PipelineRL: Faster On-policy Reinforcement Learning for Long Sequence Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Reinforcement Learning (RL) is increasingly utilized to enhance the reasoning capabilities of Large Language Models (LLMs). However, effectively scaling these RL methods presents significant challenges, primarily due to the difficulty in maintaining high AI accelerator utilization without generating stale, off-policy data that harms common RL algorithms. This paper introduces PipelineRL, an approach designed to achieve a superior trade-off between hardware efficiency and data on-policyness for LLM training. PipelineRL employs concurrent asynchronous data generation and model training, distinguished by the novel *in-flight weight updates*. This mechanism allows the LLM generation engine to receive updated model weights with minimal interruption during the generation of token sequences, thereby maximizing both the accelerator utilization and the freshness of training data. Experiments conducted on long-form reasoning tasks using 32 H100 GPUs demonstrate that PipelineRL achieves approximately $\sim 2x$ faster learning compared to conventional RL baselines while maintaining highly on-policy training data. A scalable and modular open-source implementation of PipelineRL is also released as a key contribution.

1 Introduction

Reinforcement Learning (RL) has recently become a popular tool to enhance the reasoning and agentic capabilities of Large Language Models (LLMs) [Guo et al., 2025, Wei et al., 2025]. While RL expands the range of training signals one can use to enhance LLMs, this advanced learning paradigm comes with extra challenges, including being particularly hard to effectively scale to more compute. The scaling difficulty arises from the fact that AI accelerators (like GPUs and TPUs) deliver high throughput only when generating sequences at a large batch size. Hence, naively adding more accelerators to an on-policy RL setup brings increasingly diminishing learning speed improvements because the per-accelerator throughput decreases, while the overall generation latency reaches a plateau. The common workaround of generating training data for multiple optimizer steps results in a lag between the currently trained policy and the behavior policy that generates the training data. The lagging off-policy data is known to harm the commonly used effective RL algorithms [Noukhovitch et al., 2024], including, REINFORCE [Williams, 1992], PPO [Schulman et al., 2017] and GRPO [Shao et al., 2024, Guo et al., 2025], because these algorithms were designed to be trained with on-policy or near on-policy data, with the behavior and current policy being very close.

In this paper, we present the PipelineRL approach to RL for LLMs that achieves a better trade-off between hardware utilization and on-policy learning. Like prior work on efficient RL [Espeholt et al., 2018, 2019], PipelineRL features concurrent asynchronous data generation and training. PipelineRL adapts prior asychronous RL ideas to long-sequence generation with LLMs by introducing *in-flight weight updates*. As shown in Figure 1, during an in-flight weight update the LLM generation engine

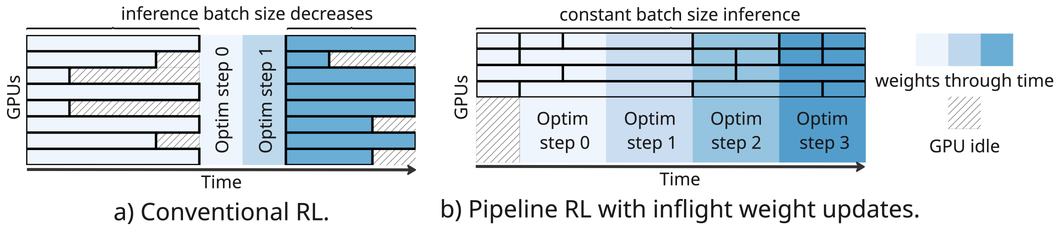


Figure 1: **a)** Conventional RL alternates between using all the GPUs for generation and then training. **b)** PipelineRL runs generation and training concurrently, always using the freshest model weights for generations thanks to the in-flight weight updates.

only briefly pauses to receive the model weights via a high-speed inter-accelerator network, and then proceeds to continue the generation of in-progress token sequences. In-flight updates eliminate the wasteful waits for the last sequence to finish, ensure high accelerator utilization at a constant generation batch size, and maximize the policy adherence of the recently generated tokens.

