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] **Yi Wu** [1 2] **Jiashu Wang** [2 3]

Abstract

Effective RL for LLMs requires massive parallelization and poses an urgent need for efficient training systems. Most existing large-scale RL systems for LLMs are synchronous by alternating generation and training in a batch setting, where the rollouts in each training batch are generated by the same (or latest) model. This stabilizes RL training but suffers from severe system-level inefficiency. Generation must wait until the longest output in the batch is completed before model update, resulting in GPU underutilization. We present AReaL, a *fully asynchronous* RL system that completely decouples generation from training. Rollout workers in AReaL continuously generate new outputs without waiting, while training workers update the model whenever a batch of data is collected. To stabilize RL training, AReaL balances the workload of rollout and training workers to control data staleness, and adopts a staleness-enhanced PPO variant to better handle outdated training samples. Extensive experiments on math and code reasoning benchmarks show that AReaL achieves **up to 2.57× training speedup** compared to the best synchronous systems with the same number of GPUs and matched or even improved final performance.

1. Introduction

Reinforcement learning (RL) has been a new scaling paradigm for enhancing the capabilities of large language models (LLMs) by enabling thinking abilities (Wei et al., 2022). Effective RL training often requires massive parallelization to derive a large batch of rollouts for sufficient exploration, which is the key to obtaining the optimal model performance. For example, popular RL algorithms, such as PPO (Schulman et al., 2017) and GRPO(Shao et al., 2024), often require an effective training batch of thousands of outputs (Yu et al., 2025; Yue et al., 2025; Xin et al., 2024). Moreover, an LRM can generate tens of thousands of thinking tokens for each input prompt (DeepSeek-AI et al., 2025), further posing an urgent need for an efficient training system to run RL training on a large scale.

Most existing large-scale RL systems are designed in a fully synchronous manner (Mei et al., 2024; Hu et al., 2024; Sheng et al., 2025; Shen et al., 2024) by strictly alternating between LLM generation and training, which ensures that the LLM is always trained on the latest outputs for the best practical performance. In such a synchronous design, the generation step must wait until the finish of the longest output within a batch. Due to the varying output lengths for LRM, a synchronous RL system suffers from severe training inefficiency. Very recently, there have also been attempts to explore parallel generation and training (Noukhovitch et al., 2024; Luo et al., 2025a; Team et al., 2025). These works use outputs generated from a previous model version to update the current model. For the best performances, the model version used for rollout generation is limited to only one or two steps older. However, all these systems still follow a batched generation setting, where all the samples within a training batch are from the same model version. Accordingly, the issue of system inefficiency during the generation phase still remains unaddressed.

To fundamentally resolve the issues in system design, we develop AReaL, a fully Asynchronous RL training system for LRMs that completely decouples generation from training without hurting the final performance. AReaL runs LLM generation in a streaming manner, where each rollout worker continuously generates new outputs without waiting, leading to high GPU utilization. Meanwhile, the trainer workers in AReaL run parallel model updates whenever a training batch is obtained from the rollout workers. Once the model is updated, we synchronize the model weights in each rollout worker. In such an asynchronous design, each training batch of AReaL may contain samples generated by

[*]Equal contribution [1]IIIS, Tsinghua University [2]Ant Research [3]HKUST. Correspondence to: Wei Fu <fuwth17@gmail.com>, Yi Wu <jxwuyi@gmail.com>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

different model versions. Therefore, AReaL incorporates a modified version of the PPO algorithm, which can leverage samples generated from models of up to 8-step older without any performance drop. AReaL also conducts a data filtering process to ensure the staleness of each training sample is well controlled.

We evaluate AReaL on challenging mathematical reasoning and code generation tasks using models up to 32B parameters. Compared to state-of-the-art synchronous systems, AReaL achieves up to 2.57× higher training throughput and linear scaling efficiency up to 512 GPUs. Crucially, this acceleration *even comes with improved solution accuracy on these tasks*, illustrating that AReaL delivers significant efficiency gains without sacrificing (and indeed enhancing) model performance.

2. Motivation

We identify two limitations in synchronous RL systems:

Inference devices are underutilized. As shown in Figure 1 (left), generation must wait for the longest sequence to complete before training can begin. This leads to non-uniform decoding length across GPUs, which underutilizes GPU compute resources.

