

DISTFLOW: A Fully Distributed RL Framework for Scalable and Efficient LLM Post-Training

Zhixin Wang

wangzx@sii.edu.cn

Shanghai Innovation Institute

Zhejiang University

Tianyi Zhou

tyzhou@sii.edu.cn

Shanghai Innovation Institute

Liming Liu

liuliming@sii.edu.cn

Shanghai Innovation Institute

Ao Li

liao@sii.edu.cn

Shanghai Innovation Institute

Jiarui Hu

hujiarui@sii.edu.cn

Shanghai Innovation Institute

Dian Yang

yangdian@sii.edu.cn

Shanghai Innovation Institute

Yinhui Lu

luyinhui@fudan.edu.cn

Shanghai Innovation Institute

Fudan University

Jinlong Hou

houjinlong@sii.edu.cn

Shanghai Innovation Institute

Siyuan Feng

syfeng@sii.edu.cn

Shanghai Innovation Institute

Yuan Cheng

cheng_yuan@fudan.edu.cn

Shanghai Innovation Institute

AI³, Fudan University

Shanghai Academy of AI for Science

Yuan Qi

qiyuan@fudan.edu.cn

Shanghai Innovation Institute

AI³, Fudan University

Shanghai Academy of AI for Science

Abstract

Reinforcement learning (RL) has become the pivotal post-training technique for large language model (LLM). Effectively scaling reinforcement learning is now the key to unlocking advanced reasoning capabilities and ensuring safe, goal-aligned behavior in the most powerful LLMs. Mainstream frameworks usually employ a hybrid-controller architecture where a single-controller dispatches the overall execution logic and manages overall data transfer and the multi-controller executes distributed computation. For large-scale reinforcement learning, minor load imbalances can introduce significant bottlenecks, ultimately constraining the scalability of the system.

To address this limitation, we introduce DISTFLOW, a novel, fully distributed RL framework designed to break scaling barrier. We adopt a multi-controller paradigm that dispatches data transfer and execution tasks to all workers, which eliminates the centralized node. This allows each worker to operate independently, leading to near-linear scalability up to 1024 GPUs and dramatic efficiency gains. Furthermore, our architecture decouples resource configuration from execution logic, allowing each worker to have a unique execution flow, offering significant flexibility for rapid and cost-effective algorithmic experimentation. Extensive experiments show that DISTFLOW achieves excellent linear scalability and up to a 7x end-to-end throughput improvement in specific scenarios over state-of-the-art (SOTA) frameworks.

1 Introduction

The large language model (LLM) and vision language model (VLM) development paradigm starts with pretraining [1] on massive datasets to build a foundation model, followed by supervised fine-tuning (SFT) [2, 3] to teach it instruction following. While this process creates a knowledgeable and task-capable model, it inherently falls short in ensuring reliable alignment with human values and robustly performing complex reasoning. Hence, the paradigm incorporates a third stage: Reinforcement Learning [4], the key technique responsible for the advanced capabilities of modern AI like Deepseek-R1 [5], GPT-4o [6], Gemini 2.5 Pro [7], Claude 4 [8] and Grok 4 [9]. Unlike earlier stages that rely on static data, RL is a dynamic and goal-directed optimization process. It quantifies human preferences by training a reward model, which serves as a guiding signal to shape the model’s output. This process reinforces advanced reasoning capabilities and enforces alignment with human values.

Reinforcement learning algorithms, such as Proximal Policy Optimization (PPO) [10] and Group Relative Policy Optimization (GRPO) [11], have emerged as mainstream approaches in LLM post-training due to their stability and efficiency. A typical RL training iteration consists of three main stages. First, the actor model generates responses to input prompts. Next, these responses are evaluated to compute an optimization signal. This signal often combines a preference-based reward, an advantage estimate to guide the learning direction, and a regularization penalty to ensure training stability. Finally, the actor model is updated

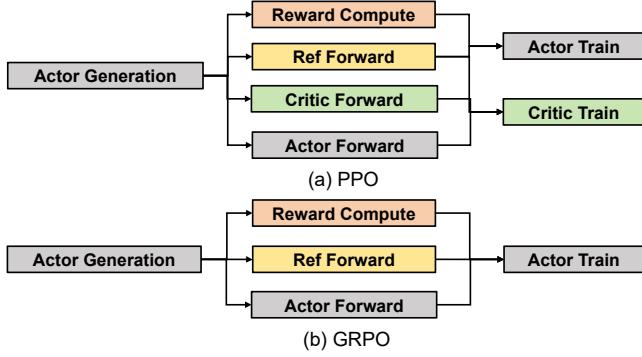


Figure 1. Popular RL algorithms, specifically (a) Proximal Policy Optimization and (b) Group Relative Policy Optimization, can be modeled as a DAG.

using this comprehensive signal to improve its alignment and capabilities.

The overall RL workflow can be modeled as a directed acyclic graph (DAG), where nodes represent computation operations and edges represent data dependencies, as shown in Figure 1. In large-scale systems, varied parallelization strategies across distinct processing stages introduce significant complexity in coordinating data and control flows. Traditional RL systems, such as OpenRLHF [12], employed a disaggregated architecture, partitioning the system into distinct services for inference and training. This architecture enables flexible resource specialization through stage-specific optimization. However, its strict synchronization requirements forcing each stage to wait for the previous one to complete. This sequential execution results in significant resource idleness and low GPU utilization. Moreover, this separation introduces substantial data transfer overhead between the services. These limitations severely decrease the throughput of the system. Researchers adopted collocated architectures to address the efficiency issues of disaggregated architectures. In this paradigm, the generation and training stages are executed on the same set of computational resources, with the system alternating between these two phases. This approach eliminates resource idleness and reduces data communication overhead. Building on this, frameworks like verl [13] have introduced hybrid controller paradigm that merge the flexibility of single-controller with the efficiency of multi-controller, thereby improving system throughput. However, such architectures introduce new challenges. Although hybrid controller evenly dispatches the computation operation to multi-controller, the dataflow is managed by single-controller, including initial dataset loading and the collection and dispatch of vast intermediate data. The centralized mode forces all data to flow through a single node, creating significant I/O and communication overhead that becomes a severe bottleneck. Consequently, when scaling the system to thousands of GPUs, this single-controller

approach is overwhelmed by the massive volume of data, leading to instability and crashes.

To address those major limitations, we introduce DISTFlow, a fully distributed RL training framework with high throughput efficiency and flexible execution pipeline. By adopting a multi-controller paradigm, DISTFlow eliminates the central node common in mainstream frameworks. It distributes data loading, computation, and collection responsibilities evenly across all workers, removing single-node bottlenecks. This decentralized data and computation flow enables the framework to achieve linear scalability up to a 1024 GPU scale and remarkable runtime efficiency.

