

xRLHF: Efficient RLHF with Serverless Rollout Actors

Rui Wei
Stevens Institute of Technology
Hoboken, USA
rwei7@stevens.edu

Hanfei Yu
Stevens Institute of Technology
Hoboken, USA
hyu42@stevens.edu

Devesh Tiwari
Northeastern University
Boston, USA
d.tiwari@northeastern.edu

Jian Li
Stony Brook University
Stony Brook, USA
jian.li.3@stonybrook.edu

Seung-Jong Park
Missouri University of
Science and Technology
Rolla, USA
seung-jong.park@mst.edu

Hao Wang
Stevens Institute of Technology
Hoboken, USA
hwang9@stevens.edu

Abstract

Reinforcement Learning from Human Feedback (RLHF) is widely used to align Large Language Models (LLMs) with human preferences and can also improve reasoning on complex tasks. However, RLHF training is highly resource-intensive because inference and training coexist, creating dynamic resource demands. Existing RLHF frameworks rely on serverful infrastructures, which handle this variability poorly and often waste resources because of idle time during synchronous training.

We present xRLHF, the first scalable training framework for synchronous RLHF, built with a centralized learner and serverless rollout actors. xRLHF adapts to dynamic resource demands, pre-computes shared prefixes to avoid repeated computation, and uses a cost-aware actor scaling strategy and efficient workload assignment to reduce cost and idle time. Experiments show that xRLHF speeds up training by 1.14–1.35 $\times$ while reducing cost by 25.81–44.8% on the physical cluster with 16 L40S GPUs, and achieves 1.23 $\times$ speedup with 38.7% lower cost in the large-scale Vidur simulation.

CCS Concepts

• **Computer systems organization** $\rightarrow$ **Cloud computing**; • **Computing methodologies** $\rightarrow$ *Reinforcement learning*.

Keywords

reinforcement learning from human feedback, serverless computing, rollout scheduling, large language models

ACM Reference Format:

Rui Wei, Hanfei Yu, Devesh Tiwari, Jian Li, Seung-Jong Park, and Hao Wang. 2026. xRLHF: Efficient RLHF with Serverless Rollout Actors. In *ACM Symposium on Cloud Computing Volume 2 (SoCC '26 Vol. 2)*, November 18–20, 2026, Singapore, Singapore. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3828161.3857868>

1 Introduction

Motivation. In the latest Large Language Models (LLMs) [13, 46, 47, 64, 85], Reinforcement Learning from Human Feedback (RLHF)

has been widely adopted to enhance performance on challenging tasks such as mathematical problem solving [47, 71], scientific question answering [25, 59], and code generation [13, 14]. RLHF training couples heavy LLM inference with distributed model updates, and is significantly more resource-intensive and complex than traditional reinforcement learning (RL). While both follow a similar staged training paradigm, existing RL frameworks are not designed for the large-scale demands of RLHF. As model sizes grow and training is distributed across multiple GPUs and machines, the resource utilization and data transmission overhead become major bottlenecks.

Limitations of existing approaches. RL separates data sampling from model updates, but the two processes remain interdependent. Model updates must wait for sampling to finish in order to use the most recent training data, while sampling is delayed by model updates because it needs synchronized model weights. While traditional RL training already suffers from resource idleness on serverful frameworks [35, 40], RLHF amplifies this inefficiency due to the demands of running LLMs and heavy data transmission. More recent RLHF systems [27, 59, 89] improve efficiency through careful model placement and parallelism strategies, but they rely on serverful infrastructures with static resource allocation, which inherently suffer from unavoidable resource inefficiency and cannot dynamically adapt to the execution characteristics of RLHF. Serverless RL frameworks [75–77] demonstrate that dynamically launching and releasing components on demand can substantially reduce idleness and training costs. However, existing serverless solutions for traditional RL cannot be directly applied to RLHF systems, and building a serverless RLHF framework is challenging due to fundamental differences between RLHF and traditional RL, such as the model scale and transmission overhead mentioned above. Moreover, the auto-regressive LLM generation and the continuously updated model weights lead to dynamic resource demands, both within a single step and across different training steps.

Key insights and contributions. Existing RLHF algorithms are highly sampling-heavy, where sequence generation dominates both execution time and resource consumption throughout training [57, 78, 80]. By analyzing existing synchronous RLHF workloads, we identify three key insights. First, tight dependencies between sampling and learning, combined with repeated key-value (KV) cache computation during parallel generation, lead to substantial and unnecessary resource wastage. Second, long-tail responses cause stragglers during sequence generation and leave allocated resources idle.



This work is licensed under a Creative Commons Attribution 4.0 International License. SoCC '26 Vol. 2, Singapore, Singapore
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2816-7/2026/11
<https://doi.org/10.1145/3828161.3857868>

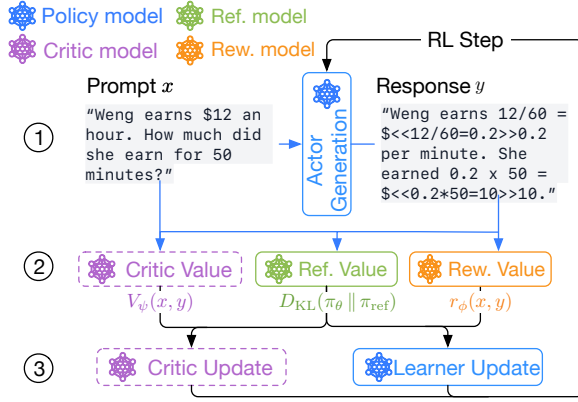


Figure 1: RLHF's dataflow in one training step, including the generation phase, preparation phase, and learning phase. Some details, like the use of the critic model, can vary depending on the specific algorithm used.

Third, workloads vary across training steps, as response lengths for the same prompt may increase or decrease over time, making static resource allocation inefficient. Motivated by these observations, we present xRLHF, the first scalable framework for synchronous RLHF with serverless rollout actors, which improves efficiency and reduces cost. Before each training step, xRLHF constructs an optimized generation plan. It first analyzes the prompt dataset and training configuration to deduplicate KV cache computation under parallel generation. Next, xRLHF estimates the upcoming workload and dynamically scales the number of sampling actors¹ to balance training speed and cost. xRLHF further groups prompts with similar predicted response lengths to mitigate stragglers and reduce intra-actor imbalance. During execution, xRLHF places actors with locality awareness, prioritizing heavy workloads on nearby GPUs to better overlap communication and computation. We summarize our main contributions as follows:

- We present xRLHF, the first scalable framework for synchronous RLHF with serverless actors, hosted on multi-GPU clusters, with a specialized prewarming and keep-alive strategy.
- We design a deduplicated prefill mechanism that eliminates redundant KV cache computation in RLHF.
- We propose a response-length predictor with cut-and-migrate backoff to guide workload scheduling and actor scaling.
- We implement xRLHF on top of VERL [59] and evaluate it on both a physical cluster with two 8-GPU instances and a simulated large-scale cluster. Extensive experiments show that xRLHF reduces cost by 25.81–44.8% and improves training speed by 1.14–1.35 $\times$ compared to the VERL baseline.

2 Background and Motivation

2.1 RLHF vs. General RL

Reinforcement Learning from Human Feedback (RLHF) is a method used to align the behavior of a language model with human preferences by incorporating feedback into the training loop. Rather

¹An actor, also known as a rollout worker, is a separate component that hosts a model replica for response generation.

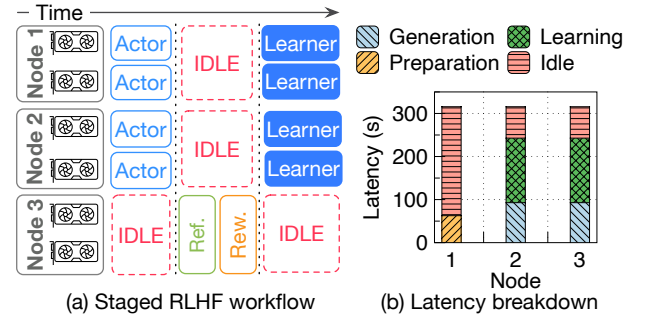


Figure 2: (a) RLHF's staged workflow leaves components idle. (b) Idle RLHF components waste resources.

than relying solely on supervised data, RLHF frames the alignment task as an RL problem, where a policy model (i.e., the LLM) π_θ is optimized to generate outputs that maximize the predicted reward provided by the pre-trained Reward Model (RM) $r_\phi(x, y)$. RLHF training contains multiple iterative training steps, and each training step involves three main phases, shown in Fig. 1:

Phase-1 Generation: For a batch of prompts x from the prompt dataset, multiple sampling **actors** will be launched using the policy π_θ to generate one or more candidate responses y . Each actor maintains its own replica of the policy model.

Phase-2 Preparation: The RM $r_\phi(x, y)$ assigns a scalar score to each generated response. To prevent the policy from deviating too far from its original behavior, many algorithms [33, 57, 85] incorporate a Kullback–Leibler (KL) divergence penalty between the current policy π_θ and the reference model π_{θ_0} .

Phase-3 Learning: A **learner**, with a training engine [51, 82], is responsible for computing gradients and updating the policy model using the data produced from previous phases.

Why do existing RL frameworks fall short? Existing RL frameworks [35, 75–77] are not designed to support the scale and complexity of RLHF for LLMs. Traditional RL workloads involve small models that can be trained on a small set of GPUs and rely on simple, synchronous architectures. In contrast, LLM training requires distributing models across multiple GPUs using techniques like data, tensor, and pipeline parallelism [27, 51, 59, 82], along with specialized inference [31, 60, 84] and training [51, 82] engines. Moreover, scaling up the sampling actors in RLHF increases the need for synchronized replicas, which leads to expensive and slow weight synchronization [59]. In such a case, various RLHF-optimized frameworks [27, 58, 59, 68, 74] have been implemented to address those new challenges.

2.2 Limitations of Serverful RLHF Systems

RLHF faces several challenges from both its staged RL workflow and the use of LLMs. The dependencies across RLHF phases can lead to GPU idleness, while the continually updated policy model and varying response lengths create dynamic resource demands. Existing RLHF frameworks [27, 59, 91] rely on serverful infrastructures and cannot address these inefficiencies.

1) Between phases: Idle components waste resources on serverful platforms. In RLHF, components from different phases depend on each other, which often leaves GPU resources idle during

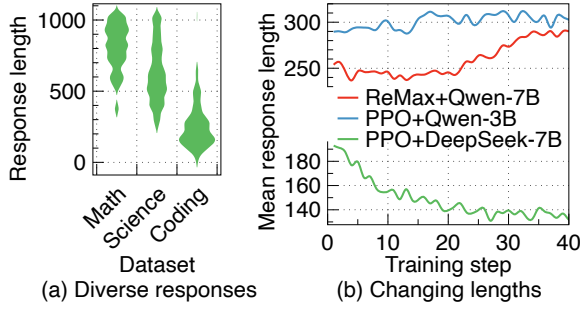


Figure 3: Dynamic resource demands caused by: (a) significantly varying response lengths across and within different types of datasets [30, 39, 53]; (b) changing response lengths.

parts of a training step. Fig. 2(a) shows the workflow of Group Relative Policy Optimization (GRPO) [57] under standalone placement, where each model uses a separate set of GPUs. Fig. 2(b) shows the latency breakdown of a VERL [59] training step using Qwen2.5-3B [73] on GSM8K [9]. Some recent works [15, 20, 21, 91] reduce this inefficiency with asynchronous training, but stale data can degrade training quality [15, 32, 45, 83]. Advanced synchronous frameworks [27, 58, 59] instead optimize model placement, but still face a trade-off: separating components causes under-utilization, while co-locating them can introduce out-of-memory (OOM), context switching latency, or resource interference [58, 59, 87].