Scalability is poor in synchronous RL systems. synchronous systems distribute generation across all devices, reducing the per-GPU decoding batch size. This pushes the decoding process into a memory-IO-bound regime (Chen et al., 2024; Miao et al., 2024) where additional devices fail to improve throughput.

3. System Architecture of AReaL

3.1. System Overview

Figure 2 presents the architecture and data flow of AReaL. The system comprises 4 core components:

Interruptible Rollout Worker handles two types of requests: (1) The `generate` request generates responses given prompts. (2) The `update_weights` request interrupts all ongoing generations and loads parameters of new versions. We emphasize that such interruptions and in-flight weight updates would result in trajectories composed of segments produced by different model versions. This introduces a novel algorithmic challenge, which will be addressed in Section 4.

Reward Service evaluates the accuracy of the responses generated by the model. For example, in the coding task, this service runs unit tests to verify its accuracy.

Trainer Workers continuously sample from the replay buffer, accumulating data until reaching the configured training batch size. They then perform PPO updates and store the resulting parameters in distributed storage. To ensure data freshness, data from the replay buffer is used only once.

Rollout Controller serves as a critical bridge between the rollout workers, reward service, and the model workers. During the training process, it reads data from the dataset and invokes the rollout worker's `generate` request. The received response is then sent to the reward service to obtain the reward. The trajectory, along with the reward, is stored in the replay buffer, waiting to be trained by the model worker. After the model worker updates the parameters, the controller calls the rollout worker's `update_weight`. We illustrate the generation and training management in Figure 3. This asynchronous pipeline ensures continuous full utilization of both generation and training resources.

3.2. Algorithmic Challenges

Data Staleness Due to the asynchronous nature of AReaL, each training batch contains data from multiple prior policy versions. Data staleness would lead to a distribution gap between the training data and the latest model. In asynchronous RL training for LRMs, this issue could be even more severe for long trajectories due to extended decoding time.

Inconsistent Policy Versions As discussed in Sec. 3.1, the generated trajectories may involve segments produced by different policy versions. This inconsistency fundamentally violates the formulation of standard PPO that assumes all actions being generated by a single policy π_{old}.

4. Addressing the Algorithmic Challenges

4.1. Staleness-Aware Training

We introduce a hyperparameter η representing *the maximum permitted staleness*. Given the latest parameter version i, total generated trajectories N_r, and training batch size B, we enforce:

$$\lfloor N_r/B \rfloor \leq i + \eta. \tag{1}$$

When $\eta = 1$, the system recovers to the previous one-step overlap methods (Noukhovitch et al., 2024; Luo et al., 2025a). While this approach guarantees bounded staleness, overly conservative η values can unnecessarily throttle generation throughput—particularly for long-context generations where the completion time of a batch varies significantly. This motivates our adoption of a decoupled PPO objective that can make efficient use of slightly staled data.

4.2. Decoupled PPO Objective

We apply a decoupled PPO objective (Hilton et al., 2022) that disentangles the *behavior policy* and the *proximal policy*. The behavior policy π_{behav} represents the policy used for sampling trajectories and the proxy policy π_{prox} is a proximal policy serving as a recent target to regularize the

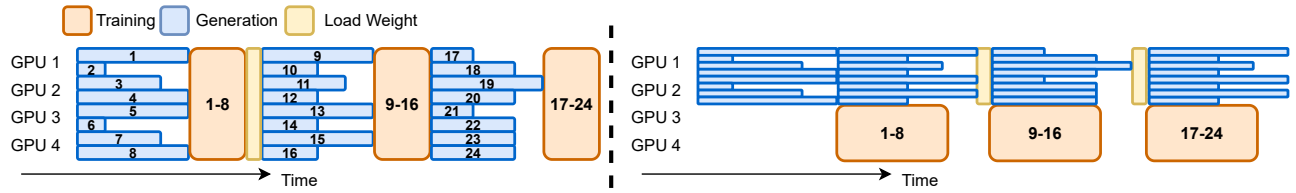


Figure 1: Execution timeline of a synchronous (left) and an one-step overlap (right) RL system showing underutilized inference devices.

