AReaL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning

Wei Fu[1,2], **Jiaxuan Gao**[1,2], **Xujie Shen**[2], **Chen Zhu**[2], **Zhiyu Mei**[1,2],
Chuyi He[2], **Shusheng Xu**[1,2], **Guo Wei**[2], **Jun Mei**[2], **Jiashu Wang**[2,3],
Tongkai Yang[2], **Binhang Yuan**[3], **Yi Wu**[1,2]

[1] IIIS, Tsinghua University, [2] Ant Research, [3] HKUST
`fuwth17@gmail.com`, `jxwuyi@gmail.com`

Motivation: RL for Advanced Reasoning

- Reinforcement Learning (RL) has become a key paradigm for enhancing the reasoning capabilities of Large Language Models (LLMs).
- RL allows an LLM to generate "thinking tokens" before providing a final answer, improving performance on complex tasks like math, coding, and logic puzzles.
- These models are often called Large Reasoning Models (LRMs).
- Effective RL training requires massive parallelization to generate large batches of sample outputs ("rollouts") for exploration.

The Problem with Existing Synchronous RL Systems

- Most large-scale RL systems are synchronous, strictly alternating between a "generation" phase and a "training" phase.
- This design ensures training stability, as the model is always trained on the most recent data samples.
- However, this approach suffers from major system-level inefficiencies.
- Primary Bottleneck: The system must wait until the longest output in a generation batch is complete before any model updates can occur.

GPU Underutilization in Synchronous RL

- The varying lengths of generated responses lead to significant idle time on most GPUs, resulting in poor resource utilization.
- This issue is shown in the timeline below, where GPUs finish at different times but must wait for the slowest one.

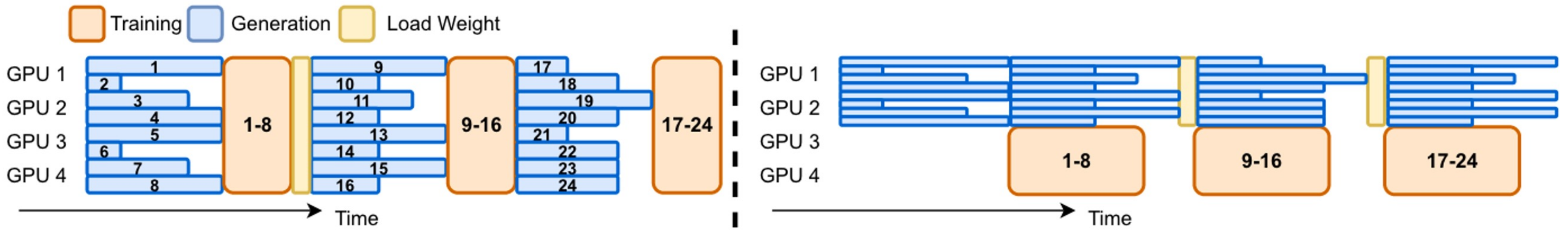


Figure 1: Execution timeline of a synchronous (left) and a one-step overlap (right) RL system showing underutilized inference devices.

Solution: AReaL

- We present AReaL, a fully asynchronous RL system that completely decouples the generation and training processes.

- Key Idea:
    - Rollout (generation) workers continuously generate new data without waiting.
    - Training workers update the model whenever a new batch of data is collected.
    - This design leads to substantially higher GPU utilization and training throughput.