2) Across training steps: Dynamic response lengths make static resource allocation inefficient. Prior work shows that generation lengths can change across training steps [13, 61, 64]. We verify this pattern in Fig. 3(b) using different RLHF algorithms and model combinations on GSM8K. For many reasoning models, response lengths increase as training proceeds [13, 64]. Because LLM generation is token-level and auto-regressive, longer responses directly increase inference time and create dynamic resource demands across training steps. Serverful platforms, with static allocation and long startup times, cannot adapt to these changes.

3) During each generation phase: Redundant KV cache calculation causes unnecessary costs. When an LLM generates a response, the prompt is first processed once during *prefilling* to build the KV cache, and *decoding* then reuses this cache [12, 67]. Many RLHF algorithms generate multiple responses per prompt [57, 78, 80, 85], so the same prefill work is often repeated. Prior LLM serving systems reduce this cost² by reusing KV cache for shared prefixes [31, 37, 62, 70]. Some RLHF systems such as VERL [59] compute the cache once and reuse it across batches in the same step, as shown in Fig. 4(a), but this serializes generation across batches. Parallel generation improves throughput, but it also repeats KV computation across workers, as shown in Fig. 4(b). Fig. 4(c) shows that about 22% of total active GPU×second is spent on redundant computation when running GRPO with Qwen2.5-3B on GSM8K and generating four responses per prompt.

4) Inside the actors: Varying response lengths lead to GPU under-utilization. Sampling actors can consume very different amounts of compute because response lengths vary widely. Within

²Cost here is calculated as active GPU×second, which is the sum of allocated GPU count multiplied by the lifetime of each active component.

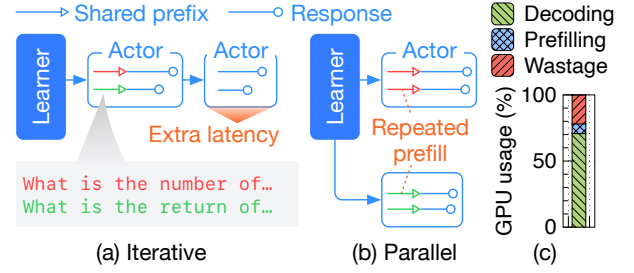


Figure 4: Existing generation strategies [27, 31, 59, 84] for sampling-heavy RLHF workloads, including (a) iterative generation, which introduces additional latency, and (b) parallel generation, which leads to (c) unnecessary KV recalculation.

one actor, some responses finish after only a few decoding steps, while others continue much longer, extending the generation phase even after most responses have completed [89]. This imbalance leaves some GPU resources idle while waiting for the longest responses to finish. Fig. 3(a) shows this pattern using Qwen2.5-3B with maximum response length 1024 on AIME2024 [39], GPQA [53], and LiveCodeBench [30].

2.3 Opportunities for Serverless RLHF

We propose the first RLHF system leveraging serverless computing for the rollout actors, by identifying the following opportunities to address the limitations mentioned above.

Serverless can release idle components on time to mitigate wastage between phases. Serverless’s elasticity and function-like pay-as-you-go execution can lower active GPU×second training cost by launching components only when needed and promptly releasing them when idle, as shown in prior serverless RL systems [75–77]—motivating a serverless RLHF design.

Scaling actors to accommodate the changing resource demands across training steps. Serverless enables us to adapt actor counts to shifting workloads and search for “sweet spots” where extra compute shortens step time enough to reduce total cost; achieving this requires a cost model to guide scaling decisions, and is harder on serverful platforms due to long startup latency [3, 18, 43].

Deduplicated prefill in a serverless environment to mitigate repeated calculations during the generation phase. By grouping prompts with shared prefixes ahead of time, and exploiting algorithms (e.g., GRPO [57]) that sample multiple responses per prompt, we can compute prefill once and reuse it across responses in a parallel generation manner.

Grouping prompts with similar lengths in the same actor to address workload imbalance inside actors. Because actor cost depends on execution time, a few long responses can keep an actor alive while other resources sit idle; predicting response lengths and co-locating similar-length prompts lets some actors finish sooner and be released earlier.

3 Objectives and Challenges

We design xRLHF, the first RLHF system with serverless actors, to seize the opportunities and address the limitations. The final goal is to apply serverless execution and elasticity to accelerate

synchronous RLHF training and reduce the active GPU×second costs, without changing the training semantics of RLHF algorithms:

Speed up training. We aim to find the best actor configuration with lower cost and faster speed. We will dynamically scale actor counts to optimize the data parallelism to match changing workloads across training steps, thus accelerating training.

Reduce redundant computation. We aim to decouple the prefill and decode stages during the generation phase and use deduplicated assigned prompts to avoid repeated prefill computation during parallel generation.

Improve resource utilization. We aim to assign prompts more evenly, by grouping workloads with similar inference times in the same actor, to reduce idle time and GPU waste in each actor.

These objectives lead to three corresponding main challenges:

How to balance speed and cost? Using more actors can reduce generation time while possibly raising cost. The system needs to estimate both time and cost to find a good balance.

How to reduce the transmission overhead in deduplicated prefill? Deduplicated prefill decouples the prefill and decode, while introducing additional KV transmission between the prefill actor and the decode actors, which might slow down training execution.

How to handle prediction errors in prompt assignment? Even if the workloads inside each actor can be estimated, the prediction accuracy won't be able to reach 100% due to the inherent randomness in LLM computation. There might still be a few outliers assigned to the wrong actor, which requires a backoff strategy.

4 xRLHF's Design

4.1 Overview

xRLHF is the first RLHF training system running with serverless actors for generation. It leverages the elasticity and fine-grained resource management of serverless computing to facilitate RLHF training. xRLHF has two parts: a client side and a provider side. The client side lets users define and submit RLHF workloads. The provider side runs the submitted workload with an elastic configuration optimized for the available GPU cluster. The provider side is assumed to have access to the GPU cluster topology. Fig. 5 illustrates xRLHF's workflow. Before each RLHF step, xRLHF constructs a rollout execution plan. xRLHF then predicts response lengths, evaluates candidate decode actor numbers, assigns prompts to actors, and chooses actor placement using profiled compute and communication costs. During execution, the prefill actor computes reusable prompt KV cache, decode actors generate responses, completed actors are released after keep-alive expiration, and the learner consumes the completed on-policy batch in the original prompt order. The five main steps of xRLHF in each training step are listed below:

Step ①: Setting up the prefill actor. xRLHF analyzes the prompt dataset or algorithm-specific hyperparameters to identify a deduplicated batch of shared prefixes. These shared prompts are assigned to a dedicated prefill actor, which computes the initial KV cache to be reused by all decode actors during the decoding stage.

Step ②: Iterating over all available actor numbers. xRLHF employs a cost-aware planning module to estimate both execution time and total cost across a range of possible actor numbers, with the corresponding prompt assignment strategies.

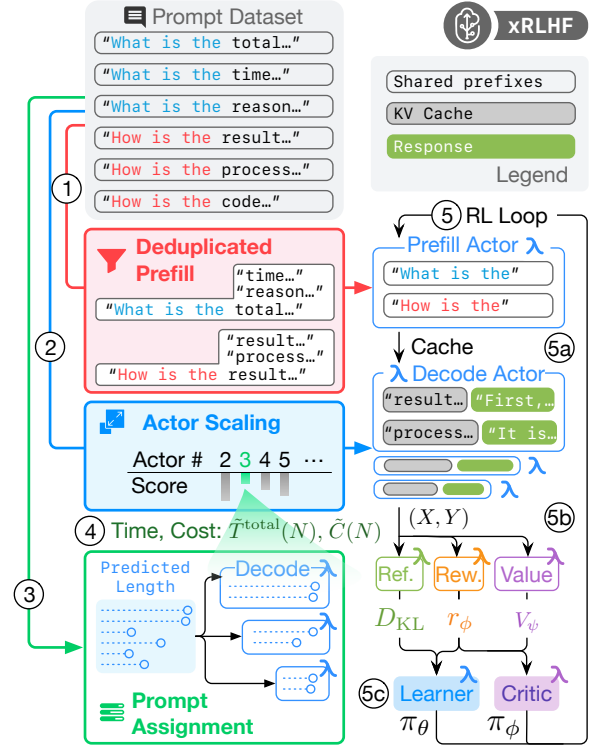


Figure 5: xRLHF's overview.

Step ③: Prompt assignment among decode actors. When calculating the scaling score with a given actor number and workload, xRLHF first generates a corresponding prompt assignment plan for better cost and time estimation. xRLHF utilizes historical training data to predict the response length of each prompt in the current training batch. It then sorts prompts by estimated length and evenly distributes them into the N' decode actors, grouping prompts with similar lengths to minimize resource wastage and improve actor efficiency.

Step ④: Scaling the decode actors. The prompt assignment module returns the workload assignment plan given the actor number and its corresponding execution time and cost estimation. Based on those results, xRLHF's actor scaling module selects the optimal number of actors N' and its corresponding workload assignment plan by balancing speedup with resource efficiency.

Step ⑤: Executing the RLHF loop. After generating the execution plan for the generation phase of the next training step, xRLHF initiates the step and executes the three RL phases, including 5a) generation, 5b) preparation, and 5c) learning, following the structure of conventional RLHF systems. The training process repeats the five steps until completion.

4.2 Deduplicated Prefill

RLHF algorithms [2, 26, 57, 78, 80] have duplicated KV cache computations in the generation phase of a training step due to multiple responses generated for a single prompt. As a result, the number of unique KV cache computations required in the prefill stage is often

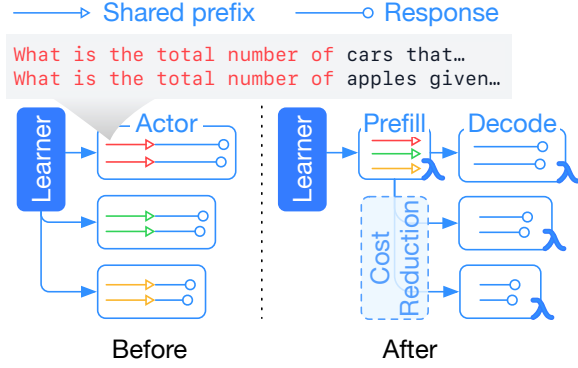


Figure 6: xRLHF's deduplicated prefill design.

less than the number of response generations performed in the decode stage. Existing serverful RLHF frameworks [27, 59] ignore this characteristic and allocate static resource sizes throughout training. In contrast, serverless computing offers an elastic execution model, allowing us to reduce cost by applying a deduplicated prefill-decode design. This enables allocating only the necessary resources for the prefill actor and computing each unique KV cache exactly once.