Our experiments on RL training for long-form reasoning show that on 4 DGX-H100 nodes, PipelineRL learns $\sim 2x$ faster than the comparable conventional RL baseline. We also observe that PipelineRL training data stays highly on-policy, and that models trained by PipelineRL perform comparably to similarly trained models from the literature. Lastly, a key contribution of this work is a scalable and modular PipelineRL implementation that we release as open-source software.[1]

2 Background

2.1 Reinforcement Learning for Large Language Models

Reinforcement learning (RL) is commonly used to train Large Language Models (LLM) to respect human preferences [Ouyang et al., 2022] for the LLM's outputs or to perform long-form reasoning to solve problems [Guo et al., 2025]. One can view LLM's weights as parameterizing a multi-step policy that assigns probabilities to the next token y_i given the prompt x and the previously generated tokens $y_{<i}$:

$$\pi(y|x) = \prod_{i=1}^{n} \pi(y_i|x, y_{<i}). \tag{1}$$

Recent works have shown that variations of basic policy gradient algorithms such as REINFORCE [Williams, 1992] are as effective for training LLMs as more sophisticated alternatives [Ahmadian et al., 2024, Roux et al., 2025]. Given a set of prompts $x_1, \ldots, x_m$, REINFORCE maximizes the expected return $J(\pi)$ of the policy π by following an estimate $\tilde{\nabla} J(\pi)$ of the policy gradient $\nabla J(\pi)$:

$$J(\pi) = \frac{1}{m} \sum_{j=1}^{m} \left[\mathbb{E}_{y\sim\pi(\cdot|x_j)} R(x_j, y)\right] \tag{2}$$

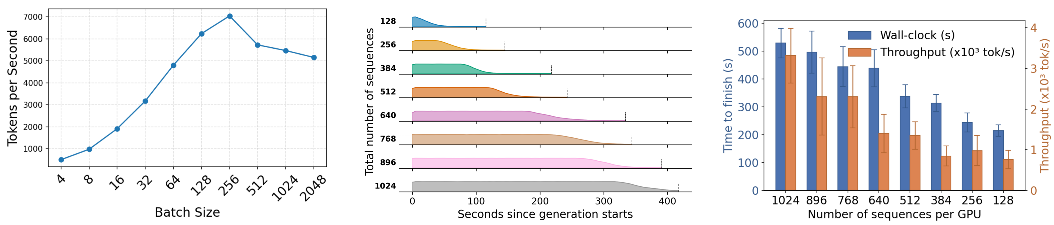
$$\nabla J(\pi) = \frac{1}{m} \sum_{j=1}^{m} \left[\mathbb{E}_{y\sim\pi(\cdot|x_j)} \nabla \log \pi(y \mid x_j) R(x_j, y)\right] \tag{3}$$

$$\tilde{\nabla} J(\pi) = \frac{1}{mK} \sum_{j=1}^{m} \sum_{k=1}^{K} \nabla \log \pi(y \mid x_j) \left(R(x_j, y_k) - v_k(x_j)\right), \tag{4}$$

where $v_k(x_j)$ is the control variate term that reduces the estimate's variance, and K is the number of samples per prompt x. In this study, we use the empirical mean $v_k(x_j) = \sum_{k=1}^{K} R(x_j, y_k)/K$ as the control variate.

In most practical RL setups, the *current policy* π will often slightly differ from the *behavior policy* μ that generates y_k. This difference is usually handled by either a trust region constraint [Schulman

[1]The code is available online under Apache 2 license, we will add the link to the camera-ready version



(a) Throughput vs batch size. (b) Inference batch size vs time. (c) Time vs Throughput.

Figure 2: **Analysis of generation times and throughput.** We perform all measurements using a vLLM engine serving a Qwen 2.5 7B model on a H100 GPU. **(a)** Short prompt generation throughput increases up to batch size 256. **(b)** Generation batch size gradually decreases to suboptimal values as the engine finishes sequences **(c)** Generation time reaches a plateau and throughput decreases as the number of sequences per GPU goes down. We report the average of 5 runs and 95% CI.

et al., 2017] or using Importance Sampling (IS). In practice, the importance weights are truncated to reduce the variance of the estimator [Munos et al., 2016, Espeholt et al., 2018]:

$$\tilde{\nabla}_{IS} J(\pi) = \frac{1}{mK} \min \left(c, \frac{\pi(y \mid x)}{\mu(y \mid x)} \right) (R(x_j, y_k) - v_k(x_j)) \nabla \log \pi(y \mid x). \tag{5}$$

The Effective Sample Size (ESS) [Kong, 1992] is commonly used to quantify the quality of importance sampling estimators in RL [Schlegel et al., 2019, Fakoor et al., 2020]. When using off-policy RL, ESS measures how many samples from the current policy π would yield equivalent performance to weighted samples from the behavior policy μ. The (normalized) ESS is defined as:

$$\text{ESS} = \left(\sum_{i=1}^{N} w_i \right)^2 / N \sum_{i=1}^{N} w_i^2 \tag{6}$$

where w_i are importance weights for a sample of size N. This metric effectively ranges between 0 and 1 when normalized, with values closer to 1 indicating more efficient sampling, e.g. the ESS of on-policy data is exactly 1. Small ESS will result in a high variance REINFORCE gradient estimate and might destabilize the learning process.