AReaL System Architecture

Core Components: Workers

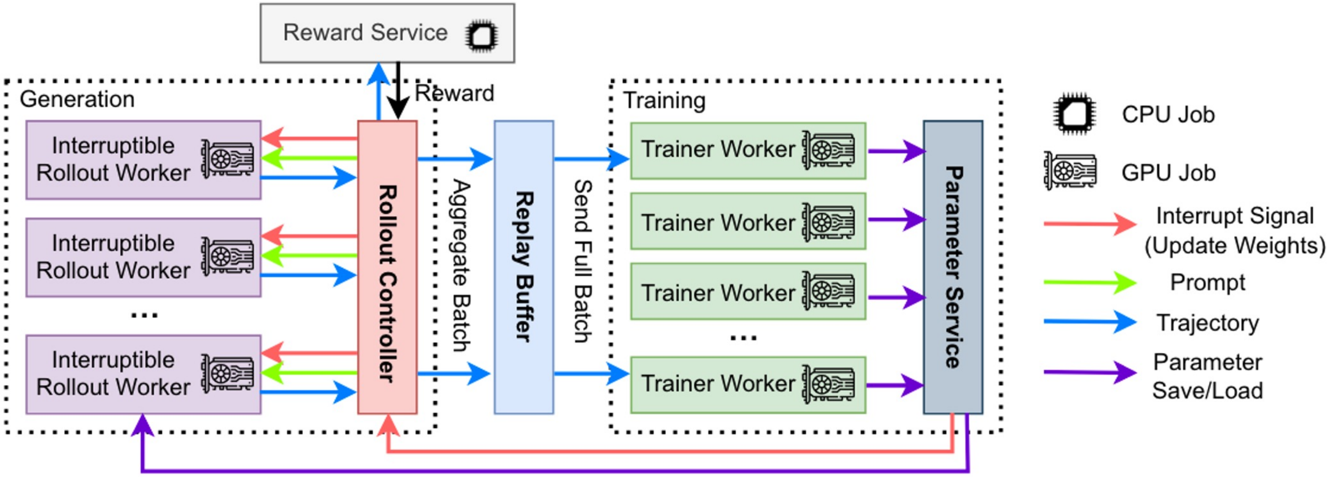


Figure 2: The AREAL architecture featuring asynchronous generation and training components.

AReaL System Architecture

Core Components: Workers

- Interruptible Rollout Workers:
  - Handle requests to generate responses from given prompts.
  - Can be interrupted by update_weights requests to load new model parameters mid-generation.
- Trainer Workers:
  - Continuously sample data from a Replay Buffer to form a training batch.
  - Perform PPO updates and save the new model parameters.

AReaL System Architecture

Core Components: Management

- Rollout Controller:
  - Acts as the bridge between all components.
  - It sends prompts to rollout workers, forwards completed trajectories to the reward service, and stores the results in the replay buffer.
- Reward Service:
  - A separate CPU job that evaluates the correctness of generated responses (e.g., by running unit tests for code).

Interruptible Generation Workflow

- When new model weights are ready, an interrupt signal is sent.
- Ongoing generations are paused, KV caches from old weights are discarded and recomputed with new weights, and then decoding continues.
- This ensures generation workers are always using the most up-to-date models possible without having to wait for a full batch to complete.

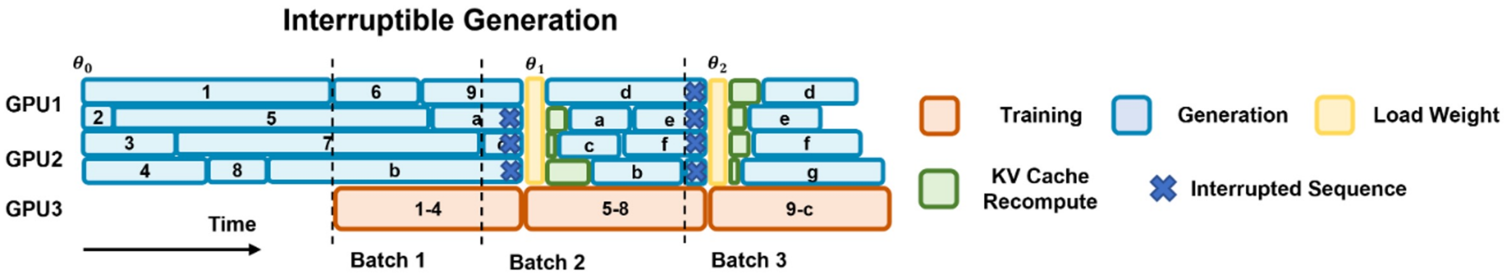


Figure 3: Illustration of generation management in AREAL. Vertical lines show the ready time for the next step training. Blue crosses show the interrupted requests when new parameters arrive.

Algorithmic Challenges of Asynchronicity

The asynchronous design introduces two main algorithmic challenges:

- Data Staleness: Training batches contain data generated from multiple, older policy versions. This can create a distribution gap and degrade learning performance.

- Inconsistent Policy Versions: A single trajectory may be generated by segments from different policy versions due to interrupts. This violates the core assumption of the standard PPO algorithm.