Deduplicated prefill computes the KV cache for each reusable prompt prefix once on a prefill actor. Decode actors then receive or reference the corresponding KV cache through vLLM's P2P NCCL connector and continue autoregressive decoding from that state. Since the prefill actor is compute-bound, like the learner [59, 87], we unify both components' resource allocation and model parallelism.

We first estimate its maximum prefill batch size B_{prefill} using a profiling-based capacity model. xRLHF then chooses a target shared-prefix length L so that the prefill actor is filled to capacity with deduplicated prompts. Concretely, let X be the training prompt set and define a candidate prefix length $L \in (0, L_{\min}]$, where

$$L_{\min} = \min_{x \in X} \|x\|.$$

For a candidate prefix length L , let $D(L)$ be the number of unique prefixes of length L in X (i.e., the count of deduplicated prompts at prefix length L). xRLHF selects the largest L such that

$$D(L) \leq B_{\text{prefill}}.$$

For fixed prefixes, $D(L)$ is non-decreasing with L : two prompts that differ at length L cannot become identical at a longer prefix length. xRLHF therefore selects the largest usable prefix length whose unique-prefix count still fits the prefill capacity, bounded by the shortest usable prompt length after padding removal.

To mitigate the possible communication overhead introduced by additional KV transfer, xRLHF enables deduplicated prefill only when it reduces prefill cost without increasing the step latency:

$$\begin{aligned} C_{w/\text{dup prefill}} &< C_{w/o \text{ dup prefill}}, \\ T_{\text{Step}}^{w/\text{dup prefill}} &\leq T_{\text{Step}}^{w/o \text{ dup prefill}}, \end{aligned}$$

where C represents estimated active GPU×second cost, and T means estimated execution time.

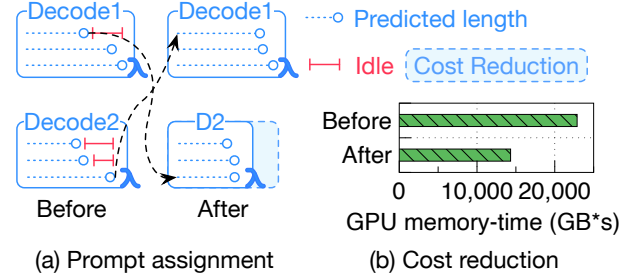


Figure 7: (a) xRLHF's prompt assignment design, and (b) the potential benefits that can be achieved under GRPO training with the Qwen2.5-3B model [73] and the GSM8K dataset [9].

4.3 Prompt Assignment

Due to the varying lengths of responses generated within a single sampling actor, GPU utilization decreases when most sequences finish early and only a few long responses remain. Previous work [89] proposes a cut-and-migrate strategy to interrupt sequence generation and migrate unfinished responses to another actor. However, this method adds transmission overhead and interrupts all actors simultaneously, harvesting resources only at coarse granularity.

To enable efficient cut-and-migrate in serverless actors, we propose a dynamic ranking-based prompt assignment strategy, shown in Fig. 7(a), which reduces transmission cost by minimizing the number of cut responses. This strategy allows actors to finish at different times while keeping the workload balanced within each actor. As shown in Fig. 7(b), grouping prompts with similar predicted response lengths into the same actor enables earlier actor termination, thereby reducing cost. To support this design, we require a response length predictor to estimate workloads.

Response length estimation. Accurately estimating response generation workloads in advance is important for cost-efficient prompt assignment. Recent works propose response length prediction methods for LLM serving [17, 49, 86], but they usually require either fine-tuning the LLM for self-prediction [86] or training a proxy model [17, 49, 88], which adds extra cost and complexity.

A key advantage in RLHF is that we have access to both the training dataset and historical response lengths from earlier training steps. Moreover, RLHF includes a reference model to constrain each training step's update, which creates an opportunity to rely on recent length results for estimation. To validate this, we track the response length changes for each prompt throughout Proximal Policy Optimization (PPO) training. Most response lengths remain relatively stable over time: approximately 70% vary by no more than 50 tokens across two consecutive epochs, and around 90% change by no more than 100 tokens per epoch. This suggests that historical response lengths from previous training steps can serve as a reliable estimator for the current step, avoiding costly model-based predictors [17, 49, 86]. For the first epoch without historical lengths, xRLHF uses ground-truth answer lengths or a pretrained length predictor [17, 49] for temporary estimates. After generated batches become available, xRLHF updates the estimator from observed response lengths. We adopt the common exponentially weighted moving average (EWMA) algorithm to take historical data as input and output the length estimation for each prompt.

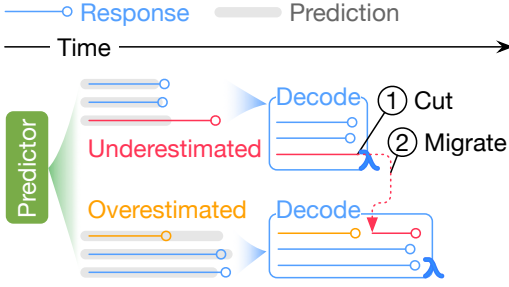


Figure 8: xRLHF’s cut-and-migrate strategy. xRLHF cuts unfinished, underestimated responses and migrates them to available decode actors with open slots, which have been released by completed, overestimated responses.

Cut-and-migrate in xRLHF. Even with reasonably accurate response length estimation, misassignments can still occur. To mitigate such errors, we apply a cut-and-migrate strategy inspired by prior work [89], shown in Fig. 8. During execution, if an actor is delayed by long-running prompts and other running actors have available slots, we terminate those unfinished generations and migrate them to the available actors. This reduces tail latency and improves overall GPU utilization. The timing of termination is controlled by a cutting threshold $\tau \in (0, 1]$. An actor is terminated when $\tau \times 100\%$ of its assigned responses have completed generation. This threshold balances the trade-off between waiting for stragglers and minimizing resource wastage. Cut-and-migrate does not change the RLHF sample distribution because migration occurs within the same synchronous generation step and before any learner update. xRLHF moves the generated token prefix, KV cache, and other required states together. The learner receives the completed samples from the current policy and restores them to the original prompt order before loss computation.

4.4 Cost-Aware Actor Scaling

Response lengths vary across RLHF training steps, so a fixed actor count can waste GPU resources on shorter responses or slow decoding on longer ones. This motivates adapting the actor count to each step’s generation workload. Because each sampling actor holds a policy replica, increasing the actor count increases data parallelism. For a fixed prompt batch, this added parallelism can shorten the memory-bound generation phase by distributing the batch across more actors [34, 59, 87, 89]. However, extra GPUs may raise training cost even when generation finishes sooner.

Prior serverless RL systems demonstrate elastic actor scaling [75, 76], while applying their scaling methods to RLHF raises two issues. First, existing methods either rely on a pretrained scheduler [76], or estimate data needs from the policy Hessian [75]—both incur additional costs and can be expensive for billion-parameter models. Second, actor scaling serves a different purpose in RLHF. In traditional RL, it changes training-data volume, whereas in RLHF it primarily changes the GPU resources allocated to a training step’s generation workload. These differences motivate a scaling policy that accounts for both generation time and GPU cost.

Balanced speedup with lower final costs. We estimate execution time and GPU consumption before changing the actor count to balance faster generation against its resource cost. The total cost for the N decode actors, denoted as C_{Decode} , can be estimated as

$$C_{\text{Decode}} := \sum_{i=1}^N T(X_i, G_i) \times G_i,$$

where X_i represents the set of prompts assigned to the i th decode actor, and G_i denotes the number of GPUs allocated to it. $T(X_i, G_i)$ is the estimated execution time for processing workload X_i using G_i GPUs, defined as

$$T(X_i, G_i) := T_{\text{Decode}}(X_i, G_i) + T_{\text{Sync}}(i) + T_{\text{KV}}(i),$$

which includes the weight synchronization, KV transfer, and decode time. To estimate the execution time $T(X_i, G_i)$, we leverage profiled data collected using dummy batches prior to training. Specifically, we measure the time-per-output-token (TPOT) latencies in the decode step, under different maximum response lengths, following the approach used in prior work [7, 87]. For each prompt $x \in X_i$, the corresponding sequence length is estimated using the response length prediction module described in §4.3.

The end-to-end execution time for N decode actors, denoted as $T^{\text{total}}(N)$, is determined by the slowest (i.e., longest-running) actor and is computed as

$$T^{\text{total}}(N) := \max_{i \in [1, N]} T(X_i, G_i).$$

For all candidate actor counts $N \in [N_{\min}, N_{\max}]$, where $N_{\min}$ and $N_{\max}$ denote the predefined minimum and maximum numbers of decode actors, respectively, we apply min-max normalization to the estimated execution time $T^{\text{total}}(N)$ and decode cost $C_{\text{Decode}}(N)$ over all candidate N values to ensure fair comparison. To determine the optimal number of actors N' , we formulate the objective as a weighted sum of the normalized time and cost:

$$N' := \arg \min_{N \in [N_{\min}, N_{\max}]} \{ \lambda \cdot \tilde{T}^{\text{total}}(N) + (1 - \lambda) \cdot \tilde{C}(N) \}, \quad (1)$$

where $\lambda \in [0, 1]$ is a user-defined weight controlling the trade-off between execution time and cost. A larger λ prioritizes faster execution, while a smaller λ emphasizes lower cost. xRLHF replays the user-defined static actor count with the actual response lengths after each step. When the scaled actor execution plan leads to a worse scaling score (Equation 1) than this static replay, xRLHF falls back to the predefined static actor count in the next step.

4.5 Locality-Aware Actor Placement

Model synchronization is one of the main bottlenecks in RLHF training [59, 89]. It occurs between the learning phase of the previous step and the generation phase of the current step, when model weights are transmitted from the learner to the sampling actors. To reduce both model synchronization latency and KV cache transfer latency, xRLHF adopts a locality-aware actor placement strategy to overlap communication with computation, reducing the execution time of each training step.

As discussed in §4.3, xRLHF groups prompts with similar expected workloads into the same decode actor. Under this design, the end-to-end generation time is determined by the actor with the longest workload. To reduce this critical-path latency, we first co-locate the

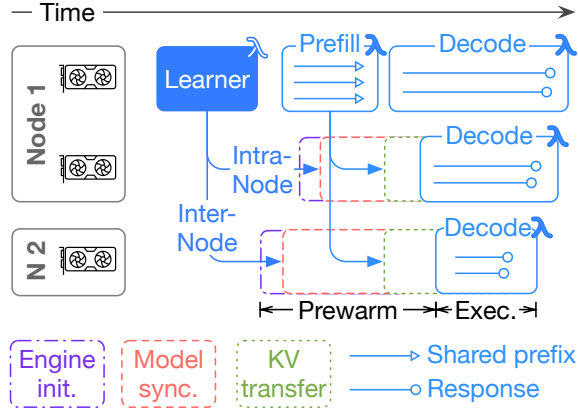


Figure 9: xRLHF’s locality-aware actor placement and pre-warming. Engine initialization and state transfer overlap computation to reduce startup and communication overhead.

prefill actor with the learner on the same GPU resources, which reduces model synchronization overhead. Next, we co-locate the decode actor with the longest workload with the prefill actor, so it can start without additional model-weight or KV-cache transfer latency. For the remaining decode actors introduced by actor scaling (§4.4), xRLHF places heavier workloads on GPU nodes closer to the learner and prefill actor (e.g., within the same node), and lighter workloads on more distant resources.