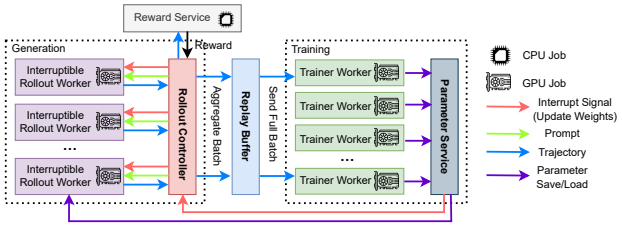


Figure 2: The AReaL architecture featuring asynchronous generation and training components.

update of π_θ. By applying importance sampling on the sampled trajectories, we could derive a decoupled PPO objective suitable for asynchronous RL training,

$$J(\theta) = \mathbb{E}\left[\sum_{t=1}^{H} \min\Big(\underbrace{\frac{\pi_\theta}{\pi_{\text{behav}}}}_{\text{Importance Ratio}}\hat{A}_t,\ \overbrace{\frac{\pi_{\text{prox}}}{\pi_{\text{behav}}}}^{\text{Importance Ratio}}\text{clip}\Big(\underbrace{\frac{\pi_\theta}{\pi_{\text{prox}}}}_{\text{Trust Region Center}}\Big)\hat{A}_t\Big)\right] \tag{2}$$

$$= \mathbb{E}\left[\sum_{t=1}^{H} \frac{\pi_{\text{prox}}}{\pi_{\text{behav}}} \min\left(u_t^{\text{prox}}(\theta)\hat{A}_t, \text{clip}\left(u_t^{\text{prox}}(\theta)\right)\hat{A}_t\right)\right], \tag{3}$$

where $u_t^{\text{prox}}(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{prox}}(a_t|s_t)}$ is the importance ratio w.r.t. the proximal policy. The main difference between the asynchronous PPO objective Equation (3) and the standard one lies in the proximal policy π_{prox} for regularizing the model update. In asynchronous PPO training, using the behavior policy as the proximal policy will pull the latest policy π_θ towards the old-version and low-quality policies, thus slowing down model improvements. By employing a recent policy as the proximal policy, model updates would happen within the trust region around the high-quality proximal policy π_{prox}, thus stabilizing training.

Equation (3) provides a natural benefit: it relaxes the requirement that all data within one training batch should be generated with a singe policy. This property is crucial for maintaining algorithmic correctness when combining interruptible generation with policy updates.

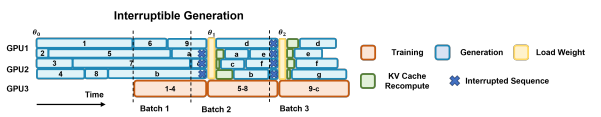


Figure 3: Illustration of generation management in AReaL. Vertical lines shows the ready time for the next step training. Blue crosses show the interrupted requests when new parameters arrive.

Proposition 4.1. *For any sequence $(q, a_1, \ldots, a_H)$ generated by policies $(\pi_\theta, \ldots, \pi_{\theta+k})$ where $\pi_{\theta+i}$ produces tokens $(a_{t_i}, \ldots, a_{t_{i+1}})$, where $1 = t_0 < \cdots < t_{k+1} = H$, there exists a behavior policy π_{behav} such that the interrupted generation is equivalent to sampling entirely from π_{behav}.*

Practical Remark While Hilton et al. (2022) maintains an exponential moving average of parameters for π_{prox}, this approach is prohibitively expensive for LRMs. Consequently, we simply use the parameters before each model update step as π_{prox}, i.e., recomputing token probabilities upon the arrival of the global batch in each training step.

5. Experiments

Our evaluation comprises three components: (1) comprehensive comparisons against state-of-the-art open-source frameworks across model sizes, (2) strong-scaling analysis with varying compute resources, and (3) ablation studies validating our design choices. Experiment settings can be found in Appendix B.

5.1. End-to-End Comparison

We establish two state-of-the-art baselines using synchronous RL systems: DeepScaleR (Luo et al., 2025b) for mathematical reasoning with a 1.5B model, and DeepCoder (Luo et al., 2025a) for code generation with a 14B model, both trained using verl (Sheng et al., 2025). For larger 7B and 32B models where comparable baselines are unavailable, we performed controlled experiments by training from scratch using a synchronous variant of AReaL. Our main results are shown in Table 1. Since the code of obtaining previous SOTA models can be out-of-date, we measure the throughput and estimate the training hours using the latest verl code for a fair comparison. AReaL consistently matches or exceeds baseline performance while achieving significant speedups. In particular, our system demonstrates up to **2.8× improvement in training throughput** compared to synchronous approaches without significant performance degradation.