Solution 1:Staleness-Aware Training

- To manage data staleness, we introduce a hyperparameter, η, which defines the maximum permitted staleness for any sample in a training batch.
- When creating a new training batch for policy version i, we ensure that all data comes from policies no older than $i-\eta$.
- The Rollout Controller dynamically controls the rate of generation requests to enforce this constraint.
- This is a simple yet effective way to prevent the model from training on excessively outdated data.

Solution 2: Decoupled PPO Objective

To handle inconsistent and stale policies, we adopt a **decoupled PPO objective**.

This objective disentangles the **behavior policy** (the policy that generated the data, π_{behav}) from the **proximal policy** (the policy used as the baseline for the update, π_{prox}).

By using a more recent model as π_{prox}, we stabilize training by ensuring updates happen within a trust region of a high-quality policy, rather than an old, low-quality one.

This formulation is robust to trajectories generated by multiple policy versions.

$$J(\theta) = \mathbb{E}_{q\sim\mathcal{D}, a_t\sim\pi_{\text{behav}}}\left[\sum_{t=1}^{H} \min\Big(\underbrace{\boxed{\frac{\pi_\theta}{\pi_{\text{behav}}}}}_{\text{Importance Ratio}} \hat{A}_t,\ \overbrace{\frac{\pi_{\text{prox}}}{\pi_{\text{behav}}} \text{clip}\Big(\underbrace{\boxed{\frac{\pi_\theta}{\pi_{\text{prox}}}}}_{\text{Trust Region Center}}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_t\Big)}^{\text{Importance Ratio}}\right] \tag{4}$$

$$= \mathbb{E}_{q\sim\mathcal{D}, a_t\sim\pi_{\text{behav}}}\left[\sum_{t=1}^{H} \frac{\pi_{\text{prox}}}{\pi_{\text{behav}}} \min\left(u_t^{\text{prox}}(\theta)\hat{A}_t, \text{clip}\left(u_t^{\text{prox}}(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{5}$$

Experimental Setup

- Tasks: Challenging math (AIME24) and coding (LiveCodeBench) benchmarks.
- Models: Distilled Qwen2 models, with sizes from 1.5B to 32B parameters.
- Hardware: An H800 GPU cluster with up to 64 nodes (512 GPUs).
- Device Allocation: For AReaL, we used a fixed 75-25 split between inference and training devices, respectively, as it yielded the highest throughput in early tests.
- Baselines: We compare against state-of-the-art synchronous systems (DeepScaleR, DeepCoder) and a synchronous variant of AReaL.

Results: End-to-End Performance

1. AReaL consistently matches or improves final model performance while drastically reducing training time.
2. Across various model sizes, AReaL achieves up to a 2.77x training speedup compared to synchronous systems.

Model	AIME24 ↑	# Nodes	PPO Steps	Training Hours ↓
1.5B basemodel	29.3	-	-	-
w/ VeRL	**43.1***	16	250	33.6
w/ Sync.AReaL	42.0	16	250	41.0
w/ AReaL (ours)	42.2	16	250	**14.8**
7B basemodel	54.3	-	-	-
w/ VeRL	-	24	250	52.1
w/ Sync.AReaL	63.0	24	250	57.7
w/ AReaL (ours)	**63.1**	24	250	**25.4**

Model	LiveCodeBench ↑	# Nodes	PPO Steps	Training Hours ↓
14B basemodel	53.4	-	-	-
w/ VeRL	57.9*	32	80	44.4
w/ Sync.AReaL	56.7	32	80	48.8
w/ AReaL (ours)	**58.1**	32	80	**21.9**
32B basemodel	57.4	-	-	-
w/ VeRL	-	48	60	46.4
w/ Sync.AReaL	**61.2**	48	60	51.1
w/ AReaL (ours)	61.0	48	60	**31.1**

Results: Scalability

1. We compared the strong-scaling of AReaL against verl, a state-of-the-art synchronous system.
2. AReaL demonstrates nearly linear scaling as the number of GPUs increases.
3. The synchronous system fails to scale effectively, especially with longer context lengths.