2.2 Conventional RL

Most RL implementations alternate between generating sequences and training the policy on the generated data. We refer to this approach as Conventional RL and describe it in detail in Algorithm 1. When training involves doing $G > 1$ optimizer steps, the current policy π gets ahead of the behavior policy μ that was used to generate the data. We adopt the term *lag* to refer to the number of optimizer steps between μ and π.

2.3 Efficient Sequence Generation with LLMs

Transformer models generate sequences one token at a time, left-to-right. To make this process efficient, advanced generation (inference) engines such as vLLM and SGLang process a batch of sequences at a time, while carefully managing their past keys and values in a paged structure called KV cache [Kwon et al., 2023]. All modern generation engines support adding new generation requests *in-flight* to the ones in progress without stopping the generation process. Based on accelerator specifications, generation engines should achieve the maximum generation throughput at very large batch sizes of several thousand sequences [2]. In practice, at very large batch sizes, the per-sequence latency can become prohibitively high, KV cache may grow too large to fit in accelerator memory, or the request queue management overheads can dominate.

[2]https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Algorithm 1 Conventional RL

Require: Current policy π.
Require: Optimizer state opt_state.
Require: Number of optimizer steps per RL step G.
Require: Training batch size B.
  while True **do**
    // **generation** ▷ RL step starts
    $\mu \leftarrow \pi$ ▷ Initialize behavior policy μ
    sequences $\leftarrow$ generate BG sequences from μ
    batches $\leftarrow$ split sequences in G batches of size B
    // **training**
    lag $\leftarrow 0$ ▷ lag between μ and π
    for batch **in** batches **do**
      π, opt_state $\leftarrow$ optimizer_step(π, opt_state, batch)
      lag $\leftarrow$ lag + 1
    end for ▷ RL step ends
  end while

3 The learning speed ceiling of Conventional RL

Reinforcement learning for LLMs can be slow when the LLM is trained to generate long sequences of tokens, e.g., long-form reasoning to solve mathematical problems, because each generation can take up to several minutes. Here we explain why it is challenging to effectively scale up long sequence RL, i.e. to effectively use a larger number of accelerators N to make average reward $R(t)$ at time t grow faster. As a mathematical function, one can view $R(t)$ as a composition of the functions $R(S)$ and $S(t)$, where S is the number of samples the RL learner will have processed by time t. A faster RL learner will have a higher *learning speed* $\frac{dR}{dt}$ which we can express as the product of *learning effectiveness* and *learning throughput* as follows:

$$\underbrace{\frac{dR}{dt}}_{\text{speed}} = \underbrace{\frac{dR}{dS}}_{\text{effectiveness}} \times \underbrace{\frac{dS}{dt}}_{\text{throughput}} . \tag{7}$$

The Conventional RL algorithm from Algorithm 1 has the highest $\frac{dR}{dS}$ when it is fully on-policy, i.e., when one performs only one optimizer step per each RL step. Yet the throughput $\frac{dS}{dt}$ in the pure on-policy case can be low because the accelerators will be working on at most batch size B samples at a time. Increasing the number of accelerators N will yield diminishing returns in increasing $\frac{dS}{dt}$, because the throughput of each accelerator will decrease when the number of samples per accelerator $\frac{B}{N}$ goes below the optimal range (Figure 2c). For example, see Figure 2a for inference throughput for a 7B Qwen model on a single H100 GPU. One can see that the throughput increases almost linearly up to the generation batch size of 128. Hence, e.g. using $2N$ GPUs to generate 32 samples will not be much faster than using N GPUs to generate 64. Furthermore, as the LLM finishes the shorter generations, there will be fewer longer generations still in progress, see Figure 2b for an illustration. Hence, to make good use of the hardware, one should use each accelerator to generate many times more sequences than the optimal batch size.

Commonly, to increase the throughput, most practitioners perform multiple $G > 1$ optimizer steps per RL step, which entails generating BG rollouts at each generation stage. This way, one can often achieve a higher throughput $\frac{dS}{dt}$ by increasing N up to a point when $\frac{BG}{N}$ becomes too small. It is, however, known from the literature that going too off-policy by using a high value of G will eventually decrease the learning effectiveness $\frac{dR}{dN}$ [Noukhovitch et al., 2024]. Clearly, at some points, the rollouts from the old policy become too stale and no longer useful as the source of learning signal for the current policy. Hence, given a fixed optimizer batch size B, one scales up Conventional RL by increasing G and N until the product $\frac{dR}{dS}\frac{dS}{dt}$ no longer improves, and the hard ceiling of $\frac{dR}{dt}$ for the given number of accelerators N is achieved.