Another key feature of DISTFlow is its modular pipeline, defined by a user-input DAG. This design completely decouples the algorithm’s logic from physical resource management. Researchers can define their entire RL workflow in a DAG, focusing solely on algorithmic design. The framework then automatically maps this logical graph to the underlying hardware. This maximizes resource utilization and empowers researchers to develop and validate novel algorithms efficiently and cost-effectively.

To validate the effectiveness of our framework, we conduct a comprehensive experimental evaluation. The results show that DISTFlow exhibits exceptional performance and linear scalability across various cluster configurations, ranging from a single node to a thousand-GPU scale. Compared to current SOTA synchronous frameworks, DISTFlow achieves up to a 7x speedup in end-to-end training throughput across different scenarios.

The main contributions of this work can be summarized as follows:

- We analyze the core performance bottlenecks of existing RL frameworks, identifying the centralized dataflow controller as a critical constraint on both scalability and efficiency.
- We introduce DISTFlow, a novel RL framework that achieves high scalability and efficiency through its fully distributed architecture and offers significant flexibility via its DAG-defined design.
- We conduct extensive evaluations of DISTFlow against SOTA systems. Our results demonstrate near-linear scalability up to a 1024 GPU scale and show significant end-to-end throughput improvements across various algorithms, model sizes, and model types, reaching up to 7x in specific scenarios.

2 Background and Motivation

2.1 Reinforcement Learning for LLMs

Models and Workflow. Recent advancements in artificial intelligence have demonstrated that RL provides a powerful framework for enhancing language models beyond their pre-trained capabilities [2, 4], enabling them to better align with human preferences and solve increasingly complex tasks.

During a single optimization iteration in large-scale RL, the Actor Model’s update is computed through the interaction of four core models [14, 15]: the Actor Model itself generates a response, a Reward Model provides preference-based rewards for this response [16], a Critic Model estimates the expected return [17], and a Reference Model imposes a KL-divergence penalty for regularization.

The training process follows a meticulously designed three-step iterative workflow: Generation, Evaluation, and Training. During the Generation phase, the Actor Model receives a batch of prompts as input and auto-regressively generates corresponding text responses for each one. Once response generation is complete, the process advances to the Evaluation phase, where the Reward Model, Reference Model, and Critic Model each provide scores for the current responses. The final Training phase constitutes the core of the RL training process, wherein these three scores are integrated to calculate the advantage function, and through backpropagation, only the parameters of the Actor and Critic are updated, while the Reward and Reference remain frozen throughout the process [18].

Algorithms. Policy Optimization is a fundamental class of RL methods that iteratively refines a model’s policy to maximize the expected cumulative reward based on feedback signals, as illustrated in Figure 1. As a widely-adopted policy optimization algorithm, PPO enhances training stability by employing a clipped surrogate objective function. This mechanism constrains the magnitude of policy updates, thereby preventing destructive changes while maintaining high sample efficiency. Because PPO is reliable and performs well, it has become the accepted standard for fine-tuning large models. GRPO is a variant of PPO that enhances a model’s mathematical reasoning abilities while also optimizing PPO’s memory efficiency. It removes the separate critic model, which uses a lot of computing power, and instead estimates baselines directly from group rewards. This approach trades some baseline accuracy for a large increase in training speed, making it very suitable for large language models that require significant computational resources [5, 11].

Paradigm Shift. While pre-training instills broad knowledge, RL introduces a critical paradigm shift from static next-token prediction to dynamic, goal-oriented optimization defined by reward functions [19–21]. This transition transforms language models into agents that are refined through interaction and feedback, enabling them to improve beyond the limitations of their initial training data [22, 23]. As performance gains from scaling pre-training begin to diminish, large-scale RL has emerged as a new frontier for advancing model capabilities [24–26]. Consequently, applying RL to the largest models has become a core component of the modern development lifecycle, creating an urgent demand for efficient and scalable frameworks to manage its immense computational complexity.

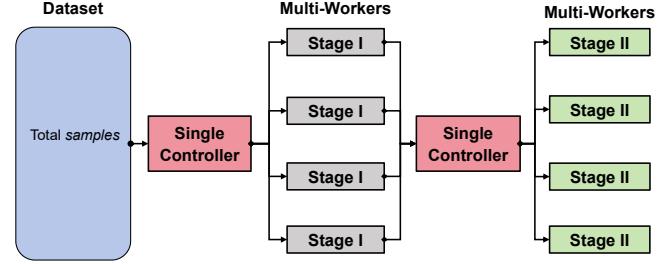


Figure 2. The bottleneck of centralized data management on a single controller. All data operations are flowed through the centralized controller, leading to severe communication overhead and scalability limitations.

2.2 Distributed System Architectures for RL

While common parallelism strategies like Data Parallel (DP), Tensor Parallel (TP), and Pipeline Parallel (PP) [27, 28] distribute the workload, the underlying distributed system architecture is critical for orchestrating the complex flow of data and computation. At the heart of this architectural design is the choice of a controller paradigm, which dictates how tasks are managed across hardware resources and fundamentally shapes how algorithms are implemented, scaled, and optimized.

Controller Paradigms. Distributed machine learning systems, particularly those designed for reinforcement learning, employ different controller paradigms [29] to manage computation across hardware resources. These paradigms fundamentally shape how algorithms are implemented, scaled, and optimized in practice.

Single-Controller. The Single-Controller paradigm employs a centralized controller to manage the overall execution flow of the distributed program. With centralized control logic, users can build core functionalities of the dataflow as a single process, while the controller automatically generates distributed workers to carry out the computation. This approach provides a global view of hardware and dataflow graphs, allowing flexible and optimized resource mapping and execution order coordination among dataflow tasks. However, coordination messages are passed from the controller to all workers, incurring significant dispatch overhead when executing expansive dataflow graphs on large clusters.

Multi-Controller. In contrast, the Multi-Controller paradigm [30] distributes control logic by giving each device (or worker) its own controller. This approach is commonly used in RL frameworks, similar to recent RL training systems [18], where multiple long-running distributed programs operate with each component coordinating execution order through hard-coded data synchronization. While this reduces the central coordination bottleneck, it often results in complex implementation and maintenance challenges, particularly when scaling to large cluster sizes.

2.3 Limitations of existing RL systems

Demanding. In the current era of AI research, training frontier models on thousands of GPUs has become a core requirement. However, systems built on a single-controller dataflow, whether fully centralized or hybrid, are inherently incapable of meeting this demand. This architectural choice creates a severe bottleneck at large scales, leading to instability and failures that fundamentally constrain modern research and development, rendering such designs unsuitable for large-scale AI.