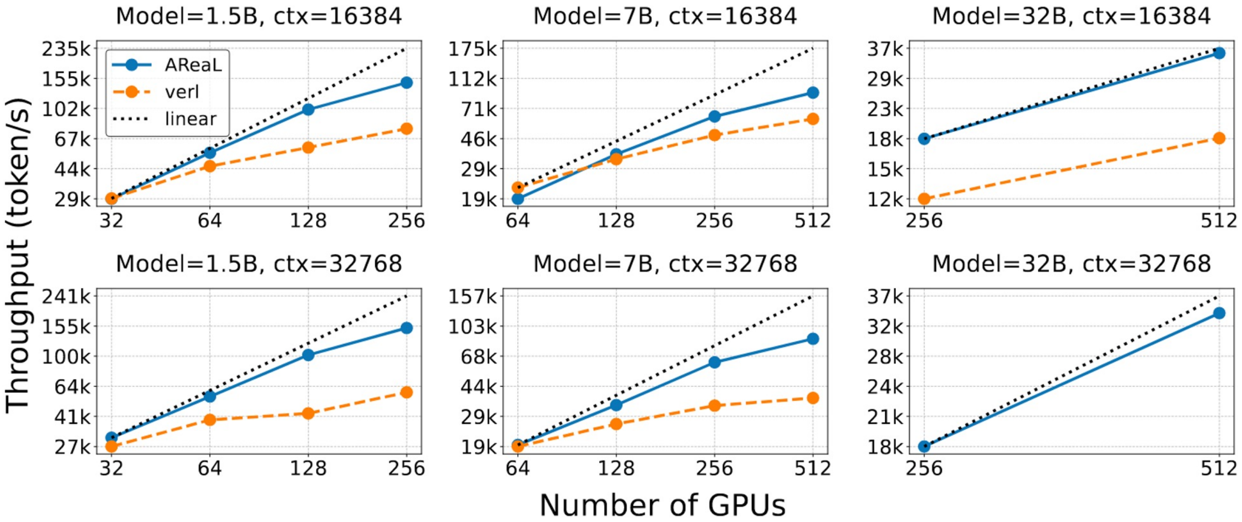
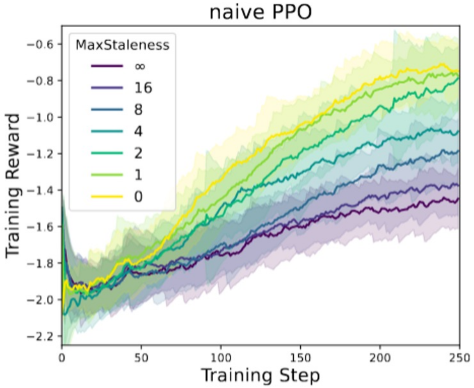


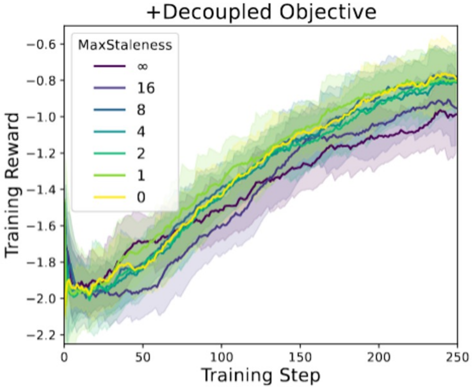
Figure 4: The strong scaling trend. Dotted lines indicate ideal linear scaling. verl consistently encounters OOM with 32k context length and the 32B model so the data points are missing.

Algorithm Ablation: Staleness and Decoupled PPO

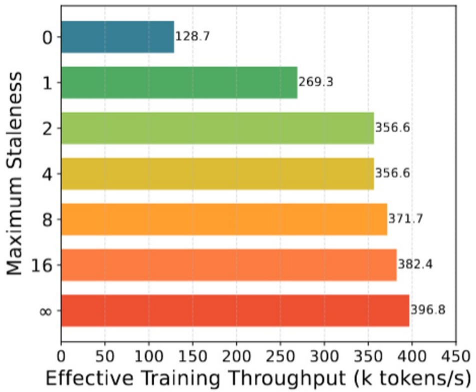
1. Naive PPO (left): Performance degrades significantly as data staleness (η) increases.
2. Decoupled PPO (center): The decoupled objective substantially improves training stability and performance, even with stale data.
3. Throughput (right): Allowing for moderate staleness dramatically increases effective training throughput.



(a) Learning curves with naive PPO.



(b) Learning curves with eq. (5).



(c) Effective training throughput.