Since model synchronization starts after the learner update, while KV cache transfer starts after prefill completes, transmission can be overlapped with execution. As shown in Fig. 9, this overhead can be hidden as long as the following condition holds for all $i \in \{2, \dots, N\}$:

$$T_{\text{Sync}}(i) + T_{\text{KV}}(i) + T_{\text{Decode}}(X_i, G_i) \leq T_{\text{Prefill}} + T_{\text{Decode}}(X_1, G_1),$$

where N is the total number of decode actors, $T_{\text{Sync}}(i)$ and $T_{\text{KV}}(i)$ are the model synchronization and KV cache transfer latency for the i th decode actor, $T_{\text{Decode}}(X_i, G_i)$ is its execution time, $T_{\text{Decode}}(X_1, G_1)$ is the execution time of the decode actor co-located with the learner and prefill actor, and T_{Prefill} is the execution time of the prefill actor. We incorporate this constraint into actor scaling to avoid prolonged execution caused by large-scale data transfer. xRLHF targets a provider-hosted serverless cluster, where locality-aware placement can be supported.

4.6 Prewarming and Keep-Alive

To mitigate the cold-start latency in the serverless rollout actors, we design specialized prewarming and keep-alive strategies in xRLHF besides locality-aware placement. The prewarming and keep-alive scheduling for training step $t+1$ happens after the generation phase of step t , in parallel with the preparation and learning phases at step t . The prewarming and keep-alive periods are included in the component lifetime and the active GPU×second cost.

Prewarming with invocation prediction. We predict when rollout actors will be needed using the learning phase latencies observed in earlier steps, following prior work [48, 56]. Let $\hat{T}_{\text{Learn}}$ be the predicted duration of the current learning phase, $\hat{T}_{\text{Init}}$ be the

time needed to launch an actor and initialize its inference engine, and γ be a safety margin. Measured from the start of learning, xRLHF begins prewarming at

$$T_{\text{Prewarm}} = \max(0, \hat{T}_{\text{Learn}} - \hat{T}_{\text{Init}} - \gamma).$$

When $\hat{T}_{\text{Learn}} \geq \hat{T}_{\text{Init}} + \gamma$, actor initialization is expected to finish before learning ends. Otherwise, prewarming starts immediately, and part of the initialization time may remain on the critical path. Model-weight synchronization starts after the learner update, while KV-cache transfer starts after prefill.

Cost-aware keep-alive. Similar to common serverless systems [5], xRLHF may keep completed decode actors active for reuse. Actors not selected by the actor-scaling module’s next-step plan are released immediately. For a selected actor, keeping it alive is cheaper than releasing and recreating it when

$$\hat{T}_{\text{Prep}} + \hat{T}_{\text{Learn}} + \hat{T}_{\text{Prefill}} \leq \hat{T}_{\text{Init}} + \gamma.$$

Otherwise, xRLHF releases the actor and recreates it through prewarming. Model synchronization and KV-cache transfer are required in both cases and therefore do not affect the decision.

5 Implementation

We build xRLHF on top of VERL [59], with vLLM [31] as the inference engine and Fully Sharded Data Parallel (FSDP) [82] as the training engine. xRLHF does not change VERL’s RLHF training logic. It adds an elastic execution layer that controls how rollout actors are launched, placed, and released during the generation phase.

Client and provider. xRLHF separates workload submission from workload execution. The clients define the RLHF workload and submit it to the system. On the provider side, xRLHF schedules and runs the workload with an elastic configuration for the available GPU pool. We assume the provider-side scheduler can access GPU cluster topology, so it can choose topology-aware placements.

Serverless cluster. Due to the limited GPU availability in commercial serverless platforms [4, 42], we build our own serverless cluster using Docker [41], a common approach in existing serverless environments, following prior work [5, 8]. The Ray framework [44] manages compute resources for the serverless actor functions and coordinates communication among them.

Serverless functions. xRLHF adopts hybrid execution with a persistent learner and serverless actors. The learner can run either as a long-running serverless function or on conventional resources, as it stores the necessary training states and remains active during training with both learning and longest rollout workloads on it. The rollout actors in xRLHF are implemented in a stateless manner, with dedicated resources for concurrent functions, and are launched or reused only when needed for generation. The prewarming and keep-alive strategy follows §4.6.

State management. Although rollout actors are function-like, RLHF training itself is not stateless. xRLHF keeps persistent training state on the learner side and moves only the state needed for rollout execution. Model weights and optimizer state remain with the learner. Before generation, the updated policy weights are synchronized to the prefill and decode actors. The KV cache is stored in vLLM’s GPU memory during generation and is transferred through NVIDIA Collective Communications Library (NCCL) when reused

by decode actors. Rollout samples are returned to the learner as VERL data objects. Prompt metadata, historical response lengths, actor assignments, and scheduling statistics are stored in Redis [52]. During cut-and-migrate, partial generations, generated tokens, KV cache, and sampling state move together with the migrated request. **Model configuration.** By default, xRLHF allocates the same number of GPUs to each decode actor. For internal model-parallel settings, such as tensor parallelism and pipeline parallelism, xRLHF follows VERL and vLLM configurations and tunes the batch size and KV cache allocation to fit the available GPUs. Thus, xRLHF does not replace the parallelism strategies inside VERL or vLLM; it optimizes the outer execution layer with better scheduling, resource allocation, and placement.

Workload and transfer profiling. xRLHF needs estimates of inference throughput and data transmission cost for actor scaling and locality-aware placement. During initialization, xRLHF runs dummy batches to profile decode speed, following prior profiling-based methods [1, 65, 87]. It also profiles GPU bandwidth and topology on the hardware testbed. These profiled results are combined with predicted response lengths to estimate each actor’s execution time and GPU cost. Model synchronization latency is measured from earlier synchronization steps and reused because it remains stable for a fixed model size, actor placement, and cluster topology.

6 Evaluation

6.1 Experimental Setup

Physical testbed. We use two AWS EC2 g6e.48xlarge instances as the main testbed. Each has 192 AMD EPYC 7R13 vCPUs, 1536 GB of system memory, and eight NVIDIA L40S GPUs with 48 GB of memory per GPU. The testbed provides 16 GPUs with 768 GB of total GPU memory. The two nodes form a single Ray-managed GPU pool, allowing xRLHF to launch, reuse, and release rollout actors.

Large-scale simulation. We complement the physical experiments with generation-phase simulations using Vidur [1] and VERL workload traces. The simulations use larger Llama models [19, 66] on clusters with up to 20 nodes and 160 NVIDIA H100 GPUs.

Workloads and datasets. We evaluate xRLHF with PPO [85], a widely used synchronous RLHF algorithm, and GRPO [57], which removes the critic model and generates multiple responses per prompt. GRPO is also the basis of several recent reasoning-oriented RLHF methods [78, 80]. We use three responses per prompt for GRPO. We use three datasets with different workload characteristics: GSM8K [9] for mathematical reasoning, GPQA [53] for scientific question answering, and LiveCodeBench [30] for code generation. Their different response-length distributions test xRLHF under both short and long generation workloads. Unless stated otherwise, physical experiments use Qwen2.5 models [73] from 3B to 7B, with maximum prompt and response lengths of 1,024 and 2,048 tokens. We run each configuration for three epochs and average results across all training steps, except where noted.

Software environment. The testbed runs Ubuntu 22.04. We use Docker to package the RLHF components, Ray to manage resources, VERL [59] as the underlying RLHF framework, vLLM as the inference engine, and FSDP as the training engine. The experiments use CUDA 12.8, NCCL 2.27, VERL 0.3.1.dev, and vLLM 0.8.3.

Baselines and comparisons. VERL [59] is our primary baseline because xRLHF is implemented on top of it. For a fair comparison, we run VERL with serverless actors during generation, releasing finished and unused actors to isolate the benefits of xRLHF’s core features. The VERL baseline uses a fixed actor count and the same RLHF configurations as xRLHF. We choose three actors for the physical baseline as profiling shows that this setting provides the best static speed–cost trade-off on our testbed. For the ablation study, we use a naive xRLHF variant that keeps the serverless rollout layer but disables its three core optimizations: DP (Deduplicated Prefill), AS (Actor Scaling), and PA (Prompt Assignment). This comparison separates the benefit of serverless function execution from the benefit of the workload-aware optimizations. We also compare prompt assignment with RLHFuse [89] in §6.5. In §6.6, we implement a model-based length predictor following prior work [17, 49, 88].

Performance metrics. We measure both wall-clock step time and active GPU×second cost. Step time covers the complete synchronous RLHF step, including generation, preparation, and learning, together with any exposed scheduling or communication overhead. For each component, active GPU cost is calculated as $C_{\text{GPU}} = \sum_j G_j T_j$, where G_j is the number of GPUs allocated to component j , and T_j is the time from GPU allocation until the resources are released. For rollout actors, this lifetime includes inference-engine initialization, prewarming, model synchronization, KV-cache transfer, response generation, migration, and keep-alive time when applicable. For the learner, it includes the preparation and learning phases. Using active GPU×second provides a provider-independent comparison of the resources used by xRLHF and the baselines, following prior work [54, 77].

xRLHF configuration. Each rollout actor uses two GPUs for the 3B model and four GPUs for the 7B model. All decode actors in the same experiment use the same GPU count and internal model-parallel configuration. For both xRLHF and VERL, we follow the training configurations provided by VERL [59] and adjust only hardware-dependent parameters, such as batch size and the vLLM KV-cache allocation ratio, to avoid out-of-memory errors. The invocation prediction buffer γ for prewarming is set to 5% of the estimated learner execution time. For xRLHF, the response-length estimator uses an EWMA window size of 1, the cut-and-migrate threshold is set to $\tau = 0.7$, and the actor-scaling objective uses $\lambda = 0.7$. We analyze these settings and their sensitivity in §6.6. The actor-scaling search uses the same minimum and maximum actor counts across comparable workloads and includes the predefined static actor count as the fallback configuration.