Table 1: End-to-End Performance Comparison. We evaluate on the AIME24 benchmark for math and LiveCodeBench (8/1/24-2/1/25) for coding. We limit the maximum generation length to 32K tokens and sample 32 responses per question, reporting the average pass@1 accuracy. * represents the best known reproducible results obtained via RL , as cited from DeepScaler (Luo et al., 2025b) and DeepCoder (Luo et al., 2025a) respectively. AReaL achieve the comparable performance with 2x less training hours.

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**

5.2. Scalability

We compare the scalability of AReaL with verl (Sheng et al., 2025), the state-of-the-art synchronous RL system, across different model sizes and context lengths. We measure the *effective throughput* for training, defined as the rate of consuming generated tokens during PPO updates, after proper warmup steps. Figure 4 presents the results for context lengths of 16k and 32k. Across all settings, AReaL demonstrates an approximate linear scaling trend with increased device count, while the synchronous system typically fails to scale effectively. Additionally, AReaL is more robust with longer generation lengths due to asynchronous and interruptible generation. The generation of long responses can be fully hidden in the critical path, so extending generation length does not drastically affect the effective training throughput of AReaL.

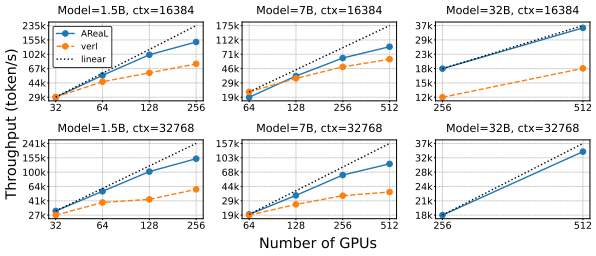
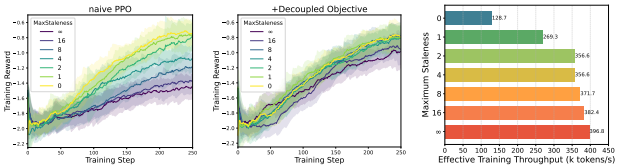


Figure 4: The strong scaling trend. Dotted lines indicate ideal linear scaling. verl consistently encounters OOM with 32k context length and the 32B model so the data points are missing.



(a) Learning curves with naive PPO. (b) Learning curves with Equation (3). (c) Effective training throughput.

Figure 5: Ablation studies of the decoupled PPO objective and staleness control. Both algorithmic choices are essential. With a moderate staleness value and the decoupled objective, training progress can be accelerated by 2x while maintaining final evaluation performance.

Table 2: Evaluation scores when varying data staleness, comparing performance with and without the decoupled objective. Numbers within ±1 of the oracle score are underlined.

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

5.3. Algorithm Ablations

We conduct ablation studies to validate our algorithmic innovations in Section 4 by training a 1.5B LRM on math tasks. We follow the basic experiment setting of DeepScaleR and then gradually increase the η value for ablation purposes. Specifically, we vary the maximum allowed staleness η and compare configurations with and without the decoupled PPO objective. Figures 5a and 5b show the learning curves after 250 training steps. Table 2 presents the corresponding final evaluation performances across multiple mathematical reasoning benchmarks. We follow the common practice of PPO and perform multiple mini-batch updates within each training step. We emphasize that η constrains the training batch staleness regarding training steps.

6. Conclusion

This paper introduces AReaL, a fully asynchronous system designed for efficient large-scale reinforcement learning (RL) training. We contribute several algorithmic innovations, including staleness-aware training and a decoupled PPO objective, which enable efficient and stable PPO training in asynchronous environments. Our experimental results demonstrate AReaL's superior hardware efficiency, sample efficiency, and scalability compared to existing synchronous RL systems.