Bottleneck in Single-Controller Paradigm. A critical architectural bottleneck emerges in single-controller paradigm when the central controller node is also tasked with managing the data plane for the entire workflow, as illustrated in Figure 2. In such a design, the controller orchestrates not only the execution flow but also the transfer of all large-scale intermediate data between distributed computational stages. This centralization of the data path forces costly "one-to-all" and "all-to-one" communication patterns, introducing substantial I/O and network overhead that severely degrades system efficiency, particularly for data-intensive tasks like multi-modal or long-text generation. Furthermore, this architecture fundamentally constrains system scalability. During large-scale distributed training, the peak volume of intermediate data can overwhelm the controller node's memory capacity, leading to out-of-memory (OOM) errors and imposing a hard limit on the system's data processing throughput. Consequently, tasking the single controller with data plane management creates a dual bottleneck, simultaneously limiting both the performance and the maximum scale of the entire system.

Rigid Algorithmic Pipeline. Furthermore, the rigid algorithmic pipeline constitutes another limitation. The data flow and control flow in RL systems are inherently complex, and the computational workflow in such frameworks is engineered as a highly integrated, fixed logic that lacks sufficient flexibility. This predefined architectural design forces users to directly engage with the source code for any modifications to the pipeline. This approach not only presents an engineering challenge but also prolongs the iteration cycle for innovative experiments, severely limiting the framework's potential for scientific exploration.

2.4 Design Considerations

These fundamental limitations in scalability and flexibility demonstrate that existing frameworks, which rely on a centralized dataflow controller, are unsuitable for the growing demands of large-scale AI research. In contrast, a multi-controller architecture is inherently well-suited for this challenge, as it enables a fully distributed system where both data and computation can be managed without a central bottleneck. Therefore, the core motivation of this work is to

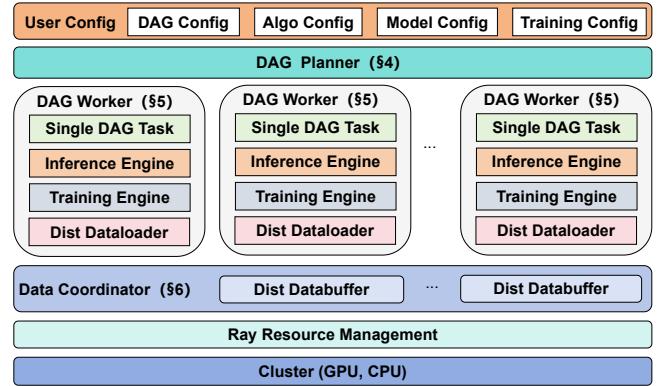


Figure 3. Overview of DISTFLOW.

design a novel framework built upon this principle. We propose a fully distributed architecture designed to address the limitations of traditional systems. Its core features include a decentralized, multi-controller architecture that eliminates the central node, which delivers high throughput and linear scalability. Additionally, DISTFLOW utilizes a modular pipeline defined by a user-input DAG, which decouples the algorithm's logic from physical resource management.

3 DISTFLOW Overview

Based on these design considerations, we propose DISTFLOW, a fully distributed RL framework designed for scalability on large-scale clusters. As illustrated in Figure 3, DISTFLOW employs a multi-controller paradigm that uniformly dispatches all computational and data flow across each GPU. DISTFLOW consists of three main components: a DAG Planner (§4), DAG Workers (§5), and a Data Coordinator (§6).

At its core, this architecture separates the control flow from the data flow. The DAG Planner translates the user's high-level DAG into concrete, executable tasks. These tasks are then executed by the DAG Workers, the primary computational units bound to individual GPUs. Concurrently, the Data Coordinator manages the entire data lifecycle, orchestrating the complex data redistribution required when parallelism strategies change between stages. This separation of concerns is critical as it simplifies the overall system logic, allows for independent optimization of computation and data transfer, and provides greater flexibility for complex workflows.

We implemented our system based on PyTorch [31]. For resource management of GPU and CPU resources, we use Ray [32] which is an open source framework to build and scale ML and Python applications easily. Our system's architecture integrates specialized engines for different stages. We use PyTorch Fully Sharded Data Parallel (FSDP) [31] and Megatron [28] as the training engine. For generation stage, we utilize the vLLM [33] and SGLang [34] inference engines, which are designed for efficient auto-regressive generation.

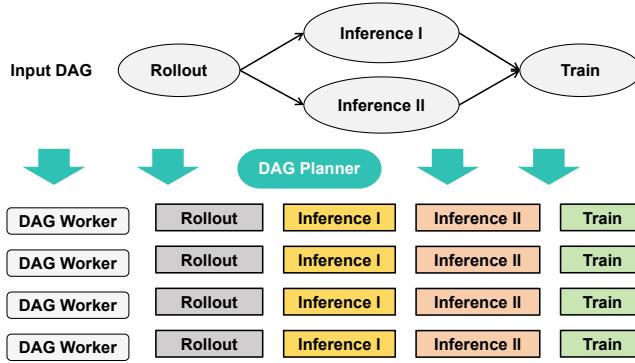


Figure 4. Decomposing a user-defined DAG into a sequential execution pipeline.

To manage these components, and inspired by the hierarchical API design of verl, our system uses the 3DParallelWorker base class.

4 DAG Planner

To address the complexity of implementing diverse RL workflows, our framework is centered on a DAG-defined execution model. The core design principle is to decouple the logical representation of the algorithmic workflow from its physical computation resource. This separation of concerns is achieved through two main components: a declarative DAG interface for users and a backend DAG Planner that translates the logical graph into an executable task chain.

4.1 Input DAG Definition

The framework empowers users to define a complete RL workflow through a configuration file. This file specifies a DAG where each node represents a primitive computational step. The node abstraction is defined by four key attributes: a unique Node ID for identification; a Role (e.g., ACTOR, CRITIC, REWARD, REFERENCE) to specify its functional purpose; a Type (e.g., MODEL_INFERENCE, MODEL_TRAIN, COMPUTE) to clarify the nature of the computation; and Dependencies to establish the execution order and data flow between nodes. By using this high-level abstraction, users can focus on algorithmic logic rather than the low-level complexities of distributed scheduling.

4.2 DAG Decomposition

A primary challenge in executing a user-defined DAG is ensuring its safe and efficient adaptation to a colocated architecture with limited resources, where multiple large models share the same resource pool.