6.2 Overall Performance

We evaluate how xRLHF improves RLHF training efficiency in both speed and cost by comparing it with a naive version built on VERL [59]. This version keeps the serverless actors but disables all core optimizations, including DP: Deduplicated Prefill, AS: Actor Scaling, and PA: Prompt Assignment, for a fair comparison. Experimental results in Fig. 10, collected from five repeated runs per setting, show that xRLHF speeds up RLHF training and lowers cost by (1) dynamically scaling actors to match changing resource demands, (2) eliminating repeated computation, and (3) releasing idle resources promptly. Training on GSM8K takes the longest, due to

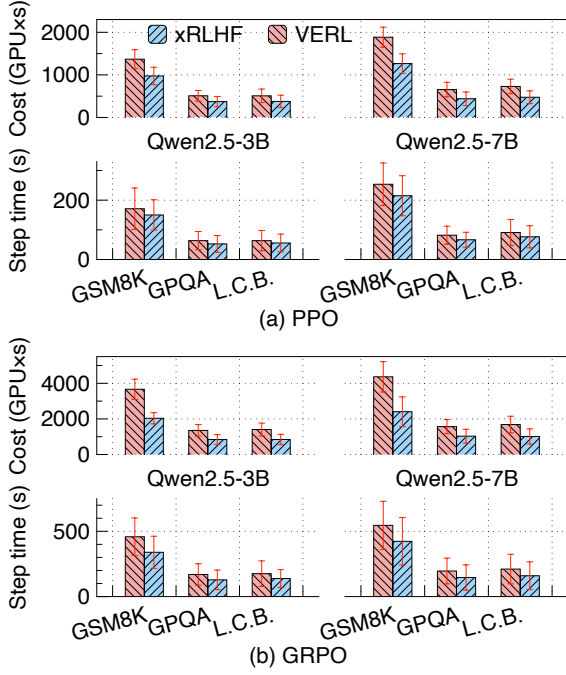


Figure 10: xRLHF’s overall performance across five runs.

Table 1: Final GRPO quality on GSM8K.

System	Final reward	Final KL	Final entropy
VERL	0.87 ± 0.014	0.00024 ± 0.00012	0.0498 ± 0.007
xRLHF	0.87 ± 0.016	0.00025 ± 0.00016	0.0495 ± 0.006

many long responses, which is consistent with the profiled results in Fig. 3(a). xRLHF delivers larger gains on GRPO than on PPO because GRPO is more sampling-heavy, leaving more room to reduce waste during generation. We also observe that as training scale and workload grow, the benefits of xRLHF increase, suggesting its potential to adapt and deliver greater gains in super-large RLHF workloads. Overall, xRLHF speeds up training by up to 1.35 $\times$ and reduces cost by up to 44.8%.

6.3 Training Correctness and Final Quality

xRLHF preserves training quality because it does not change the RLHF algorithm. It only changes how rollout generation is scheduled and adjusts the resources allocated for generation. The learner still uses the same policy version, prompt batch, reward computation, KL penalty, and synchronous update rule as VERL. After generation, xRLHF restores the completed rollout samples to the original batch order before the learner update, so the learner receives the same type of on-policy data as in VERL. As shown in Table 1, xRLHF achieves similar final reward, KL penalty, and entropy to VERL. The small differences are within the standard deviation across runs, which is expected because RLHF training and LLM generation are stochastic and can have minor GPU-level nondeterminism.

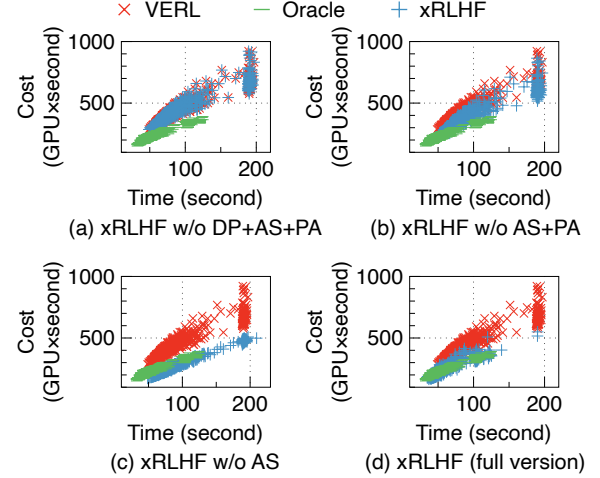


Figure 11: xRLHF’s ablation study (DP: Deduplicated Prefill; AS: Actor Scaling; PA: Prompt Assignment). Each point shows the cost and execution time of the generation phase in one training step. Points closer to the lower-left corner indicate lower cost and faster execution. Subfigure (a) shows xRLHF with all features disabled. We enable one additional feature at a time, illustrating the incremental benefits of each design.

6.4 Ablation Study

We evaluate the effectiveness of xRLHF’s core features—deduplicated prefill, prompt assignment, and actor scaling—via an ablation study using several system variants. We run experiments using the GRPO algorithm and the Qwen2.5-3B model on the GSM8K dataset. The study considers four variants of xRLHF: 1) xRLHF w/o DP+AS+PA: raw xRLHF based on VERL, with no deduplicated prefill, no prompt assignment, and no actor scaling; 2) xRLHF w/o AS+PA: xRLHF with only deduplicated prefill enabled and without prompt assignment or actor scaling; 3) xRLHF w/o AS: xRLHF with no actor scaling; and 4) xRLHF (full version): xRLHF with all three core features enabled. We further compare these variants against a VERL baseline and an oracle baseline, where the oracle simulates optimal scheduling with 100% accurate length estimates. We visualize generation cost and execution time in Fig. 11. Each point represents one training step; more points closer to the lower-left indicate better performance with overall lower cost and shorter time.

Fig. 11(b) shows the cost reduction from deduplicated prefill relative to the raw version in Fig. 11(a). While the reduction is modest in this setup, its magnitude is dataset- and algorithm-dependent and can be larger under different conditions. Fig. 11(c) demonstrates that prompt assignment with cut-and-migrate further reduces GPU cost by grouping prompts of similar length within the same actor. In rare cases, if cut-and-migrate is triggered late, migrations can introduce extra latency (e.g., the rightmost outlier). Overall, transmission overhead is typically overlapped with computation and does not increase end-to-end latency. Fig. 11(d) illustrates the speedup from actor scaling, which dynamically adjusts the number of actors to match workload demands. When additional resources are allocated for heavier steps, the shorter execution time offsets the added resources, so the total cost grows slowly or remains comparable while achieving higher throughput.

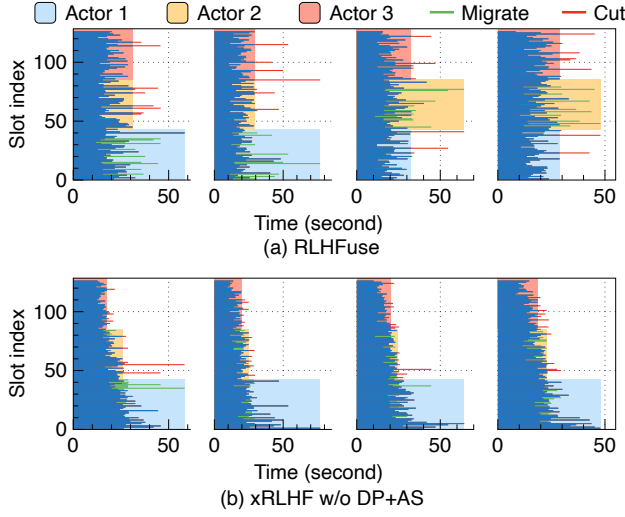


Figure 12: Effectiveness of xRLHF's prompt assignment. Prompt assignment reduces cost by grouping prompts with similar predicted response lengths into the same decode actor. (DP: Deduplicated Prefill; AS: Actor Scaling; PA: Prompt Assignment.)

6.5 Effectiveness of xRLHF

To better isolate the benefit of each core feature, we run controlled experiments that enable one feature at a time, using the same dataset and algorithm settings as in §6.4.

Effectiveness of prompt assignment. Fig. 12 shows the per-actor workloads and execution times of xRLHF compared with RLHFuse [89]. Since it is difficult to reproduce identical generated content in RLHF due to hardware-level precision loss [23, 79], we run xRLHF with only prompt assignment enabled using PPO on GSM8K, record the execution traces, and use them to simulate the raw version with all features disabled. We report results randomly selected from four consecutive training steps. In Fig. 12(a), RLHFuse incurs higher GPU time because its global cut-and-migrate strategy lacks prompt ranking, leading to uneven workload distribution across the three actors. By contrast, Fig. 12(b) shows that xRLHF groups prompts with similar expected lengths into the same actor, allowing shorter actors to finish earlier and be released sooner. As a result, prompt assignment reduces generation cost by 12.8% compared with RLHFuse. The migration overhead in xRLHF is negligible, compared to RLHFuse, because only a small number of outlier prompts need to be cut and migrated.

Effectiveness of actor scaling. Using the same setup as §6.4, we compare the raw xRLHF with the same system with only actor scaling enabled. Fig. 13(a) shows that actor scaling adapts the number of actors across training steps. In many cases, xRLHF finds a sweet spot where shorter step time offsets the added resources, reducing or maintaining total cost compared with a fixed actor count. This shows that xRLHF can estimate the next-step workload from historical results and select an effective actor count accordingly.

Effectiveness of deduplicated prefill. Using the same setup as §6.4, we compare the raw xRLHF with xRLHF with only deduplicated prefill enabled. For each step, we measure the total prefill tokens and the cost of the prefill stage during RLHF generation. Fig. 13(b)

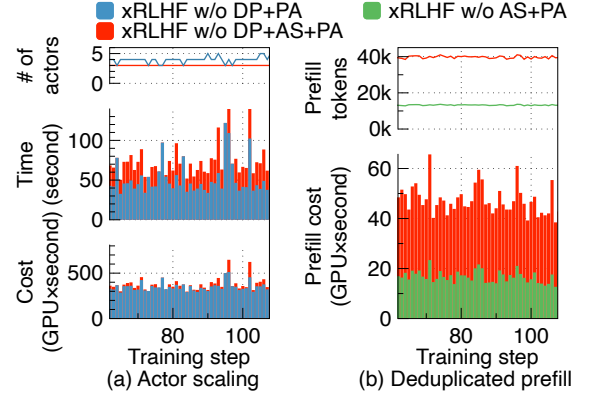


Figure 13: Effectiveness of xRLHF's (a) actor scaling and (b) deduplicated prefill. Deduplicated prefill gives the largest benefit when many responses share the same prompt or prefixes. (DP: Deduplicated Prefill; AS: Actor Scaling; PA: Prompt Assignment.)

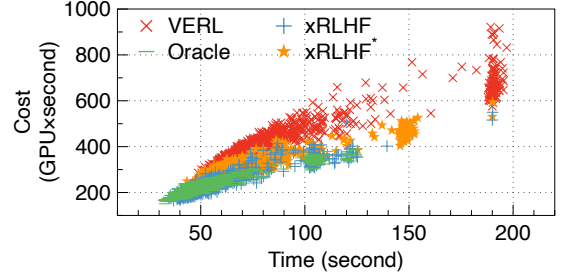


Figure 14: Sensitivity analysis of xRLHF's length prediction. xRLHF does not require exact response-length prediction. xRLHF* replaces the historical length estimator with a weaker model-based predictor, and still achieves 1.22 $\times$ speedup and 28.9% active GPU $\times$ second reduction over VERL.

shows that deduplicated prefill mitigates repeated computation. The amount of savings depends on the RLHF configuration.