Algorithm 2 PipelineRL

Require: Current policy weights π.
Require: Generation batch size H.
Require: Training sequence queue Q_{train}.

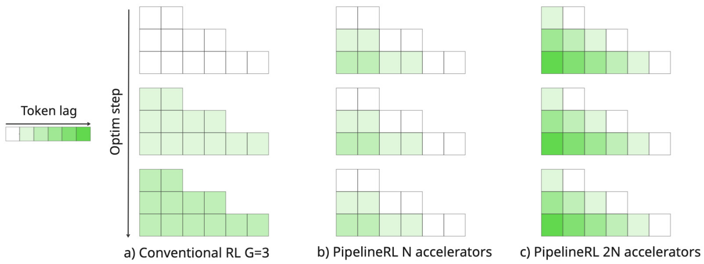
function ACTOR(π)
  sequences in progress $S_{prog} \leftarrow []$
  while True **do**
    $S_{fin}, S_{prog} \leftarrow$ pop finished sequences from S_{prog}
    $Q_{train}.put(S_{fin})$ ▷ Send finished seqs to the trainer
    if $len(S_{prog} < H)$ **then**
      add $H - len(S_{prog})$ prompts to S_{prog}
    end if
    if Trainer requests weight update **then** ▷ In-flight check for new weights
      $\pi \leftarrow$ receive_weight_update()
      $\mu \leftarrow \pi$ ▷ 0 lag between π and μ
    end if
    $S_{prog} \leftarrow$ generate next tokens with μ
  end while
end function
function TRAINER(π, opt_state)
  batch $\leftarrow$ []
  while True **do**
    batch $\leftarrow$ get B sequences from Q_{train}
    $ESS \leftarrow$ get_effective_sample_size(π, batch)
    if ESS < threshold **then**
      sleep(until Q_{train} contains on-policy data for π)
      continue
    end if
    π, opt_state $\leftarrow$ optimizer_step(π, opt_state, batch)
    request_actor_weight_update(π) ▷ In-flight weight update
  end while
end function

4 Pushing the learning speed ceiling with PipelineRL

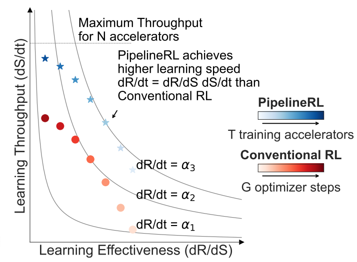
The Pipeline RL method differs from Conventional RL in two aspects: (1) running training and generation in parallel asynchronously, and (2) updating the generation weights after every optimizer step *in-flight*, i.e. without stopping the sequence generation. Algorithm 2 provides an abstracted formal description of PipelineRL in terms of two concurrent Actor and Trainer processes that communicate via a sample queue and a high-bandwidth weight transfer network.

The effectiveness-throughput trade-off for PipelineRL is the opposite of that of Conventional RL. Namely, adding more accelerators to a PipelineRL setup leads to a linear increase of $\frac{dS}{dt}$, but may eventually harm $\frac{dR}{dS}$. In Figure 3a, we illustrate how PipelineRL produces *mixed-policy sequences* in which earlier tokens are more off-policy than the recent ones. Doubling N will double the lag of the earliest tokens as well as the average lag in the PipelineRL batch. Notably, the off-policyness profile is different for PipelineRL and its conventional counterpart. Taking the average token lag as a proxy for off-policyness, in PipelineRL all batches are equally off-policy, whereas for Conventional RL later batches become progressively more off-policy. This difference makes it hard to analytically reason about the $\frac{dR}{dt}$ improvement that PipelineRL can bring over the baseline, because $\frac{dR}{dS}$ can only be estimated empirically by running RL experiments. In supplementary material, we present our simulation of how, for the same maximum lag g_{max} PipelineRL can learn 1.5x faster than Conventional RL. The empirical gains can be even larger, depending on how frequently one can make weight updates without hurting the learning effectiveness $\frac{dR}{dS}$.

Configuring PipelineRL vs Conventional RL For a fixed batch size B and a number of accelerators N, one can configure Conventional RL by choosing the number of optimizer steps G, trading off



(a) Token lags.



(b) Pareto curves.