Our framework addresses this challenge through the DAG Planner. Its fundamental responsibility is to translate the logical graph into a concrete, linearized execution pipeline that avoids resource contention and potential OOM errors. To

Node	Computational Function
node_id: "rollout_actor" node_type: "MODEL_INFERENCE" node_role: "ROLLOUT" dependencies: []	def generate(self, batch, **kwargs): batch = self._preprocess_generation_batch(batch) output = generate_sequences(batch) return output
node_id: "function_reward" node_type: "COMPUTE" node_role: "REWARD" dependencies: "rollout_actor"	def compute_reward(self, batch, **kwargs): batch = self._preprocess_generation_batch(batch) reward_tensor = self._defined_reward_func(batch) return reward_tensor
⋮	⋮
node_id: "actor_train" node_type: "MODEL_TRAIN" node_role: "ACTOR" dependencies: "reference_log_prob"	def update_actor(self, batch, **kwargs): batch = self._preprocess_generation_batch(batch) output = self._defined_train_func(gen_batch) return output

Figure 5. Mapping of node definitions to their corresponding execution functions. The DAG Worker dynamically binds a specific computational function to the node based on its attributes.

achieve this, the planner automatically serializes the workflow by analyzing the logical depth of each node. If multiple nodes exist at the same depth, which would imply parallel execution, the planner systematically introduces dependencies to enforce a sequential order. For instance, as illustrated in Figure 4, if an input DAG contains two parallel nodes, Inference I and Inference II, the planner transforms the graph by making one a prerequisite for the other.

5 DAG Worker

The central challenge after defining a logical workflow is its translation into a concrete and extensible execution model. Our system addresses this by introducing the DAG Worker, a core component designed to execute a serialized task chain on a single GPU while providing maximum flexibility for algorithmic experimentation.

The DAG Worker is the framework’s fundamental execution unit. Its design is governed by two key abstractions: a structured lifecycle and a dynamic function dispatch mechanism. The lifecycle is composed of an Initialization phase, where the worker prepares its computational environment, and an iterative Execution phase, where it processes the task chain. The dynamic function dispatch mechanism decouples a node’s logical definition (Role and Type) from its implementation, allowing for a modular and pluggable architecture.

This abstracted design is realized through a concrete operational flow. In the Initialization phase, the worker instantiates its components based on the abstract DAG. It uses a Distributed Dataloader to acquire data, loads the specified models, and initializes backend engines like vLLM, SGLang, PyTorch FSDP, or Megatron. It then materializes the task chain into a concrete execution queue, binding the appropriate function to each node, as illustrated in Figure 5.

During the subsequent Execution phase, the worker enters a loop for each RL iteration. It requests a data batch and sequentially executes each node in the chain. A databuffer

serves as an intermediary state manager, providing necessary inputs to each function and storing its outputs. Upon completing the chain, metrics are aggregated at the global rank 0. The primary benefit of this model is its inherent extensibility. The decoupling of the workflow's structure (defined in the DAG) from its operational logic (resolved by the function mapping) allows researchers to rapidly innovate. For instance, introducing a new reward calculation method or a different policy loss function does not require altering the core dataflow. A researcher can simply implement the new logic in a custom function and map it to a node in the DAG, seamlessly integrating it into the execution pipeline without modifying the surrounding framework.

6 Data Coordinator

In large-scale distributed RL, data management presents a two-fold challenge. First, a centralized approach to initial data loading fundamentally conflicts with a scalable distributed architecture. Forcing a single node to load and then distribute a massive dataset creates an inherent bottleneck, limiting both scalability and efficiency. Second, the dynamic nature of RL workflows, where computational stages like Generation and Training may employ different parallel strategies, necessitates an efficient and correct mechanism for redistributing intermediate data across workers to prevent system stalls or silent training errors.

To address these challenges, our framework introduces a unified Data Coordinator. This coordinator is a high-level abstraction for the entire data lifecycle, composed of two specialized, distributed components: the Distributed Dataloader and the Distributed Databuffer. The Dataloader is responsible for the static, one-time loading of the initial dataset, ensuring data is correctly partitioned at the source. The Databuffer, in contrast, manages the dynamic, transient flow of intermediate data between computational stages, ensuring correct data circulation. Together, these components guarantee load balancing from initial data loading through the entire data flow among DAG Workers. By consolidating all data management logic within Data Coordinator, the framework achieves a clean separation of the data flow from the control flow. This architectural principle is crucial, as it simplifies handling complex data pathways and provides a robust foundation for managing more intricate workflows in the future.

6.1 Distributed Dataloader

In large-scale scenarios, a centralized approach where one node loads the entire dataset is fundamentally inefficient and unscalable. Therefore, to maintain architectural consistency and performance, our framework implements a Distributed Dataloader. The number of Distributed Dataloaders equals the number of DAG Workers, i.e., the number of GPUs. Each dataloader only loads the data required by its corresponding

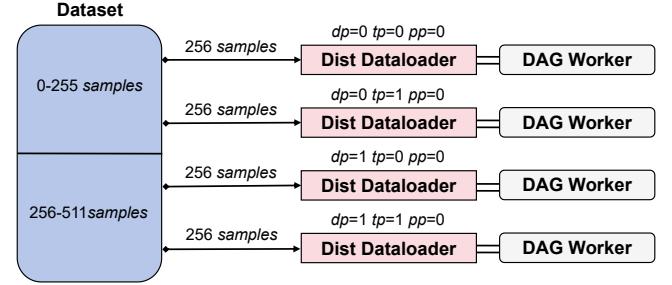


Figure 6. Workflow of Distributed Dataloader. Each worker is only responsible for loading its own assigned piece of the total data.

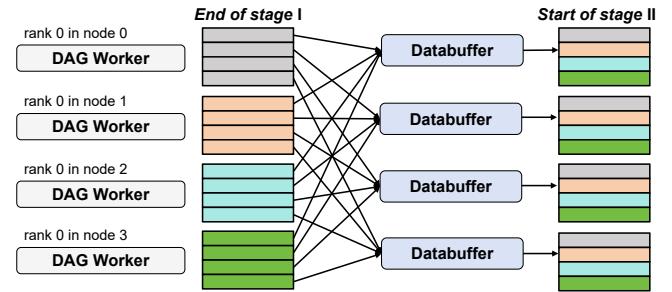


Figure 7. The data redistribution mechanism when DP size changes. This process involves breaking up the data, redistributing the pieces among all buffers, and then reassembling them into new batches.

DAG Worker during the rollout stage, avoiding any redundant data. This approach inherently avoids single-node memory bottlenecks and achieves higher data loading efficiency through parallelism.

During initialization, the Distributed Dataloader queries the parallelism strategy of its associated worker. It then partitions the global dataset into a number of shards equal to the DP size. Based on its DAG worker's DP rank, each Distributed Dataloader identifies and exclusively loads the appropriate shard. Figure 6 illustrates this logic, showing how workers belonging to different DP groups access distinct regions of the dataset and load them in parallel.