6.6 Sensitivity Analysis

To evaluate xRLHF under different length-prediction accuracies and justify our EWMA window size, cut-and-migrate threshold τ , and actor-scaling weight λ , we repeat the setup in §6.4 and report each setting's mean per-step cost and time in Figs. 14 and 15.

Sensitivity to response length prediction accuracy. In xRLHF, both actor scaling and prompt assignment rely on response length estimates. To test robustness, we replace our historical training-step-based estimator with a lower-accuracy model-based predictor from prior work [17, 49], denoted as xRLHF*. Following prior response-length prediction methods [17, 49, 88], this predictor is attached to the policy model, requires pretraining on the target RLHF dataset, takes about 11 GPU-hours to train, and achieves about 44% accuracy during RLHF. Even with this lower accuracy, Fig. 14 shows that xRLHF* still achieves 1.22 $\times$ speedup and 28.9% cost reduction over the VERL baseline. This shows that xRLHF tolerates imperfect estimates because prompt assignment depends mainly on

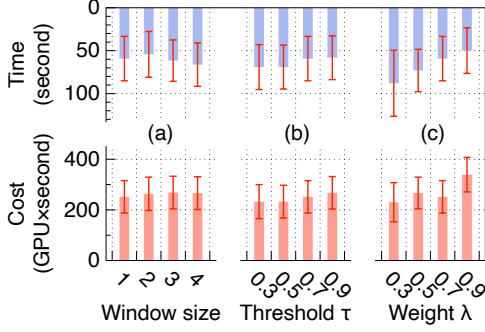


Figure 15: xRLHF hyperparameter sensitivity: (a) EWMA window size for length prediction, (b) cut-and-migrate threshold τ , and (c) actor scaling weight λ .

the ranking of expected lengths, cut-and-migrate corrects harmful mis-assignments at runtime, and actor scaling uses profiled TPOT curves that reduce the impact of individual prediction errors.

Sensitivity to EWMA window size. We run fifteen epochs and vary the EWMA window from 1 to 4. Fig. 15(a) shows only marginal gains from larger windows. Since training is short, response-length trends are often monotonic. We therefore use a window size of 1.

Sensitivity to cut-and-migrate threshold τ . We run three epochs and vary $\tau \in [0.3, 0.9]$. As shown in Fig. 15(b), smaller τ triggers earlier cuts and can reduce cost, but it also increases per-step time due to more migrations and state transfers. Moreover, cost does not decrease further for $\tau \leq 0.5$, because early cuts still wait for slots on other actors. We therefore set $\tau = 0.7$ to balance time and cost.

Sensitivity to actor-scaling weight λ . We run three epochs and vary $\lambda \in [0.3, 0.9]$, where a larger λ favors speed and a smaller one favors cost. Fig. 15(c) shows that $\lambda = 0.7$ gives the best trade-off in our experiments, achieving the second-lowest cost and second-shortest step time. This is because the min-max normalization in the objective does not fully preserve the original distribution of cost and time. Although the best value may vary across workloads, the trend is consistent, so we use $\lambda = 0.7$ as a practical default.

6.7 Scalability Analysis

To evaluate xRLHF at a larger scale under a limited budget, we run an extra large-scale physical experiment with 4×AWS EC2 g7.48xlarge nodes running GRPO with Qwen3.5-27B on GSM8K. We additionally implement a simulator for PPO generation on GSM8K using Vidur [1], a widely used LLM serving simulator, together with RLHF training traces from VERL [59]. We simulate a cluster with up to 20 nodes, each equipped with 8 NVIDIA H100 GPUs connected via NVLink. We use the Llama2-70B model with a generation batch size of 512 and scale the number of nodes.

Fig. 16(a) shows that per-step generation time decreases as more resources are added under a fixed workload in the simulated cluster, indicating strong scaling. Meanwhile, the scheduling overhead of xRLHF remains stable and within 30 ms across all configurations. We report the average performance of xRLHF in the four-node physical experiment and the simulated cluster, with relative results compared to the VERL baseline, in Fig. 16(b) and (c), respectively. In the physical experiment, xRLHF achieves 1.26× speedup and reduces

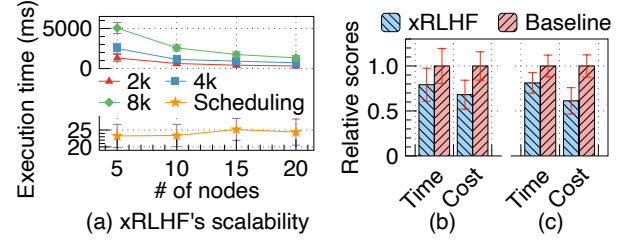


Figure 16: xRLHF's scalability analysis. (a) Execution time for response generation with lengths from 2k to 8k tokens, along with scheduling overhead, under different cluster sizes. We also report average normalized performance from (b) four physical nodes and (c) large-scale simulation experiments.

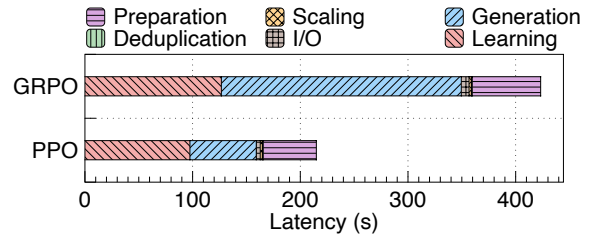


Figure 17: xRLHF's latency breakdown for PPO and GRPO, averaged across training steps.

cost by 31.9% compared with the VERL baseline. In the simulated cluster, xRLHF achieves 1.23× speedup and reduces cost by 38.7%.

Larger models can increase both transfer overhead and long-tail decoding time. We present an illustrative example using four actors running FP8 Kimi K2 [63] with 1T total parameters, each spanning two 8×H200 nodes. Extrapolating processing speed from a vLLM benchmark³ to assumed maximal output lengths of 8,192 tokens and 2,048 on remote actors gives decoding times of 369 and 92 seconds. Three policy copies plus a conservative 144-GB prompt-KV allowance per remote actor take 3.52 TB. At an assumed aggregate bandwidth of 25 GB/s, which is usually much higher with RDMA or InfiniBand, transfer takes 141 seconds. Even placing all transfers after prefill and before remote decoding, locality-aware placement can still hide communication in this 1T model setup.

6.8 Latency and Cost Breakdown

We evaluate xRLHF's latency and cost breakdown by running PPO and GRPO, two representative RLHF algorithms, on GSM8K.

Latency breakdown. Fig. 17 shows the latency breakdown across RLHF training steps. The overhead of xRLHF's core features, including prompt deduplication, response length prediction, and cost/time estimation, is negligible relative to measured step time because these components are model-free and lightweight. The I/O latency covers model synchronization, KV transfer, and workload migration. Model synchronization starts after the learner update, while KV transfer starts after prefill. Their exposed latency is negligible

³<https://docs.vllm.ai/projects/recipes/en/latest/moonshotai/Kimi-K2.html#fp8-benchmark-on-16xh200>

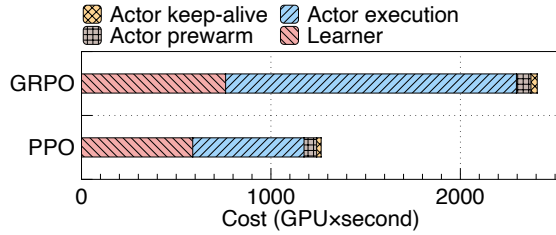


Figure 18: xRLHF’s active GPU×second cost breakdown for the learner and rollout actors.

in the end-to-end breakdown. This result reflects xRLHF’s locality-aware design, which places heavy computation near the centralized learner and overlaps communication with computation.

Cost breakdown. Fig. 18 reports the measured cost in active GPU×second for the centralized learner and elastic actors. The learner runs the preparation and learning phases, while actors produce responses for the generation phase. Each actor’s cost covers its lifetime from GPU allocation until release to the Ray GPU pool, including initialization, prewarming, model synchronization, KV transfer, generation, migration, and keep-alive time when applicable. Prewarming and keep-alive costs are counted but remain negligible relative to active computation because learning-phase durations are relatively stable across training steps and xRLHF times prewarming using learning-phase latencies observed in earlier steps.

7 Related Work

RLHF algorithms. Commonly used RLHF algorithms include PPO [55, 85], GRPO [57], ReMax [33], REINFORCE [2], and REINFORCE++ [26]. Non-RL alignment methods such as Direct Preference Optimization (DPO) [50] remove the RL loop by reformulating preference learning as supervised optimization, but recent work shows that such methods can underperform RL-based methods on complex reasoning and coding tasks [72]. GRPO-based methods [57, 78, 80] deprecate the critic model in the RL loop, and are commonly used today. But such methods are usually sampling-heavy due to multiple responses for each prompt. We focus on these workload characteristics and optimize sampling efficiency.

RLHF frameworks. Traditional RL frameworks [24, 28, 35, 90] target smaller models and do not scale well to LLM alignment due to model size and communication overhead. Recent LLM alignment frameworks [10, 15, 20, 27, 32, 58, 59, 74, 89, 91] combine high-throughput inference engines [11, 31, 84] with distributed training backends [51, 60, 82]. Some systems improve efficiency through asynchronous off-policy execution [15, 20, 32, 91], but bounded data staleness cannot guarantee training quality [45, 83]. Other systems optimize synchronous training through model placement and parallelism [27, 58, 59, 74]. However, these frameworks rely on serverful infrastructure and cannot adjust resource allocation at fine granularity across training steps. RLHFuse [89] offloads long-tail responses to dedicated GPUs, but its global cut-and-migrate strategy can introduce extra overhead without workload-aware prompt assignment. StreamRL [88] uses an LLM-based length predictor, which adds training overhead and needs to be updated as model behavior changes during RLHF. Multi-turn RL systems [38, 69] train LLM agents through repeated interactions with environments or

tools, and are outside the scope of this paper. xRLHF targets optimizing synchronous and single-turn RLHF with a model-free scheduler to dynamically launch and release serverless rollout actors.

KV cache reuse and prefill optimization. Prior LLM serving systems reduce prefill overhead by managing or reusing KV cache [31, 37, 62, 70, 84, 87], which mainly target online inference to improve serving throughput, memory efficiency, or latency under concurrent requests. xRLHF targets a different setting: synchronous RLHF rollout generation, where prompts come from a training batch, model weights change every step, and multiple responses are often sampled for the same prompt. In this setting, xRLHF uses deduplicated prefill to reduce repeated prefill computation and GPU×second cost while keeping parallel decode actors for rollout generation.

Serverless computing for LLM. Serverless computing has recently been studied for LLM serving [16, 29, 36]. ServerlessLLM [16] and ENOVA [29] improve serving efficiency through elastic deployment and auto-scaling, but they are designed for inference rather than RLHF training. ProRL Agent [81] improves modularity by decoupling rollout from training through a persistent service interface, whereas xRLHF improves efficiency by making RLHF rollout itself serverless and elastically provisioned.