Figure 3: **(a)** For Conventional RL, the token lag increases with the number of optimizer steps. In PipelineRL with N accelerators, the token lag varies throughout the sequence, where earlier tokens have higher lag. The lag structure in each batch is the same. Doubling the PipelineRL accelerators, everything else constant, double the lag of early tokens. **(b)** Schematic illustration of PipelineRL's throughput-effectiveness trade-off as a function of training accelerators T and of Conventional RL as a function of lag G. PipelineRL achieves a higher $\frac{dR}{dS}\frac{dS}{dt}$ for the same number N of accelerators.

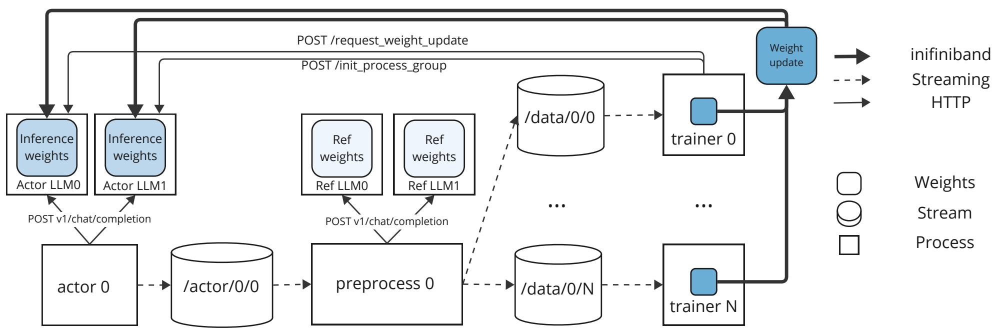
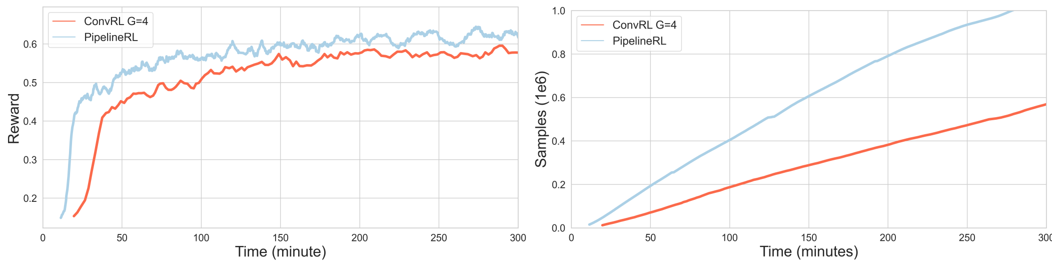


Figure 4: The three pipeline stages of PipelineRL implementation: actor, preprocessor and trainer. Earlier stages stream the data to the latter ones using Redis as the streaming broker.

the learning effectiveness for the throughput. The PipelineRL configuration can likewise be mostly reduced to a single parameter, namely the number of training accelerators T out of N available ones. Setting a higher T will almost linearly decrease the time t_{train} that is needed for the trainer to process B sequences and perform an optimizer step. T effectively determines the optimal generation batch size H to be used at all $N - T$ accelerators. Using a lower H leads to a lower maximum generation latency t_{gen}, which consequently reduces the maximum lag $g_{max} = \lceil t_{gen}/t_{train} \rceil$. Hence, it makes sense to use the smallest H that suffices to produce enough training data. Consequently, the maximum lag g_{max} for PipelineRL grows with the number of training accelerators T, as higher T requires a higher H and leads to a lower t_{train} and a higher t_{gen}. On the contrary, the sample throughput of PipelineRL grows with T up to a point when $N - T$ accelerators cannot generate enough data for the over-powered trainer. We recommend avoiding extreme configurations with T too high (very high lag G) and T too low (bad hardware utilization, one can just as well scale down the compute). Figure 3b visualizes how different configurations of PipelineRL and Conventional RL achieve different learning effectiveness $\frac{dR}{dS}$ and throughput $\frac{dS}{dt}$, with PipelineRL setups reaching higher $\frac{dR}{dt} = \frac{dS}{dt}\frac{dR}{dS}$ isocurves.