6.2 Distributed Databuffer

The Distributed Databuffer, a core component for data flow, is responsible for data redistribution between RL stages. One instance is allocated per node and shared by local workers. Its primary function is to act as a parallelism-aware intermediary, ensuring both the correctness and efficiency of data flow during stage transitions where the DP sizes of consecutive stages may differ.

The operational logic begins after a computation stage completes. To avoid data redundancy from multiple model replicas, only the DAG Worker with a TP rank of 0 places its

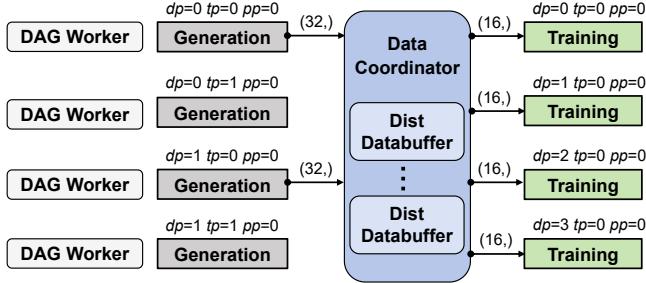


Figure 8. Workflow of Distributed Databuffer. The databuffer handles data redistribution between stages with different DP sizes. It collects data from the previous stage and reorganizes it to fit the needs of the next one.

generated data into its local databuffer. Its operational path is then determined by a key condition: whether the DP size changes for the subsequent stage. When the DP size remains unchanged, it executes a fast-path operation using shared memory for minimal overhead. In contrast, if the DP size differs, it initiates a more complex redistribution process. This process uses a cluster-wide all-to-all communication pattern to correctly re-partition the data according to the new DP configuration, as conceptualized in Figure 7.

Therefore, when the DAG Workers for the next stage request their data, the databuffer distributes the appropriate slices to its intra-node workers according to their new DP rank. Figure 8 provides a concrete example of this process. The figure illustrates a scenario where data from a Generation stage (DP=2) is automatically collected and repartitioned for a subsequent Training stage with a different parallelism strategy (DP=4). This automated handling ensures correct data flow and load balancing for any stage transition.

7 Evaluation

7.1 Experiments Setup

We conduct a series of experiments to evaluate DISTFlow’s efficiency and scalability across four key scenarios. First, we assess overall performance on language models from 7B to 72B using PPO and GRPO algorithms, scaling up to 128 GPUs. Second, we test the linear scalability of DISTFlow with VLMs on up to 1024 GPUs using the GRPO algorithm; this is performed only on DISTFlow as the baseline system encounters OOM errors under the same global batch size. Third, we compare DISTFlow’s performance against the baseline using the maximum batch sizes supported by the baseline in a multi-modal setting. Fourth, we evaluate performance in long-context scenarios with context lengths from 8K to 64K tokens to highlight DISTFlow’s dataflow optimizations. Finally, a convergence test is run for 20 epochs to ensure that DISTFlow’s performance improvements do not compromise model accuracy.

Testbed. We deploy DISTFlow on a cluster with 128 nodes, where each node is equipped with 8 NVIDIA Hopper GPUs interconnected with NVLink. The nodes are connected by an RDMA network over RoCE v2. Our evaluation is conducted under the software settings with PyTorch 2.6.0, CUDA 12.6, vLLM 0.8.5.post1, and NCCL 2.21.5.

Models and Algorithms. We evaluate system performance using the PPO and GRPO algorithms. For the PPO experiments, a function reward is utilized in place of a reward model, with the critic model matching the actor’s size. We use the Qwen-2.5-Instruct series for language models and the Qwen-2.5-VL-Instruct series for VLMs, with model sizes of 7B, 32B, and 72B.

Datasets. For language model setting, we use DeepScaleR-Preview-Dataset [35], which contains about 40,000 unique math problems, while for VLM experiments, we choose MM-Eureka-Dataset [36]. All experiments are under the default maximum prompt length 2048, and the maximum response length 4096, with padding applied to shorter responses.

Baseline. We benchmark DISTFlow against verl [13] v0.4.0, a state-of-the-art RL training system. Other frameworks [12, 27, 37] are not selected for comparison due to their lower throughput relative to verl. Both DISTFlow and verl use vLLM as an inference engine and PyTorch FSDP as the training backend. The primary performance metric is throughput, measured in tokens per second, and is calculated from the total tokens in a global batch divided by the time for one iteration. Results are averaged over several iterations following a warm-up period to ensure accuracy.

7.2 End-to-End Evaluation

In our end-to-end evaluation, shown in Figure 9 and Figure 10, DISTFlow consistently outperforms the baseline across all tested configurations. Our framework’s advantage is most pronounced in data-intensive scenarios. For PPO algorithm (Figure 9), we achieve 1.09x - 1.64x speedup comparing to the baseline. Remarkable, with the GRPO algorithm (Figure 10), which involves a larger data volume, DISTFlow achieves a speedup of up to 2.62x. This highlights how our architecture excels where the baseline’s centralized data handling fails.

This performance gap widens as computational resources increase. As we scale to more GPUs, the baseline’s single-node bottleneck becomes more severe, lowering its throughput. In contrast, DISTFlow’s performance scales effectively, with its speedup over the baseline increasing with the GPU count. The baseline’s architectural limits are made clear when it produces an OOM error with the 72B model on 32 GPUs, a task DISTFlow handles without issue. Additionally, smaller models, which have a higher communication-to-compute ratio, benefit most. By optimizing the dataflow that

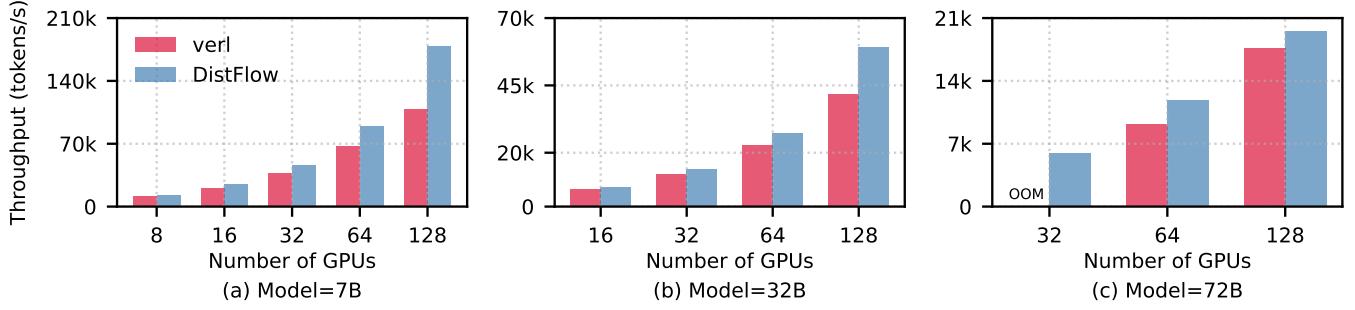


Figure 9. Throughput comparison of DISTFlow and verl using the PPO algorithm. The results show that DISTFlow is faster than the baseline for all tested model sizes and GPU counts. This speedup increases as more GPUs are added, and DISTFlow can successfully complete large-scale tasks.