8 Conclusion

This paper presents xRLHF, the first serverless framework for RLHF training. By targeting key inefficiencies in serverful RLHF systems—including idle time across dependent training phases, dynamic response lengths, and redundant KV cache computation—xRLHF leverages serverless execution for fine-grained resource adaptation. Through deduplicated prefill, prompt assignment, and cost-aware actor scaling, xRLHF reduces resource wastage both across the RLHF loop and within individual components. xRLHF also adopts locality-aware actor placement to overlap communication with computation, and uses specialized prewarming and keep-alive policies to keep actors ready in time without increasing cost. Experiments show that xRLHF achieves 1.14–1.35× speedup and 25.81–44.8% cost reduction on the physical testbed, and an average 1.23× speedup with 38.7% cost reduction on a simulated large-scale GPU cluster, demonstrating the effectiveness of serverless RLHF training.

Acknowledgments

The work of Rui Wei, Hanfei Yu, and Hao Wang was supported in part by NSF 2534241 and 2523997. The work of Devesh Tiwari was supported in part by Northeastern University. The work of Jian Li was supported in part by NSF 2337914. The work of Seung-Jong Park was supported in part by NSF 2403248 and 2403399. This research was supported by the National Artificial Intelligence Research Resource (NAIRR) Pilot allocation NAIRR240269. This work used Jetstream2 [22] at Indiana University through allocation CIS250978 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program [6], which is supported by National Science Foundation grants 2138259, 2138286, 2138307, 2137603, and 2138296. We thank Shubham Jain and Yoganjan Sivakumar for contributing to an earlier version of this paper while interning at IntelliSys Lab as master’s students at Stevens.

References

- [1] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav Gulavani, Ramachandran Ramjee, and Alexey Tumanov. 2024. Vidur: A Large-Scale Simulation Framework for LLM Inference. arXiv:2405.05465 [cs.LG] <https://doi.org/10.48550/arXiv.2405.05465>
- [2] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs. arXiv:2402.14740 [cs.LG] <https://doi.org/10.48550/arXiv.2402.14740>
- [3] Amazon Web Services. 2025. AWS Elastic Compute Cloud. <https://aws.amazon.com/ec2/>.
- [4] Amazon Web Services. 2025. AWS Lambda. <https://aws.amazon.com/lambda/>.
- [5] Apache. 2018. Apache OpenWhisk Official Website. <https://openwhisk.apache.org>.
- [6] Timothy J. Boerner, Stephen Deems, Thomas R. Furlani, Shelley L. Knuth, and John Towns. 2023. ACCESS: Advancing Innovation: NSF's Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support. In *Practice and Experience in Advanced Research Computing (PEARC '23)*. Association for Computing Machinery, New York, NY, USA, 173–176. doi:10.1145/3569951.3597559
- [7] Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdous, Aditya Tanikanti, Ken Raffanetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. 2024. LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators. arXiv:2411.00136 [cs.LG] <https://doi.org/10.48550/arXiv.2411.00136>
- [8] Cloud Native Computing Foundation. 2018. Knative. <https://knative.dev/docs/>.
- [9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. arXiv preprint arXiv:2110.14168 (2021). <https://doi.org/10.48550/arXiv.2110.14168>
- [10] Colossal-AI Corporation. 2023. Colossal-Chat: A Project to Implement LLM with RLHF. <https://github.com/hpcaitech/ColossalAI/tree/main/applications/ColossalChat>.
- [11] NVIDIA Corporation. 2023. TensorRT-LLM: A TensorRT Toolbox for Optimized Large Language Model Inference. <https://github.com/NVIDIA/TensorRT-LLM>.
- [12] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. arXiv:1901.02860 [cs.LG] <https://doi.org/10.48550/arXiv.1901.02860>
- [13] 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, Jia Shi 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, Qincheng Wang, Qinyu Chen, Qiusi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shutong Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanbiao 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, Xinman 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, Yudian 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, Yuxian 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. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] <https://doi.org/10.48550/arXiv.2501.12948>
- [14] Sujian Dutta, Sayantan Mahinder, Raviteja Anantha, and Bortik Bandopadhyay. 2024. Applying RLAI for Code Generation with API-usage in Lightweight LLMs. arXiv:2406.20060 [cs.CL] <https://doi.org/10.48550/arXiv.2406.20060>
- [15] 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. 2025. AReAL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning. arXiv:2505.24298 [cs.LG] <https://doi.org/10.48550/arXiv.2505.24298>
- [16] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 135–153. <https://www.usenix.org/conference/osdi24/presentation/fu>
- [17] Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. 2024. Efficient LLM Scheduling by Learning to Rank. arXiv:2408.15792 [cs.LG] <https://doi.org/10.48550/arXiv.2408.15792>
- [18] Google Cloud Platform. 2025. GCP Compute Engine Instance. <https://cloud.google.com/compute/docs/instances>.
- [19] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jellmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kunal Lakhotia, Lauren Rantala-Young, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Paspuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsim-poukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Omur Celebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoqiang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stéphane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkze, Vincent Gouget, Virginie Do, Vish Vogeti, Vitor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenying Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delprat Coudert, Zheng Yan, Zhengcheng Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkan Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood,

- Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Cagioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Sweet, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lashya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabasa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangarabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Sibi, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 Herd of Models. *arXiv:2407.21783* [cs.AI] <https://doi.org/10.48550/arXiv.2407.21783>
- [20] Tyler Griggs, Sumanth Hegde, Eric Tang, Shu Liu, Shiyi Cao, Dacheng Li, Charlie Ruan, Philipp Moritz, Kourosh Hakhmaneshi, Richard Liaw, Akshay Malik, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Evolving SkyRL into a Highly-Modular RL Framework. *Notion Blog*.
- [21] Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, Huazhong Li, Wenjie Liu, Yu Huang, Yixiang Zhang, Chenyi Pan, Jing Wang, Xin Huang, Chunsheng Li, and Jianping Wu. 2025. AsyncFlow: An Asynchronous Streaming RL Framework for Efficient LLM Post-Training. *arXiv:2507.01663* [cs.LG] <https://doi.org/10.48550/arXiv.2507.01663>
- [22] David Y. Hancock, Jeremy Fischer, John Michael Lowe, Winona Snapp-Childs, Marlon Pierce, Suresh Marru, J. Eric Coulter, Matthew Vaughn, Brian Beck, Nirav Merchant, Edwin Skidmore, and Gwen Jacobs. 2021. Jetstream2: Accelerating Cloud Computing via Jetstream. In *Practice and Experience in Advanced Research Computing (PEARC '21)*. Association for Computing Machinery, New York, NY, USA, Article 11, 8 pages. doi:10.1145/3437359.3465565
- [23] Horace He and Thinking Machines Lab. 2025. Defeating Nondeterminism in LLM Inference. *Thinking Machines Lab: Connectionism* (2025). doi:10.64434/tml.20250910 <https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/>
- [24] Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Marón, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, Léonard Hussenot, Robert Dadashi, Gabriel Dulac-Arnold, Manu Orsini, Alexis Jacq, Johan Ferret, Nino Vieillard, Seyed Kamyar Seyed Ghasemipour, Sertan Girgin, Olivier Pietquin, Feryal Behbahani, Tamara Norman, Abbas Abdulmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Abe Friesen, Ruba Haroun, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. 2020. *Acme: A Research Framework for Distributed Reinforcement Learning*. *arXiv preprint arXiv:2006.00979* (2020). <https://doi.org/10.48550/arXiv.2006.00979>
- [25] Zhenyu Hou, Pengfan Du, Yilin Niu, Zhengxiao Du, Aohan Zeng, Xiao Liu, Minlie Huang, Hongning Wang, Jie Tang, and Yuxiao Dong. 2024. Does RLHF Scale? Exploring the Impacts From Data, Model, and Method. *arXiv:2412.06000* [cs.CL] <https://doi.org/10.48550/arXiv.2412.06000>
- [26] Jian Hu, Jason Klein Liu, and Shen Wei. 2025. REINFORCE++: An Efficient RLHF Algorithm with Robustness to Both Prompt and Reward Models. *arXiv:2501.03262* [cs.CL] <https://doi.org/10.48550/arXiv.2501.03262>
- [27] Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. 2024. OpenRLHF: An Easy-to-use, Scalable and High-performance RLHF Framework. *arXiv:2405.11143* [cs.AI] <https://doi.org/10.48550/arXiv.2405.11143>
- [28] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. 2022. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research* 23, 274 (2022), 1–18. <https://dl.acm.org/doi/10.5555/3586589.3586863>
- [29] Tao Huang, Pengfei Chen, Kyoka Gong, Jocky Hawk, Zachary Bright, Wenxin Xie, Keqiang Huang, and Zhi Ji. 2024. ENOVA: Autoscaling towards Cost-effective and Stable Serverless LLM Serving. *arXiv:2407.09486* [cs.DC] <https://arxiv.org/abs/2407.09486>
- [30] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *arXiv preprint arXiv:2403.07974* (2024). <https://doi.org/10.48550/arXiv.2403.07974>
- [31] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. <https://doi.org/10.1145/3600006.3613165>
- [32] Xiaocan Li, Shiliang Wu, and Zheng Shen. 2026. A-3PO: Accelerating Asynchronous LLM Training with Staleness-aware Proximal Policy Approximation. *arXiv:2512.06547* [cs.LG] <https://doi.org/10.48550/arXiv.2512.06547>
- [33] Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. 2024. ReMax: A Simple, Effective, and Efficient Reinforcement Learning Method for Aligning Large Language Models. *arXiv:2310.10505* [cs.LG] <https://doi.org/10.48550/arXiv.2310.10505>
- [34] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. *arXiv:2302.11665* [cs.LG] <https://doi.org/10.48550/arXiv.2302.11665>
- [35] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. *arXiv:1712.09381* [cs.AI] <https://doi.org/10.48550/arXiv.1712.09381>
- [36] Mengfan Liu, Wei Wang, and Chuan Wu. 2025. Optimizing Distributed Deployment of Mixture-of-Experts Model Inference in Serverless Computing. *arXiv:2501.05313* [cs.DC] <https://doi.org/10.48550/arXiv.2501.05313>
- [37] Yuhuan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntao Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. *arXiv:2310.07240* [cs.NI] <https://doi.org/10.48550/arXiv.2310.07240>
- [38] Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. 2025. Agent Lightning: Train ANY AI Agents with Reinforcement Learning. *arXiv:2508.03680* [cs.AI] <https://arxiv.org/abs/2508.03680>
- [39] Maxwell Jia. 2025. American Invitational Mathematics Examination (AIME) 2024 Dataset. https://huggingface.co/datasets/Maxwell-Jia/AIME_2024
- [40] Zhiyu Mei, Wei Fu, Jiaxuan Gao, Guangju Wang, Huanchen Zhang, and Yi Wu. 2024. SRL: Scaling Distributed Reinforcement Learning to Over Ten Thousand Cores. *arXiv:2306.16688* [cs.DC] <https://doi.org/10.48550/arXiv.2306.16688>
- [41] Dirk Merkel et al. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal* (2014).
- [42] Microsoft Azure. 2025. Microsoft Azure Functions. <https://azure.microsoft.com/pricing/details/functions/>
- [43] Microsoft Azure. 2025. Microsoft Azure Virtual Machines. <https://azure.microsoft.com/pricing/details/virtual-machines/>
- [44] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. *arXiv:1712.05889* [cs.DC] <https://doi.org/10.48550/arXiv.1712.05889>
- [45] Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. 2025. Asynchronous RLHF: Faster and More Efficient Off-Policy RL for Language Models. *arXiv:2410.18252* [cs.LG] <https://doi.org/10.48550/arXiv.2410.18252>