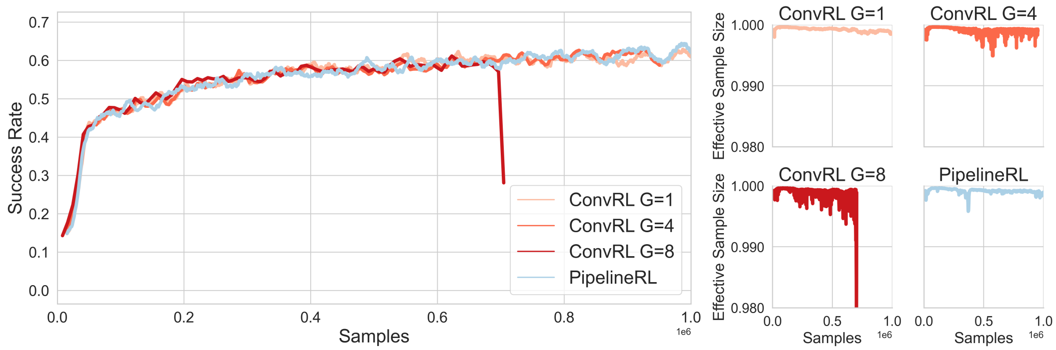
PipelineRL Safety Mechanism While in-flight weight updates can be useful, on the flip side, the mixed-policy sequences generated by the in-flight behavior policy can present a risk to the stability of the training process, in particular because after an in-flight weight update, the generation server continues with the stale key and value vectors that were computed by a prior version of the model. To remediate these risks, we monitor the Effective Sample Size (ESS) of each training batch. Once ESS drops below a certain threshold, we stop updating the current policy until it accumulates a full batch of purely on-policy sequences, see lines 21-23 in Algorithm 2.



(a) Learning speed (dR/dt)

(b) System throughput (dS/dt)

Figure 5: **Learning speed and throughput.** PipelineRL achieves higher throughput and learning speed than Conventional RL with G=4 optimizer steps per each RL step.



(a) Reward R after training on S samples

(b) ESS after S samples

Figure 6: **(a)** PipelineRL attains the same average rewards for each number of training samples as pure on-policy $G = 1$ Conventional RL **(b)** PipelineRL stays mostly on-policy.

Architecture and Implementation Details Our PipelineRL implementation concurrently runs many distributed vLLM generation engines and DeepSpeed training workers in a three stage pipeline that we describe in Figure 4. The middle Preprocessor stage that we omitted from Algorithm 2 for simplicity, computes reference model log-probabilities often used in Reinforcement Learning from Human Feedback [Ouyang et al., 2022]. The PipelineRL architecture is highly modular — any generation software that supports the three HTTP API endpoints that PipelineRL requires can be easily integrated in the future. The three APIs are the popular `/v1/chat/completions` for generation, `/init_process_group` for creating the weight transfer process group, and `/request_weight_update` for initiating the in-flight weight update. Key optimizations in PipelineRL include online sequence packing for fast training and using ring buffers to minimize the lag when earlier pipeline stages run faster than the later ones, e.g. when the trainer makes a checkpoint.

5 Experiment

For the experimental validation of PipelineRL's high learning effectiveness $\frac{dR}{dS}$ and throughput $\frac{dS}{dt}$, we have chosen the challenging task of training a base (i.e. not instruction-tuned) model to perform long-form reasoning to solve mathematical problems. We find this task to be a great testbed for PipelineRL because the policy undergoes rapid changes over the course of training. In particular, the length of generated sequences grows dramatically [Guo et al., 2025], making it essential to stay on-policy for effective learning.

Experimental setup. For each experiment, we train the Qwen 2.5 base model [Yang et al., 2024] with 7B parameters on 17K math problems from the OpenReasoner Zero dataset [Hu et al., 2025] for 1000 optimizer steps with the batch size $B = 1024$. We use Adam optimizer [Kingma, 2014] with the learning rate 1e-6. We run the PipelineRL experiments on 4 DGX-H100 nodes, using 16 GPUs for generation at batch size $H = 64$ and 16 GPUs for training. We tweak PipelineRL to simulate Conventional RL by accumulating and shuffling a buffer of BG samples at the Preprocessor stage before the G optimizer steps of each RL step start. To estimate the Conventional RL throughput, we use 4 nodes for generation at batch size $H = 128$ and 2 nodes for training, and then add a correction for training on 2x fewer GPUs than what an efficient Conventional RL implementation with a quick generation-training transition could use. We give reward 1 to any generated sequence with the correct answer and 0 otherwise. We train every model with importance weighted REINFORCE as described in Section 2 and clamp the importance weights to 5.

Table 1: Success rate of models trained with PipelineRL compared to results in the literature.