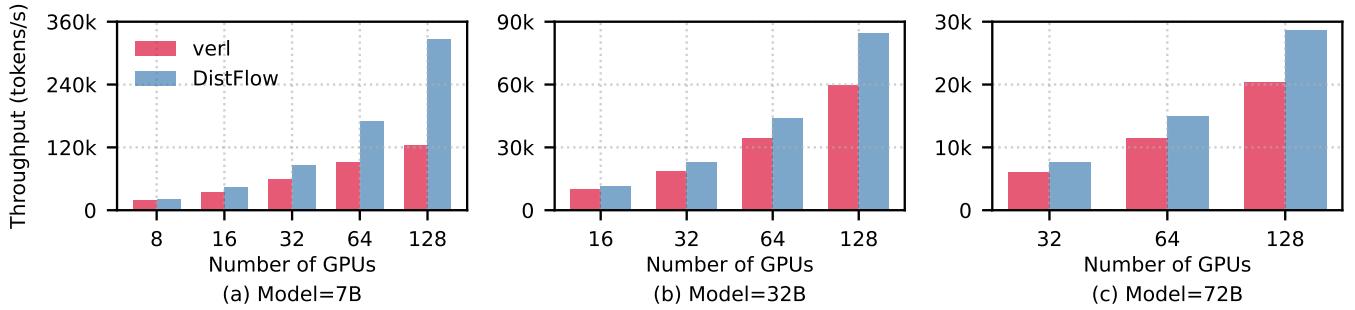


Figure 10. Throughput comparison of DISTFlow and verl using the GRPO algorithm. Throughput comparison of DISTFlow and verl using the GRPO algorithm. With this more data-intensive algorithm, DISTFlow's speed advantage becomes even greater, as its distributed data system handles the increased data load more efficiently.

constitutes a larger portion of their runtime, DISTFlow delivers a 2.26x speedup for the 7B model on 128 GPUs, demonstrating the profound efficiency gains of our distributed approach.

7.3 Scalability Evaluation

The practical benefits of DISTFlow's architecture are clearly demonstrated in our scalability experiments, conducted using the GRPO algorithm with VLMs on the MM-Eureka-Dataset. In this experiment, we scale the global batch size proportionally with the number of nodes. As shown in Figure 11, the resulting performance (solid line) closely tracks the ideal linear scalability curve (dotted line). We quantify this linearity using the Scaling Efficiency metric, defined as follows:

$$\text{Scaling Efficiency} = \frac{T_2/T_1}{N_2/N_1} \times 100\% \quad (1)$$

where T is throughput and N is the number of GPUs, with (N_1, T_1) representing the baseline and (N_2, T_2) the scaled configuration. The results show excellent linearity across all model sizes. Specifically, the system achieves a remarkable scaling efficiency of 93.9% for the 32B model.

Table 1. Maximum global batch size supported by the baseline at different GPU scales.

Model	# GPUs	Global Batch Size
7B	32	1024
	64	512
	128	256
	256	64
32B	64	512
	128	256
	256	64
	512	32
72B	128	256
	256	64
	512	32
	1024	16

The experimental results from our scalability evaluation directly highlight the benefits of our framework's fully distributed design. Figure 11 demonstrates near-linear scaling, a critical capability for efficient large-scale training from 32 GPUs up to 1024 GPUs. Such consistent performance

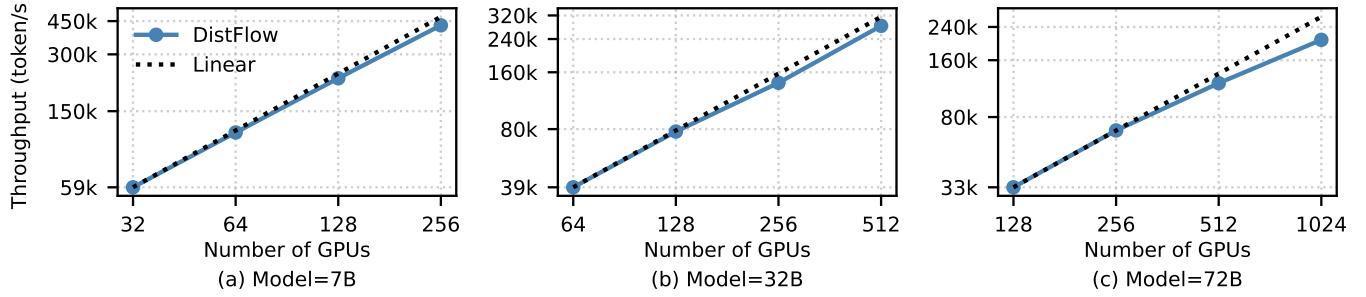


Figure 11. Scalability evaluation of DISTFlow. This experiment shows that DISTFlow achieves near-linear scalability on large clusters of up to 1024 GPUs. This strong performance is attributed to its fully distributed architecture, which uniformly balances both computational and dataflow workloads to maintain high efficiency at scale.

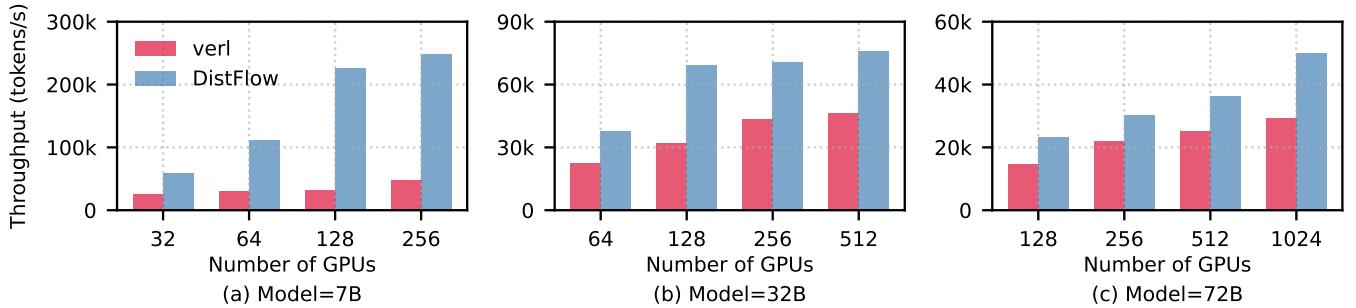


Figure 12. Performance comparison using baseline-constrained maximum batch sizes. To account for the baseline’s OOM errors, this evaluation was constrained to the maximum batch size supported by the baseline. Under these conditions, DISTFlow still demonstrated a substantial performance advantage, achieving a speedup of up to 7x.