References

Chen, Z., May, A., Svirschevski, R., Huang, Y., Ryabinin, M., Jia, Z., and Chen, B. Sequoia: Scal-

able and robust speculative decoding. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 129531–129563. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ea1f5f0878d43ff4fb8bf64ef4a2326c-Paper-Conference.pdf.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., and Li, S. S. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025. doi: 10.48550/ARXIV.2501.12948. URL https://doi.org/10.48550/arXiv.2501.12948.

Hilton, J., Cobbe, K., and Schulman, J. Batch size-invariance for policy optimization. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/6ceb6c2150bbf46fd75528a6cd6be793-Abstract-Conference.html.

Hu, J., Wu, X., Wang, W., Xianyu, Zhang, D., and Cao, Y. Openrlhf: An easy-to-use, scalable and high-performance RLHF framework. *CoRR*, abs/2405.11143, 2024. doi: 10.48550/ARXIV.2405.11143. URL https://doi.org/10.48550/arXiv.2405.11143.

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Dang, K., Yang, A., Men, R., Huang, F., Ren, X., Ren, X., Zhou, J., and Lin, J. Qwen2.5-coder technical report. *CoRR*, abs/2409.12186, 2024. doi: 10.48550/ARXIV.2409.12186. URL https://doi.org/10.48550/arXiv.2409.12186.

Jain, N., Han, K., Gu, A., Li, W., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In Flinn, J., Seltzer, M. I., Druschel, P., Kaufmann, A., and Mace, J. (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Luo, M., Tan, S., Huang, R., Patel, A., Ariyak, A., Wu, Q., Shi, X., Xin, R., Cai, C., Weber, M., et al. Deepcoder: A fully open-source 14b coder at o3-mini level, 2025a.

Luo, M., Tan, S., Wong, J., Shi, X., Tang, W. Y., Roongta, M., Cai, C., Luo, J., Li, L. E., Popa, R. A., and Stoica, I. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Mode 2025b. Notion Blog.

Mei, Z., Fu, W., Li, K., Wang, G., Zhang, H., and Wu, Y. Realhf: Optimized RLHF training for large language models through parameter reallocation. *CoRR*, abs/2406.14088, 2024. doi: 10.48550/ARXIV.2406.14088. URL https://doi.org/10.48550/arXiv.2406.14088.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., Shi, C., Chen, Z., Arfeen, D., Abhyankar, R., and Jia, Z. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, pp. 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

Noukhovitch, M., Huang, S., Xhonneux, S., Hosseini, A., Agarwal, R., and Courville, A. C. Asynchronous RLHF: faster and more efficient off-policy RL for language models. *CoRR*, abs/2410.18252, 2024. doi: 10.48550/ARXIV.2410.18252. URL https://doi.org/10.48550/arXiv.2410.18252.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Shen, G., Wang, Z., Delalleau, O., Zeng, J., Dong, Y., Egert, D., Sun, S., Zhang, J. J., Jain, S., Taghibakhshi, A., Ausin, M. S., Aithal, A., and Kuchaiev, O. Nemo-aligner: Scalable toolkit for efficient model alignment. *CoRR*, abs/2405.01481, 2024. doi: 10.48550/ARXIV.2405.01481. URL https://doi.org/10.48550/arXiv.2405.01481.

Sheng, G., Zhang, C., Ye, Z., Wu, X., Zhang, W., Zhang, R., Peng, Y., Lin, H., and Wu, C. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019. URL http://arxiv.org/abs/1909.08053.

Team, P. I., Jaghouar, S., Mattern, J., Ong, J. M., Straube, J., Basra, M., Pazdera, A., Thaman, K., Ferrante, M. D., Gabriel, F., Obeid, F., Erdem, K., Keiblinger, M., and Hagemann, J. Intellect-2: A reasoning model trained through globally decentralized reinforcement learning, 2025. URL https://arxiv.org/abs/2505.07291.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Confere
html.

Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., and Liang, X. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *CoRR*, abs/2405.14333, 2024. doi: 10.48550/ARXIV.2405.14333. URL https://doi.org/10.48550/arXiv.2405.14333.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Yang, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao, R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Liu, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., Guo, Z., and Fan, Z. Qwen2 technical report. *CoRR*, abs/2407.10671, 2024a. doi: 10.48550/ARXIV.2407.10671. URL https://doi.org/10.48550/arXiv.2407.10671.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024b. doi: 10.48550/ARXIV.2412.15115. URL https://doi.org/10.48550/arXiv.2412.15115.