Method	**Math 500**	**AIME24**	**# samples ($\cdot 10^6$)**	**training data**
Qwen 2.5 base 7b	31.6	3.3	-	-
SimpleRL Zero [Zeng et al., 2025]	78.2	20.0	0.82	Math Level 3-5
OpenReasoner Zero [Hu et al., 2025]	$\sim$ 82.0	$\sim$ 20.0	8.2	OpenReasoner
PipelineRL (batch size 1024)	81	17.5	2.0	OpenReasoner
PipelineRL (batch size 4096)	84.6	19.8	6.2	OpenReasoner

PipelineRL learns faster due to higher throughput. We compare the learning speed of PipelineRL to that of Conventional RL with $G = 4$ optimizer steps, as that was the maximum G for which Conventional RL training was stable. PipelineRL achieves the same reward values approximately $\sim 2x$ faster than this baseline (Figure 5a) due to $\sim 2x$ faster sample throughput (Figure 5b). The main cause of the throughput increase is that GPU utilization for $G = 4$ experiment on 32 GPUs is relatively low for each GPU when it has to generate just 4096 / 32 = 256 sequences (see Figure 2b).

PipelineRL learns effectively. To better measure learning effectiveness $\frac{dR}{dS}$ of PipelineRL, we also run Conventional RL experiments with $G = 1$ and $G = 8$ optimizer steps. Notably, the $R(S)$ curves are indistinguishable for all compared methods up to a point when high G runs diverge, likely because of going too far off-policy. This result validates that PipelineRL's signature in-flight weight updates do no harm to the sequence generation process. For the PipelineRL run the ESS safety mechanism was never triggered, but in our preliminary experiments, it was sometimes activated and prevented the policy blow-up.

PipelineRL matches comparable results on reasoning tasks. Table 1 compares the test performance of PipelineRL to similar experiments that start training from the same Qwen 2.5 7B model. In this experiment we used batch size 4096 because we found it leads to a higher performance. On the math reasoning benchmarks MATH500 [Hendrycks et al., 2021] and AIME2024 [Li et al., 2024]. PipelineRL matches or exceeds the performance of Open Reasoner Zero and SimpleRL Zero.

PipelineRL stays more on-policy. To gain a better understanding of which training methods stay more on-policy, we plot the evolution of the ESS on-policyness measure throughout the training. Figure 6b shows that for a purely on-policy run with $G = 1$, ESS stays close to 1.[3] For $G = 8$, ESS generally decreases with the lag between the behavior and the current policy. We note that the magnitude of the ESS drop varies throughout training for $G = 4$ and $G = 8$ runs. The ESS of PipelineRL follows a different pattern. It stays close to ESS of $G = 1$ gold-standard run with some large drops when the current policy quickly shifts and the variance of the importance weights increases. These drops are the reason why we recommend using the ESS-based safety mechanism for PipelineRL. Notably, even though the maximum lag g_{max} in our PipelineRL experiment was around 8 on average, Figure 6b shows that PipelineRL's ESS curves look more like that of $G = 1$ on-policy run than that of $G = 8$ more off-policy experiment. We believe it is due to the lag being lower than g_{max} for a majority of tokens, since the average generated sequence length in our experiments ranged between 1K and 2K tokens, well below the 8K maximum.

[3]The reason for ESS falling below 0.999 for $G = 1$ is the consistent small difference between the log-probabilities produced by vLLM and Huggingface Transformers implementation of Qwen 2.5 model.

6 Related work

Asynchronous and high-throughput RL has been extensively studied. IMPALA [Espeholt et al., 2018] decoupled acting from learning to maximize GPU utilization. Like PipelineRL, IMPALA used truncated importance weights to estimate the value function from off-policy samples. Furthermore, IMPALA kept the policy weights constant for the length of an episode. SeedRL [Espeholt et al., 2019] proposed to update the model's parameters during an episode, resulting in trajectories where different actions were sampled by different policies. OpenAI Five [OpenAI et al., 2019] was trained using asynchronous PPO to achieve superhuman performance on Dota 2. These previous works were focused on RL for video games. Closer to our work, [Noukhovitch et al., 2024] explores asynchronous RL for LLMs. In their approach, data generation for the next G optimizer steps is synchronized with training on the previous G optimizer steps, leading to higher off-policyness than Conventional RL, unlike PipelineRL. The same study shows that offline methods such as DPO [Rafailov et al., 2023] can better tolerate off-policyness.

There exist several other scalable open-source RL implementations. veRL [Sheng et al., 2024] implements Conventional RL efficiently by using a sophisticated hybrid generation-training engine that supports quick transitions between training and generation on the same GPUs. We believe veRL's throughput would be similar to our Conventional RL baseline. Without the hybrid engine, in OpenRLHF [Hu et al., 2024] training GPUs idle during generation and vice-versa.

7 Conclusion and Discussion

We have shown how in-flight weight updates help PipelineRL break the learning speed ceiling of the conventional two-stage RL approach. We believe that for long sequence generation, in particular, this speedup would be very difficult to attain with another asynchronous RL approach, as synchronous waits for generation to finish would hurt the throughput and/or learning effectiveness. The stale KV-cache risk that in-flight updates introduce can be mitigated by recomputing the KV cache after each update, which can be done fast at a high GPU utilization, but will still lower the throughput.