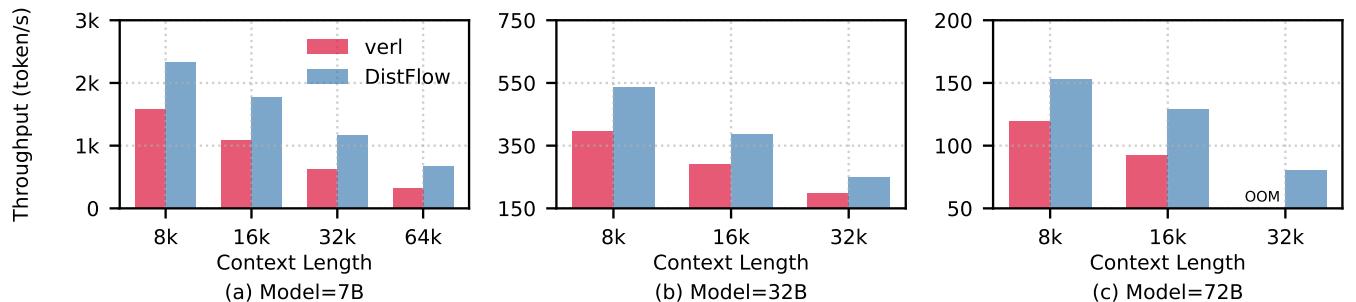


Figure 13. Long-context performance evaluation. The results show that DISTFlow’s performance advantage over the baseline increases with longer context lengths.

indicates that the system effectively distributes all workloads, including data communication, thus avoiding the bottlenecks that typically hinder performance as a cluster grows. In contrast, the baseline system could not complete the same linearity tests, encountering OOM errors.

A direct comparison under the ideal scaling configuration was infeasible. Consequently, we designed an experiment to provide a direct performance comparison. For this test, we first identify the maximum global batch size the baseline can support at each cluster scale and benchmark both systems under those conditions, with the specific global batch sizes

detailed in Table 1. The results, presented in Figure 12, show that DISTFlow is significantly faster than the baseline, especially in VLM settings and on large-scale clusters, achieving a speedup of up to 7x. This performance gap underscores the critical advantage of DISTFlow’s fully distributed approach.

7.4 Long-Context Evaluation

Long-context capability is a critical frontier for LLMs, particularly for the development of advanced agent systems that must process extensive histories or documents. These scenarios create highly data-intensive workloads where the sheer

volume of token data can overwhelm a system’s communication fabric. Our evaluation (Figure 13) in these long-context settings demonstrates that DISTFlow’s fully distributed dataflow provides a significant and scalable advantage. By allowing each node to manage its portion of the data, our framework avoids the severe communication overhead that troubles centralized systems, where all data must be funneled through a single, congested point.

The results, presented in Figure 13, confirm this advantage empirically. For the 7B model, DISTFlow’s throughput speedup over the baseline progressively grows from 1.48x at an 8k context length to an impressive 2.03x at 64k. This clear trend is highly significant; it shows that as the data volume and complexity of the task increase with longer contexts, the efficiency gains from our distributed design become even more pronounced. This directly implies that for future, more demanding applications, DISTFlow’s architectural superiority is an even greater asset.

Furthermore, the baseline system encounters a critical OOM error with the 72B model at a 32k context length, a demanding task that DISTFlow handles without issue. This is not merely a performance dip but a fundamental breakdown, which underscores the scalability limitations of a centralized data management approach. This failure highlights a practical ceiling on the complexity that such systems can manage. In contrast, DISTFlow’s ability to complete the task demonstrates its superior robustness and its capacity to push the boundaries of what is possible in data-intensive, long-context scenarios.

7.5 Convergence

To ensure performance improvements do not compromise model accuracy, we conduct an experiment comparing verl and DISTFlow. The experiment trains a 32B model using the GRPO algorithm on 32 GPUs with the DeepScaleR-Preview-Dataset for 20 epochs, for a total of 700 steps, results shown in Figure 14. With identical hyperparameters, the reward and entropy curves for DISTFlow and the baseline followed the same trajectory. Under this experimental configuration, DISTFlow achieved the same results as the baseline while reducing the total execution time by 21%. This demonstrates that the efficiency and scalability gains from DISTFlow come at no cost to training accuracy.

8 Related Works

RL Training Frameworks. The system architecture of large language model reinforcement learning training frameworks has undergone significant evolution. Early frameworks, such as DeepSpeed-Chat [38], OpenRLHF [12], and NeMo-Aligner [37], explored different architecture. While DeepSpeed-Chat pioneered the colocated architecture that deploys all computation stages on the same set of GPUs with time-sharing scheduling, OpenRLHF and NeMo-Aligner

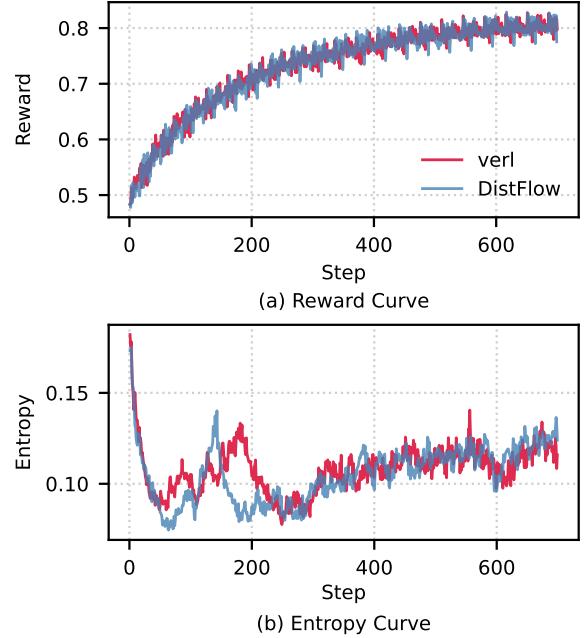


Figure 14. Reward and Entropy curves comparison between verl and DISTFlow. With the same hyperparameters, the training curves show the same trend, which indicates that the improvements in training efficiency and scalability in DISTFlow do not affect training accuracy.

adopted a disaggregated architecture, where different models (e.g., Actor, Critic) are deployed on separate GPUs. The disaggregated design, though straightforward to implement, suffers from severe pipeline bubbles and resource underutilization due to serial dependencies between stages. To enhance resource efficiency, subsequent research further refined the colocated architecture, including verl [13]. Building on this, researchers have proposed deep optimization techniques: RLHFuse [39] introduces subtask fusion to further reduce pipeline bubbles, while other works focus on optimizing the efficiency of model weight switching across different stages [13]. Nonetheless, the inherent resource coupling bottleneck of the colocated design has spurred recent frameworks such as StreamRL [40] and AREAL [41] to revisit the disaggregated architecture, exploiting techniques like stream generation and asynchronous pipelines to unlock its potential for large-scale and heterogeneous environments.

