Hao Peng, Haojie Ding, Jiaming Huang, Jiangxia Cao, Jiankang Chen, Jingyun Hua, Jin Ouyang, Kaibing Chen, Kaiyu Jiang, Kaiyu Tang, Kun Gai, Shengnan Zhang, Siyang Mao, Sui Huang, Tianke Zhang, Tingting Gao, Wei Chen, Wei Yuan, Xiangyu Wu, Xiao Hu, Xingyu Lu, Yang Zhou, Yi-Fan Zhang, Yiping Yang, Yulong Chen, Zhenhua Wu, Zhenyu Li, Zhixin Ling, Ziming Li, Dehua Ma, Di Xu, Haixuan Gao, Hang Li, Jiawei Guo, Jing Wang, Lejian Ren, Muhao Wei, Qianqian Wang, Qigen Hu, Shiyao Wang, Tao Yu, Xinchen Luo, Yan Li, Yiming Liang, Yuhang Hu, Zeyi Lu, Zhuoran Yang, and Zixing Zhang. Kwai keye-vl technical report, 2025c. URL https://arxiv.org/abs/2507.01949.

MiroMind Foundation Model Team and MiroMind AI Infra Team. Mirorl: An mcp-first reinforcement learning framework for deep research agent. https://github.com/MiroMindAI/MiroRL, 2025.

V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Bin Chen, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiale Zhu, Jiali Chen, Jing Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingdao Liu, Mingde Xu, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong,

Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Tianyu Tong, Wenkai Li, Wei Jia, Xiao Liu, Xiaohan Zhang, Xin Lyu, Xinyue Fan, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yanzi Wang, Yifan An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuting Wang, Yu Wang, Yuxuan Zhang, Zhao Xue, Zhenyu Hou, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025d. URL https://arxiv.org/abs/2507.01006.

Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, Zichen Liu, Haizhou Zhao, Dakai An, Lunxi Cao, Qiyang Cao, Wanxi Deng, Feilei Du, Yiliang Gu, Jiahe Li, Xiang Li, Mingjie Liu, Yijia Luo, Zihe Liu, Yadao Wang, Pei Wang, Tianyuan Wu, Yanan Wu, Yuheng Zhao, Shuaibing Zhao, Jin Yang, Siran Yang, Yingshui Tan, Huimin Yi, Yuchi Xu, Yujin Yuan, Xingyao Zhang, Lin Qu, Wenbo Su, Wei Wang, Jiamang Wang, and Bo Zheng. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library, 2025a. URL https://arxiv.org/abs/2506.06122.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, Zhaokai Wang, Zhe Chen, Hongjie Zhang, Ganlin Yang, Haomin Wang, Qi Wei, Jinhui Yin, Wenhao Li, Erfei Cui, Guanzhou Chen, Zichen Ding, Changyao Tian, Zhenyu Wu, Jingjing Xie, Zehao Li, Bowen Yang, Yuchen Duan, Xuehui Wang, Zhi Hou, Haoran Hao, Tianyi Zhang, Songze Li, Xiangyu Zhao, Haodong Duan, Nianchen Deng, Bin Fu, Yinan He, Yi Wang, Conghui He, Botian Shi, Junjun He, Yingtong Xiong, Han Lv, Lijun Wu, Wenqi Shao, Kaipeng Zhang, Huipeng Deng, Biqing Qi, Jiaye Ge, Qipeng Guo, Wenwei Zhang, Songyang Zhang, Maosong Cao, Junyao Lin, Kexian Tang, Jianfei Gao, Haian Huang, Yuzhe Gu, Chengqi Lyu, Huanze Tang, Rui Wang, Haijun Lv, Wanli Ouyang, Limin Wang, Min Dou, Xizhou Zhu, Tong Lu, Dahua Lin, Jifeng Dai, Weijie Su, Bowen Zhou, Kai Chen, Yu Qiao, Wenhai Wang, and Gen Luo. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency, 2025b. URL https://arxiv.org/abs/2508.18265.

Yujie Wang, Shiju Wang, Shenhan Zhu, Fangcheng Fu, Xinyi Liu, Xuefeng Xiao, Huixia Li, Jiashi Li, Faming Wu, and Bin Cui. Flexsp: Accelerating large language model training via flexible sequence parallelism. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 421–436, New York, NY, USA, 2025c. Association for Computing Machinery. ISBN 9798400710797. doi: 10.1145/3676641.3715998. URL https://doi.org/10.1145/3676641.3715998.

Zheng Wang, Anna Cai, Xinfeng Xie, Zaifeng Pan, Yue Guan, Weiwei Chu, Jie Wang, Shikai Li, Jianyu Huang, Chris Cai, Yuchen Hao, and Yufei Ding. Wlb-llm: Workload-balanced 4d parallelism for large language model training, 2025d. URL https://arxiv.org/abs/2503.17924.

Zhixin Wang, Tianyi Zhou, Liming Liu, Ao Li, Jiarui Hu, Dian Yang, Yinhui Lu, Jinlong Hou, Siyuan Feng, Yuan Cheng, and Yuan Qi. Distflow: A fully distributed rl framework for scalable and efficient llm post-training, 2025e. URL https://arxiv.org/abs/2507.13833.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa:next phase of question-answering to explaining temporal actions, 2021. URL https://arxiv.org/abs/2105.08276.

Zili Zhang, Yinmin Zhong, Yimin Jiang, Hanpeng Hu, Jianjian Sun, Zheng Ge, Yibo Zhu, Daxin Jiang, and Xin Jin. Disttrain: Addressing model and data heterogeneity with disaggregated training for multimodal large language models. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, pp. 24–38, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715242. doi: 10.1145/3718958.3750472. URL https://doi.org/10.1145/3718958.3750472.

Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, Hongyu Zhou, Yimin Jiang, Yibo Zhu, and Daxin Jiang. Streamrl:

Scalable, heterogeneous, and elastic rl for llms with disaggregated stream generation, 2025. URL https://arxiv.org/abs/2504.15930.