We believe PipelineRL may be particular useful for training LLMs to excel at agentic behaviors that involve multiple LLM generations interspersed with environment interactions. Another promising direction for future work is to study when the recent low lag tokens in PipelineRL are helpful, and on the contrary, where PipelineRL's constantly high lag of early tokens in long sequences hurts.

Limitations PipelineRL will only bring a limited throughput increase over Conventional RL if the LLM is asked to generate the exact same number of tokens for the same prompt. In this unlikely scenario, Conventional RL will be likewise capable of maintaining a constant generation batch size. The PipelineRL's stable average token lag and the low lag of recent tokens in each batch may, however, still affect the learning effectiveness. The PipelineRL throughput advantages will likewise decrease in setups with scarce or extensive compute resources. In the former case, each GPU will get enough generation tasks for the GPU utilization to be high. In the latter, the learning speed will be bounded not by the hardware utilization but by the best possible generation latency and by the environment feedback delay.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019.

Rasool Fakoor, Pratik Chaudhari, and Alexander J Smola. P3o: Policy-on policy-off policy optimization. In *Uncertainty in artificial intelligence*, pages 1017–1027. PMLR, 2020.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. OpenRLHF: An Easy-to-use, Scalable and High-performance RLHF Framework, November 2024. URL http://arxiv.org/abs/2405.11143. arXiv:2405.11143 [cs].

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Augustine Kong. A note on importance sampling using standardized weights. *University of Chicago, Dept. of Statistics, Tech. Rep*, 348:14, 1992.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention, September 2023. URL http://arxiv.org/abs/2309.06180. arXiv:2309.06180 [cs].

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models. *arXiv preprint arXiv:2410.18252*, 2024.

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019. URL https://arxiv.org/abs/1912.06680.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

Nicolas Le Roux, Marc G Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. Tapered off-policy reinforce: Stable and efficient reinforcement learning for llms. *arXiv preprint arXiv:2503.14286*, 2025.

Matthew Schlegel, Wesley Chung, Daniel Graves, Jian Qian, and Martha White. Importance resampling for off-policy prediction. *Advances in Neural Information Processing Systems*, 32, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*, 2025.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In this paper, we propose a new asynchronous system for RL training. Our main contribution is that our system is efficient and stable, as explained in both the abstract and Section 1 (Introduction). Throughout the paper, our goal is to provide empirical evidence and theoretical justification to corroborate this contribution.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of PipelineRL in Section 7.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We do not have theorems or conjectures in the paper. However, we justify our design decisions through theoretical explanations, where all the details, including the assumptions, are clearly specified.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: In the "Experimental Setup" section (Section 5), we provide the details required to reproduce our experiments. We also plan to release our codebase (upon acceptance) that includes all the configurations we used for our experiments (an anonymized version of our codebase is provided along with the submission).

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: All the datasets used in the paper are already publicly available. We plan to release our codebase with detailed documentations upon acceptance.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: This is thoroughly discussed in the "Experimental Setup" section (Section 5) and in our codebase.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Our experiments are too costly to repeat multiple times for measuring error bars and statistical significance metrics. However, we observed that throughput (the most important metric in this study) remains stable and does not vary dramatically across different runs.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All our experiments were conducted on at most 4 DGX-H100 nodes (8 GPUs per node). We also thoroughly explain the runtime details including the throughput and other efficiency measures.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We follow the NeurIPS Code of Ethics guidelines. In the paper, we use publicly available datasets that are well-known in the community.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: PipelineRL is a general tool to speed up LLM training. It does not have positive or negative societal impact.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper does not introduce new data or models. Nonetheless, we plan to release our codebase – along with detailed instructions for using our reinforcement learning training method – under the Apache 2.0 License to promote fair and open access for the community.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The main artifacts used in this paper include the OpenReasoner Zero dataset [Hu et al., 2025], Qwen-2.5 model checkpoints [Yang et al., 2024], both properly attributed through citation. Other open-source libraries that we used in our implementation are listed as dependencies in the configuration files of our codebase.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Our submission is accompanied by the source code of our implementation, which includes a README with detailed documentation and pre-defined configuration files to facilitate the reproduction of our experiments.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No experiments involving human participants were conducted in this work.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: No experiments involving human participants were conducted in this work.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Our case study in this paper focuses on fine-tuning LLMs. Details about the models used, relevant hyperparameters, and hardware specifications are provided in the "Experimental Setup" section (Section 5).

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.