Yang, A., Zhang, B., Hui, B., Gao, B., Yu, B., Li, C., Liu, D., Tu, J., Zhou, J., Lin, J., Lu, K., Xue, M., Lin, R., Liu, T., Ren, X., and Zhang, Z. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *CoRR*, abs/2409.12122, 2024c. doi: 10.48550/ARXIV.2409.12122. URL https://doi.org/10.48550/arXiv.2409.12122.

Yoo, A. B., Jette, M. A., and Grondona, M. SLURM: simple linux utility for resource management. In Feitelson, D. G., Rudolph, L., and Schwiegelshohn, U. (eds.), *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*, volume 2862 of *Lecture Notes in Computer Science*, pp. 44–60. Springer, 2003. doi: 10.1007/10968987_3. URL https://doi.org/10.1007/10968987_3.

Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Fan, T., Liu, G., Liu, L., Liu, X., Lin, H., Lin, Z., Ma, B., Sheng, G., Tong, Y., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhu, J., Chen, J., Chen, J., Wang, C., Yu, H., Dai, W., Song, Y., Wei, X., Zhou, H., Liu, J., Ma, W., Zhang, Y., Yan, L., Qiao, M., Wu, Y., and Wang, M. DAPO: an open-source LLM reinforcement learning system at scale. *CoRR*, abs/2503.14476, 2025. doi: 10.48550/ARXIV.2503.14476. URL https://doi.org/10.48550/arXiv.2503.14476.

Yue, Y., Yuan, Y., Yu, Q., Zuo, X., Zhu, R., Xu, W., Chen, J., Wang, C., Fan, T., Du, Z., Wei, X., Yu, X., Liu, G., Liu, J., Liu, L., Lin, H., Lin, Z., Ma, B., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhang, R., Liu, X., Wang, M., Wu, Y., and Yan, L. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks, 2025. URL https://arxiv.org/abs/2504.05118.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., Desmaison, A., Balioglu, C., Damania, P., Nguyen, B., Chauhan, G., Hao, Y., Mathews, A., and Li, S. Pytorch FSDP: experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16(12):3848–3860, 2023. doi: 10.14778/3611540.3611569. URL https://www.vldb.org/pvldb/vol16/p3848-huang.pdf.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C. W., and Sheng, Y. Sglang: Efficient execution of structured language model programs. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

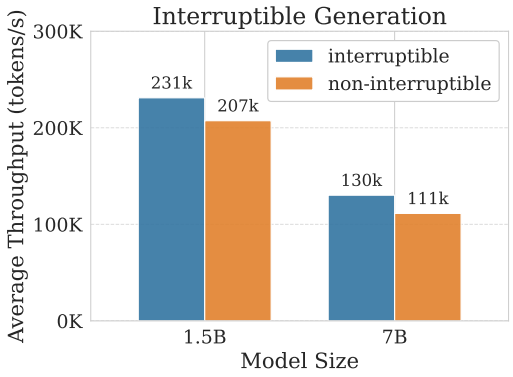


Figure 6: Ablation study of interruptible generation.

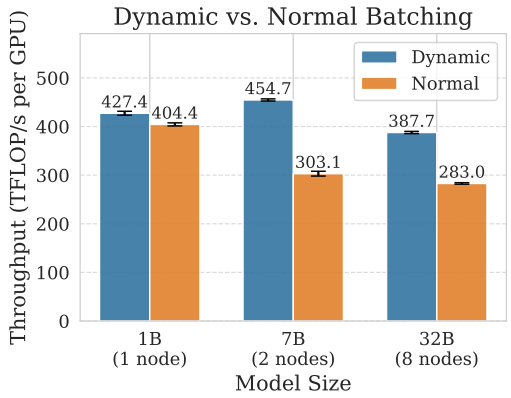


Figure 7: Ablation study of dynamic micro-batch allocation.

A. Additional Results

A.1. System Ablations

Interruptible Generation We ablate interruptible generation and present the resulting generation throughput in Figure 6. Without interruptible generation, the controller must wait for the longest response. In particular, interruptible generation leads to a 12% and 17% throughput increase for 1.5B and 7B models respectively on 4 nodes, which validates our architectural design choice.