Dataflow Architecture. The multi-model, multi-stage process of large-scale RL training essentially presents a complex dataflow challenge, whose efficient execution depends on advanced orchestration and resource management technologies. The distributed framework Ray [32], with its flexible dynamic task graph and actor model, has become a core component in orchestrating workflows for frameworks like OpenRLHF [12] and other modern RL systems [13]. To enable effective resource coordination, contemporary frameworks

often introduce abstractions such as resource pools [13] to virtualize and manage GPU resources, thereby supporting flexible model placement strategies. To further optimize the efficiency of data execution within the workflow, modern dataflow systems such as Naiad [42] offer asynchronous execution paradigms. StreamRL [40] applies these ideas in the context of large-scale RL training, achieving deep inter-stage overlap through streaming and asynchronous pipeline mechanisms, which effectively reduces waiting overheads.

9 Conclusion

This paper introduces DISTFLOW, a novel framework designed to address the scalability and flexibility challenges in large-scale RL training. To tackle the common single-controller dataflow bottleneck in existing approaches, we propose a fully distributed architecture. At its core, DISTFLOW employs a multi-controller paradigm that uniformly dispatches tasks such as data loading, computation, and intermediate data transfer across all worker nodes, thereby completely eliminating the central bottleneck and achieving near-linear scalability. Furthermore, to grant researchers greater flexibility, DISTFLOW introduces a modular pipeline driven by a user-defined DAG. This design decouples algorithmic logic from physical resource management, significantly accelerating the experimentation and iteration cycle for novel algorithms. Experiments demonstrate that DISTFLOW achieves up to a 7x end-to-end throughput improvement compared to SOTA frameworks. We believe this work paves the way for large-scale RL research by offering a more efficient, flexible, and truly scalable solution.

References

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [2] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS ’22, Red Hook, NY, USA, 2022. Curran Associates Inc.
- [3] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharar Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.
- [4] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [5] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiaoshi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghai Lu, Shangyan Zhou, Shanhua Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.
- [6] OpenAI. Gpt-4o. <https://openai.com/index/hello-gpt-4o/>, 2024.
- [7] Google Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025.
- [8] Anthropic. Introducing claudie 4. <https://www.anthropic.com/news/claudie-4>, 2025.
- [9] xAI. Grok 4. <https://x.ai/news/grok-4>, 2025.
- [10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [11] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.
- [12] Jian Hu, Xibin Wu, Weixun Wang, Dehao Zhang, and Yu Cao. Open-RLHF: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.
- [13] 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, page 1279–1297. ACM, March 2025.

- [14] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022.
- [15] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairei, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esibou, Jude Fernandes, Jeremy Fu, Wenying Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Sagar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yunling Mao, Xavier Martinet, Todor Miaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poult, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.
- [16] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [17] Volodymyr Mnih, Adrià Puigdomènec Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.
- [18] 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, 2025.
- [19] Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction, 2018.
- [20] Amelia Glaese, Nat McAlleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, Abigail See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Soňa Mokrá, Nicholas Fernando, Boxi Wu, Rachel Foley, Susannah Young, Jason Gabriel, William Isaac, John Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. Improving alignment of dialogue agents via targeted human judgements, 2022.
- [21] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: transformers for longer sequences. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [22] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020.
- [23] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022.
- [24] Angeliki Lazaridou, Anna Potapenko, and Olivier Tielemen. Multi-agent communication meets natural language: Synergies between functional and structural language learning. *arXiv preprint arXiv:2005.07064*, 2020.
- [25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.
- [26] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, page 6244–6251. IEEE Press, 2018.
- [27] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '20*, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery.
- [28] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.
- [29] Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Dan Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, Brennan Saeta, Parker Schuh, Ryan Sepassi, Laurent El Shafey, Chandramohan A. Thekkath, and Yonghui Wu. Pathways: Asynchronous distributed dataflow for ml, 2022.
- [30] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI'14*, page 583–598, USA, 2014. USENIX Association.
- [31] 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 H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [32] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, 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)*, 2018.
- [33] Woojeong Kwon, Zhihao Li, Shizhuo Zhuang, Yuze Sheng, Liuyi Zheng, Changhoon Yu, Yiyang Chen, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.
- [34] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson,

- L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 62557–62583. Curran Associates, Inc., 2024.
- [35] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. <https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2>, 2025. Notion Blog.
- [36] Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quan-feng Lu, Daocheng Fu, Tiancheng Han, Botian Shi, Wenhui Wang, Junjun He, Kaipeng Zhang, Ping Luo, Yu Qiao, Qiaosheng Zhang, and Wenqi Shao. Mm-eureka: Exploring the frontiers of multimodal reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2503.07365*, 2025.
- [37] Gerald Shen, Zhilin Wang, Olivier Delalleau, Jiaqi Zeng, Yi Dong, Daniel Egert, Shengyang Sun, Jimmy Zhang, Sahil Jain, Ali Taghibakhsh, Markel Sanz Ausin, Ashwath Aithal, and Oleksii Kuchaiev. Nemo-aligner: Scalable toolkit for efficient model alignment, 2024.
- [38] Zhewei Yao, Reza Yazdani Aminabadi, Olatunji Ruwase, Samyam Rajbhandari, Xiaoxia Wu, Ammar Ahmad Awan, Jeff Rasley, Minjia Zhang, Conglong Li, Connor Holmes, Zhongzhu Zhou, Michael Wyatt, Molly Smith, Lev Kurilenko, Heyang Qin, Masahiro Tanaka, Shuai Che, Shuaiwen Leon Song, and Yuxiong He. Deepspeed-chat: Easy, fast and affordable rlhf training of chatgpt-like models at all scales, 2023.
- [39] Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, and Xin Jin. Optimizing rlhf training for large language models with stage fusion, 2025.
- [40] 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 Dixin Jiang. Streamrl: Scalable, heterogeneous, and elastic rl for llms with disaggregated stream generation, 2025.
- [41] 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.
- [42] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martin Abadi. Naiad: a timely dataflow system. *SOSP ’13*, page 439–455, New York, NY, USA, 2013. Association for Computing Machinery.