- [46] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Batetscu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brit-tany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew Kon-drich, Aris Konstantinidis, Kyle Kopic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O’Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giammatista Parascan-dolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorný, Michelle Pokrass, Vitchyr P. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schul-man, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Pet-roski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vi-jayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] <https://doi.org/10.48550/arXiv.2303.08774>
- [47] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schul-man, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL] <https://doi.org/10.48550/arXiv.2203.02155>
- [48] Olga Poppe, Qun Guo, Willis Lang, Pankaj Arora, Morgan Oslake, Shize Xu, and Ajay Kalhan. 2022. Moneyball: proactive auto-scaling in Microsoft Azure SQL database serverless. *Proc. VLDB Endow.* 15, 6 (Feb. 2022), 1279–1287. doi:10.14778/3514061.351473
- [49] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. 2024. Efficient Interactive LLM Serving with Proxy Model-based Sequence Length Prediction. arXiv:2404.08509 [cs.DC] <https://doi.org/10.48550/arXiv.2404.08509>
- [50] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] <https://doi.org/10.48550/arXiv.2305.18290>
- [51] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deep-Speed: 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 (Virtual Event, CA, USA) (KDD ’20)*. Association for Computing Machinery, New York, NY, USA, 3505–3506. doi:10.1145/3394486.3406703
- [52] Redis. 2025. Redis Official Website. <http://redis.io/>.
- [53] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In *First Conference on Language Modeling*. <https://openreview.net/forum?id=Ti67584b98>
- [54] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. arXiv:2102.01887 [cs.DC] <https://doi.org/10.48550/arXiv.2102.01887>
- [55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] <https://doi.org/10.48550/arXiv.1707.06347>
- [56] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX annual technical conference (USENIX ATC)*.
- [57] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeek-Math: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv:2402.03300 [cs.CL] <https://doi.org/10.48550/arXiv.2402.03300>
- [58] Gerald Shen, Zhilin Wang, Olivier Delalleau, Jiaqi Zeng, Yi Dong, Daniel Egert, Shengyang Sun, Jimmy Zhang, Sahil Jain, Ali Taghibakhshi, Markel Sanz Ausin, Ashwath Aithal, and Olexii Kuchaiev. 2024. NeMo-Aligner: Scalable Toolkit for Efficient Model Alignment. arXiv:2405.01481 [cs.CL] <https://doi.org/10.48550/arXiv.2405.01481>
- [59] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. HybridFlow: A Flexible and Efficient RLHF Framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys ’25)*. ACM, 1279–1297. doi:10.1145/3689031.3696075
- [60] Mohammad Shoeybi, Mostafa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] <https://doi.org/10.48550/arXiv.1909.08053>
- [61] Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. 2024. A Long Way to Go: Investigating Length Correlations in RLHF. arXiv:2310.03716 [cs.CL] <https://doi.org/10.48550/arXiv.2310.03716>
- [62] Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyang Wang, and Furu Wei. 2024. You Only Cache Once: Decoder-Decoder Architectures for Language Models. arXiv:2405.05254 [cs.CL] <https://doi.org/10.48550/arXiv.2405.05254>
- [63] Kimi Team, Yifan Bai, Yiping Bao, Y. Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, Ruijie Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Chenxiao Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Qizheng Gu, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Yangyang Hu, Xenzheng Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yang Li, Yanhao Li, Yiwei Li, Zhaoxue Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Haoyu Lu, Lijun Lu, Yashuo Luo, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Zeyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Lin Sui, Xinjie Sun, Flood Sung, Yunpeng Tai, Heyi Tang, Jiawen Tao, Qifeng Teng, Chaoran Tian, Chensi Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Si Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Haoning Wu, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Jing Xu, Jing Xu, Junjie Yan, Yuzi Yan, Hao Yang, Xiaofei Yang, Yi Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhehui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Siyu Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Shaojie Zheng, Longguang Zhong, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. 2026. Kimi K2: Open Agentic Intelligence. arXiv:2507.20534 [cs.LG] <https://arxiv.org/abs/2507.20534>

- [64] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, and Zonghan Yang. 2025. Kimi k1.5: Scaling Reinforcement Learning with LLMs. arXiv:2501.12599 [cs.AI] <https://doi.org/10.48550/arXiv.2501.12599>
- [65] Wei Ming Thor. 2025. How To Calculate GPU VRAM Requirements for a Large-Language Model. ApX Machine Learning. <https://apxml.com/posts/how-to-calculate-vram-requirements-for-an-llm> Last updated: July 15, 2025.
- [66] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shriti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanur Goswami, Naman Goyal, Anthony Hartshorn, Saghar 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, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, 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, Aurolien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] <https://doi.org/10.48550/arXiv.2307.09288>
- [67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] <https://doi.org/10.48550/arXiv.1706.03762>
- [68] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Galoudec. 2020. TRL: Transformer Reinforcement Learning. <https://github.com/huggingface/trl>.
- [69] Zihan Wang, Kangrui Wang, Qinqing Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. 2025. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning. arXiv:2504.20073 [cs.LG] <https://arxiv.org/abs/2504.20073>
- [70] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453 [cs.CL] <https://doi.org/10.48550/arXiv.2309.17453>
- [71] Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. 2024. Iterative Preference Learning from Human Feedback: Bridging Theory and Practice for RLHF under KL-Constraint. arXiv:2312.11456 [cs.LG] <https://doi.org/10.48550/arXiv.2312.11456>
- [72] Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. 2024. Is DPO Superior to PPO for LLM Alignment? A Comprehensive Study. arXiv:2404.10719 [cs.CL] <https://doi.org/10.48550/arXiv.2404.10719>
- [73] Qwen: An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yujiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] <https://doi.org/10.48550/arXiv.2412.15115>
- [74] Zhewei Yao, Reza Yazdani Aminabadi, Olatunji Ruwase, Samyam Rajbhandari, Xiaoxia Wu, Ammar Ahmad Awan, Jeff Rasley, Minjia Zhang, Conglong Li, Connor Holmes, Zhongzhou Zhou, Michael Wyatt, Molly Smith, Lev Kurlenko, Heyang Qin, Masahiro Tanaka, Shuai Che, Shuaiwen Leon Song, and Yuxiong He. 2023. DeepSpeed-Chat: Easy, Fast and Affordable RLHF Training of ChatGPT-like Models at All Scales. arXiv:2308.01320 [cs.LG] <https://doi.org/10.48550/arXiv.2308.01320>
- [75] Hanfei Yu, Jacob Carter, Hao Wang, Devesh Tiwari, Jian Li, and Seung-Jong Park. 2025. Nitro: Boosting Distributed Reinforcement Learning with Serverless Computing. In *51st International Conference on Very Large Data Bases (VLDB)*. <https://doi.org/10.14778/3696435.3696441>
- [76] Hanfei Yu, Jian Li, Yang Hua, Xu Yuan, and Hao Wang. 2024. Cheaper and Faster: Distributed Deep Reinforcement Learning with Serverless Computing. In *Proceedings of the AAAI Conference on Artificial Intelligence*. <https://doi.org/10.1609/aaai.v38i15.29592>
- [77] Hanfei Yu, Hao Wang, Devesh Tiwari, Jian Li, and Seung-Jong Park. 2024. Stellaris: Staleness-Aware Distributed Reinforcement Learning with Serverless Computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*. <https://doi.org/10.1109/SC41406.2024.00045>
- [78] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tianfan Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. 2025. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. arXiv:2503.14476 [cs.LG] <https://doi.org/10.48550/arXiv.2503.14476>
- [79] Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. 2025. Give Me FP32 or Give Me Death? Challenges and Solutions for Reproducible Reasoning. arXiv:2506.09501 [cs.CL] <https://doi.org/10.48550/arXiv.2506.09501>
- [80] Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, Tianfan Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncal Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Ru Zhang, Xin Liu, Mingxuan Wang, Yonghui Wu, and Lin Yan. 2025. VAPO: Efficient and Reliable Reinforcement Learning for Advanced Reasoning Tasks. arXiv:2504.05118 [cs.AI] <https://doi.org/10.48550/arXiv.2504.05118>
- [81] Hao Zhang, Mingjie Liu, Shaokun Zhang, Songyang Han, Jian Hu, Zhenghui Jin, Yuchi Zhang, Shizhe Diao, Ximing Lu, Binfeng Xu, Zhiding Yu, Jan Kautz, and Yi Dong. 2026. ProRL Agent: Rollout-as-a-Service for RL Training of Multi-Turn LLM Agents. arXiv:2603.18815 [cs.AI] <https://doi.org/10.48550/arXiv.2603.18815>
- [82] Yanli Zhao, Andrew Gu, Rohan Varma, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC] <https://doi.org/10.48550/arXiv.2304.11277>
- [83] Haizhong Zheng, Jiawei Zhao, and Beidi Chen. 2025. Prosperity before Collapse: How Far Can Off-Policy RL Reach with Stale Data on LLMs? arXiv:2510.01161 [cs.LG] <https://doi.org/10.48550/arXiv.2510.01161>
- [84] 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. 2024. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104 [cs.AI] <https://doi.org/10.48550/arXiv.2312.07104>
- [85] Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. 2023. Secrets of RLHF in Large Language Models Part I: PPO. arXiv:2307.04964 [cs.CL] <https://doi.org/10.48550/arXiv.2307.04964>
- [86] Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023. Response Length Perception and Sequence Scheduling: An LLM-Empowered LLM Inference Pipeline. arXiv:2305.13144 [cs.CL] <https://doi.org/10.48550/arXiv.2305.13144>
- [87] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv:2401.09670 [cs.DC] <https://doi.org/10.48550/arXiv.2401.09670>
- [88] Yinmin Zhong, Zili Zhang, Xiaoni Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, Hongyu Zhou, Yimin Jiang, Yibo Zhu, and Daxin Jiang. 2025. StreamRL: Scalable, Heterogeneous, and Elastic RL for LLMs with Disaggregated Stream Generation. arXiv:2504.15930 [cs.LG] <https://doi.org/10.48550/arXiv.2504.15930>
- [89] Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, and Xin Jin. 2024. Optimizing RLHF Training for Large Language Models with Stage Fusion. *arXiv preprint arXiv:2409.13221* (2024). <https://doi.org/10.48550/arXiv.2409.13221>
- [90] Huanzhou Zhu, Bo Zhao, Gang Chen, Weifeng Chen, Yijie Chen, Liang Shi, Yaodong Yang, Peter Pietzuch, and Lei Chen. 2022. MSRL: Distributed Reinforcement Learning with Dataflow Fragments. arXiv:2210.00882 [cs.LG] <https://doi.org/10.48550/arXiv.2210.00882>
- [91] Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. 2025. slime: An LLM post-training framework for RL Scaling. <https://github.com/THUDM/slime>. GitHub repository. Corresponding author: Xin Lv.