Dynamic Microbatch Allocation We investigate the effectiveness of dynamic batching by comparing PPO training throughput against a standard micro-batching strategy. The standard micro-batching strategy can result in multiple long sequences being assigned to the same micro-batch, thus usually requiring a sufficiently large number of micro-batches to prevent out-of-memory errors. In our experimental setup, we configured 32 micro-batches for the standard setting and established a token budget of 32,768 per micro-batch for the dynamic batching approach. As demonstrated in Figure 7, dynamic batching yields an average of 30% throughput improvements across various model sizes.

A.2. Additional Evaluation

We evaluate the models trained with AReaL on more math benchmarks, and list the results in Table 3.

Table 3: Results on math benchmarks.

Model	AIME24	AIME25	AMC23	MATH 500
1.5B basemodel	29.3	24.4	71.0	84.3
w/ Sync. AReaL	42.0	32.9	84.4	89.2
w/ AReaL	42.2	32.0	85.1	89.5
7B basemodel	54.3	41.7	89.5	92.8
w/ Sync. AReaL	63.0	50.0	93.2	94.2
w/ AReaL	63.1	47.3	93.6	94.3

B. Implementation Details

C. Overview

We implement AReaL using Python and PyTorch (Paszke et al., 2019) built upon the ReaLHF (Mei et al., 2024) framework. Our system combines SGLang (Zheng et al., 2024) v0.4.6 for generation serving with Megatron-Core (Shoeybi et al., 2019) v0.11.0 as the training backend, managed by SLURM (Yoo et al., 2003) for resource scheduling. To maximize throughput for both generation and training phases, we implement several key system-level optimizations that address critical bottlenecks in the pipeline.

D. System Optimizations

AReaL decouples GPU computation from CPU operations, including rule-based reward computation (such as string matching for math problems or unit test execution for code) and TCP-based data transfer. By executing these operations in separate threads and pipelining the workflow, we overlap reward computation and data transfer with subsequent generation requests. We use asyncio coroutines to concurrently runs multiple requests in the rollout worker to avoid mutual blocking waits.

To handle the training with variable-length sequences, we employ a padding-free sequence packing strategy coupled with a dynamic allocation algorithm. The algorithm balances token distribution across micro-batches under fixed memory constraints (see Algorithm 1). This approach maximizes GPU memory utilization while minimizing the number of required forward-backward passes.

D.1. Experiment Setup

We evaluate AReaL on challenging math and coding tasks. We employ the distilled Qwen2 model series (Yang et al., 2024a;b) from DeepSeek-R1 (DeepSeek-AI et al., 2025) as base models (i.e., R1-Distilled-Qwen), spanning from 1.5B to 32B parameters. For each task-model combination, we train for a fixed number of PPO updates and evaluate the final checkpoint. Our evaluation on mathematical tasks follow the Qwen evaluation protocol (Yang et al., 2024c; Hui et al., 2024), while coding models are assessed on LiveCodeBench (8/1/24-2/1/25) (Jain et al., 2025) using the official protocol. Unless otherwise specified, we set the maximum staleness $\eta = 4$ and adopt the training configurations used in Section 5.1, with additional hyperparameters detailed in Appendix B.

We conduct experiments on an H800 GPU cluster comprising 64 nodes, each equipped with 8 GPUs. The cluster features NVLink for intra-node connectivity and RoCE with 3.2Tbps bandwidth for inter-node communication. To ensure rapid convergence, we allocate a minimum of 16 nodes as a baseline pod configuration for complete experiments. We scale the number of nodes proportionally with model size, ultimately utilizing 48 nodes for training our largest 32B parameter model. This scaling strategy enables us to run experiments of varying sizes in parallel while maintaining efficient resource utilization.

For AReaL, we maintain a fixed ratio between inference and training devices, allocating three-quarters of the devices for inference. This configuration was selected against an equal 50-50 partition based on our early experiments, where the 75-25 partition demonstrated higher training throughput. While we adopt this ratio as a heuristic configuration, we emphasize that the optimal partition may vary across different settings and could potentially benefit from dynamic adjustment during

training, as discussed in Section 6.