Algorithm Ablation: Performance vs. Staleness

1. With the decoupled objective, a moderate maximum staleness (η=4 or η=8) achieves performance comparable to the synchronous "oracle" (η=0).
2. However, unbounded staleness still leads to inferior performance.
3. This validates our approach of combining controlled staleness with the decoupled PPO objective.

Max.Stale.	AIME24		AIME25		AMC23		MATH 500	
	W/o	With	W/o	With	W/o	With	W/o	With
0 (Oracle)	42.0		32.9		84.4		89.2	
1	41.8	42.1	30.7	31.9	83.3	85.2	89.9	89.8
2	40.0	41.8	32.1	32.5	82.3	84.3	89.6	89.6
4	23.3	42.2	23.1	32.0	58.5	85.1	66.9	89.5
8	35.7	41.0	27.8	31.1	81.2	82.9	87.8	89.2
16	35.8	38.7	26.2	32.5	78.4	83.2	87.4	89.1
∞	34.0	36.9	26.9	29.9	79.4	81.0	87.1	88.1

System Ablation: Interruptible Generation

1. We compared the throughput of our system with and without the interruptible generation feature.
2. Interruptible generation leads to a 12% throughput increase for the 1.5B model and a 17% increase for the 7B model.
3. This confirms that dynamically updating weights without waiting for slow responses to finish is a key architectural benefit.

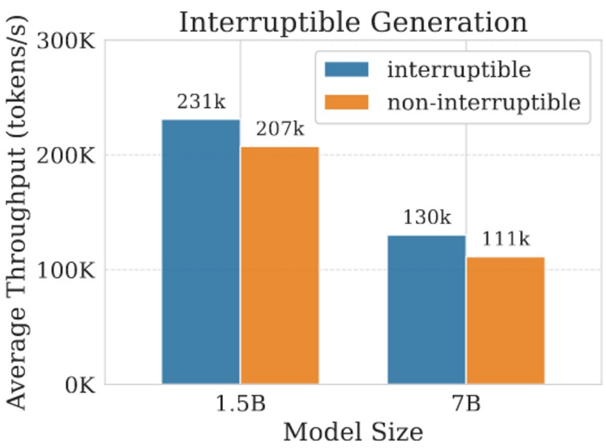


Figure 6: Ablation study of interruptible generation.

System Ablation: Dynamic Batching

1. We evaluated our dynamic micro-batch allocation algorithm against a standard strategy.
2. Our algorithm intelligently balances tokens across micro-batches to maximize GPU memory utilization and minimize padding.
3. Dynamic batching yields an average of 30% throughput improvement during PPO training across all tested model sizes.

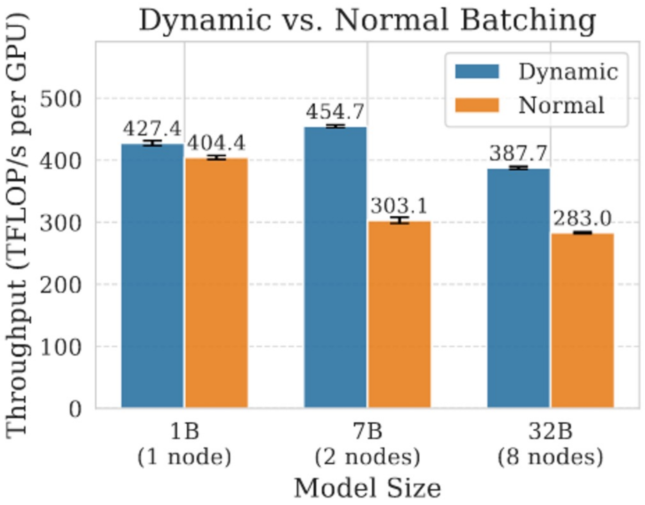


Figure 7: Ablation study of dynamic micro-batch allocation.

Conclusion

1. We introduced AReaL, a fully asynchronous system for large-scale RL training that is efficient, scalable, and stable.
2. By completely decoupling generation and training, AReaL achieves superior hardware utilization and up to a 2.77x training speedup.
3. Key Innovations:
   a. An expressive, asynchronous architecture with interruptible workers.
   b. Algorithmic enhancements—staleness-aware training and a decoupled PPO objective—that stabilize training with stale data.
4. This work provides a robust foundation for reliably scaling RL, enabling future advances in machine intelligence.