D.2. PPO Details

We disable the critic model and the reference model in PPO. The advantage estimation parameter λ in GAE and the RL discount factor γ are fixed at 1. The reward is 5 at the final token if the answer is correct and -5 otherwise. We additionally adopt advantage normalization across the global batch to stabilize the training. Other learning related hyperparameters and configurations can be found in Table 4.

Table 4: Training configurations and hyperparameters.

Training Configuration	
Batch size (number of prompts)	512
Random seed	1
PPO Parameters	
PPO Minibatches	4
Clipping ϵ	0.2
Advantage normalization	True
Discount factor γ	1.0
GAE λ	1.0
Optimizer Parameters	
Optimizer	Adam
Learning rate	2.0×10^{-5}
Weight decay	0.05
β_1	0.9
β_2	0.95
Adam ϵ	1×10^{-5}
Gradient norm clipping	1.0
Learning rate scheduler	constant
Warmup steps proportion	0.001
Precision Parameters	
Parameter dtype	fp16
KV cache dtype	fp16
Gradient dtype	fp32
Optimizer state dtype	fp32
Generation Parameters	
Answers per prompt	16
Temperature	1.0
Top-p	1.0
Top-k	-1
Max prompt length	1024
Min generation length	0
Max generation length	27648

D.3. Dataset Details

For the math task, we use the open-source data from DeepScaleR (Luo et al., 2025b), For code training, we used the dataset released by DeepCoder (Luo et al., 2025a). All compared methods use the same dataset.

D.4. Dynamic Batching

Algorithm 1 Dynamic Batching

Require: Sequence lengths $S = \{s_1, s_2, \ldots, s_n\}$, maximum micro-batch capacity C, minimum number of micro-batches k_{min}
Ensure: Balanced partition of sequences into micro-batches with total length $\leq C$
  Sort S in descending order
  batches $\leftarrow \emptyset$
  for $s \in S$ **do**
    if |batches| $< k_{min}$ **or** no existing batch can fit s **then**
      Create new micro-batch containing sequence i
      batches.append($\{s\}$)
    else
      Find all batches that can accommodate s
      Select the micro-batch with fewest sequences
    end if
  end for=0

The dynamic batching algorithm is shown in Algorithm 1.

D.5. Baselines

In our experiments, we use the lastest version (main branch of verl repository, May 7, 2025) of verl (Sheng et al., 2025) to evaluate the training throughput in Figure 4 and the training hours in Table 1. For most of the results, we use SGLang (Zheng et al., 2024) v0.4.6 as generation backend and pytorch FSDP (Zhao et al., 2023) as training backend. In a few cases where SGLang raises errors (experiments with 32B models or 64 nodes), we use vLLM (Kwon et al., 2023) v0.8.4 as a substitution.

E. Proof of Proposition 1

Proposition 1. *For any sequence $(q, a_1, \ldots, a_H)$ generated by policies $(\pi_\theta, \ldots, \pi_{\theta+k})$ where $\pi_{\theta+i}$ produces tokens $(a_{t_i}, \ldots, a_{t_{i+1}})$, where $1 = t_0 < \cdots < t_{k+1} = H$, there exists a behavior policy π_{behav} such that the interrupted generation is equivalent to sampling entirely from π_{behav}.*

Proof. For question q, let $\mathcal{S}_t(q)$ denote states encountered at step t by the sequence of policies. Since $\mathcal{S}_{t_i}(q) \cap \mathcal{S}_{t_j}(q) = \emptyset$ for $i \neq j$, we can construct:

$$\pi_{\text{behav}}(\cdot|s) = \begin{cases} \pi_{\theta+j}(\cdot|s) & \text{if } t_j \leq t \leq t_{j+1} \text{ and } s \in \mathcal{S}_t(q) \\ \text{arbitrary} & \text{otherwise} \end{cases}$$

□

F. Limitations and Future Work

Our work presents several limitations that suggest directions for future research. First, the ratio between inference and training devices could be further optimized for specific training setups. Additionally, this ratio might benefit from dynamic adjustment during training, particularly as context lengths typically increase when fine-tuning pre-trained base models. While we focused our evaluation on single-step mathematical and coding tasks, the AReaL architecture is not inherently limited to these domains. We leave the exploration of multi-turn interactions and agentic scenarios to future work.