Staggered Environment Resets Improve Massively Parallel On-Policy Reinforcement Learning

Sid Bharthulwar[1], **Stone Tao**[2], **Hao Su**[2]
sbharthulwar@college.harvard.edu, {stao,haosu}@ucsd.edu

[1]**Department of Computer Science, Harvard University, USA**
[2]**Department of Computer Science, University of California at San Diego, USA**

Abstract

Massively parallel GPU simulation environments have accelerated reinforcement learning (RL) research by enabling fast data collection for on-policy RL algorithms like Proximal Policy Optimization (PPO). To maximize throughput, it is common to use short rollouts per policy update, increasing the update-to-data (UTD) ratio. However, we find that, in this setting, standard synchronous resets introduce harmful nonstationarity, skewing the learning signal and destabilizing training. We introduce staggered resets, a simple yet effective technique where environments are initialized and reset at varied points within the task horizon. This yields training batches with greater temporal diversity, reducing the nonstationarity induced by synchronized rollouts. We characterize dimensions along which RL environments can benefit significantly from staggered resets through illustrative toy environments. We then apply this technique to challenging high-dimensional robotics environments, achieving significantly higher sample efficiency, faster wall-clock convergence, and stronger final performance. Finally, this technique scales better with more parallel environments compared to naive synchronized rollouts.

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful paradigm for tackling complex sequential decision-making problems, particularly in continuous control domains like robotics (Kober et al., 2013; Levine et al., 2016; Lee et al., 2019). However, RL often depends on vast quantities of interaction data, a requirement that can be prohibitively expensive or slow to acquire in real-world settings. Massively parallel simulation environments, especially those accelerated on GPUs (Makoviychuk et al., 2021; Freeman et al., 2021; Nikulin et al., 2023; Mittal et al., 2023), have enabled data-generation throughput on orders of magnitude greater than traditional CPU-based setups. The increase in data throughput has enabled far faster training of robotics models with successful sim2real deployments of locomotion (Rudin et al., 2022; Margolis et al., 2022), state-based manipulation (Handa et al., 2022; Lin et al., 2025), and vision-based manipulation (Tao et al., 2025; Zakka et al., 2025; Singh et al., 2025).

Despite this paradigm shift in data generation capabilities, the core algorithms, particularly on-policy methods like Proximal Policy Optimization (PPO) (Schulman et al., 2017), have often been adapted with only superficial changes—typically larger batch sizes and shorter per-environment rollouts (K) to increase the update-to-data (UTD) ratio (Rudin et al., 2022). This strategy, while seemingly maximizing hardware utilization, overlooks a critical interaction between the data collection process and the learning algorithm's stability when the task horizon (H) significantly exceeds the rollout length ($K \ll H$).

In this work, we identify a subtle yet pervasive issue: cyclical batch nonstationarity. When thousands of environments are simulated synchronously and reset only after completing a full H-step episode, the data batches fed to the RL agent become temporally homogeneous but cycle predictably through different segments of the episode. One PPO update might see data exclusively from early-episode states, the next from mid-episode, and so on, before abruptly returning to early-episode data after a mass reset. This rapid, cyclical shift in the input data distribution can destabilize value function learning, induce policy oscillations, and hinder the agent's ability to consolidate knowledge across the entire task horizon. Essentially, the high-throughput data stream becomes a "moving target" for the learner, undermining the benefits of parallelization.

We argue that stable and efficient learning in this massively parallel regime requires more than just algorithmic re-tuning; it necessitates a modification to the environment interaction protocol itself. We introduce **staggered resets**, a simple yet highly effective technique that breaks this harmful synchronicity. By initializing parallel environments at diverse effective time steps distributed across the task horizon H, staggered resets ensure that each training batch contains a rich, temporally heterogeneous mix of experiences. This provides the learner with a more stationary and representative view of the overall task dynamics within every gradient update. To summarize our contributions:

- We precisely identify and articulate the problem of cyclical batch nonstationarity stemming from synchronous full-episode resets combined with short rollouts ($K \ll H$) in massively parallel on-policy RL, explaining its detrimental impact on learning dynamics.
- We propose staggered resets, an elegant and easily implementable mechanism independent of the RL algorithm itself to ensure temporal diversity within training batches by desynchronizing the effective starting points of parallel environments across the task horizon.
- Through illustrative toy environments, we characterize the conditions under which this nonstationarity is most severe and staggered resets offer maximal benefit.
- We provide compelling empirical evidence on challenging, high-dimensional robotics tasks, demonstrating that staggered resets significantly improve sample efficiency, wall-clock convergence speed, final policy performance, and scalability with increasing parallelism compared to standard synchronous reset protocols.

2 Related Work

Massively Parallel RL The inherent sample inefficiency of many reinforcement learning algorithms has significant research into scaling and parallelization to improve training time and performance. Early approaches, such as IMPALA and others (Mnih et al., 2016; Espeholt et al., 2018; Nair et al., 2015; Horgan et al., 2018), utilized multiple CPU workers to achieve parallelism and scalability. These typically relied on benchmarks (Yu et al., 2019; Gu et al., 2023; Tunyasuvunakool et al., 2020) built on top of CPU-based simulators like MuJoCo (Todorov et al., 2012), PyBullet (Coumans & Bai, 2016–2021), PhysX etc. More recently, GPU-accelerated simulators (Makoviychuk et al., 2021; Freeman et al., 2021) and other JAX-based GPU-accelerated environments (Nikulin et al., 2023; Lange, 2022) have enabled a much greater degree of parallelism, resulting in considerable speedups for training complex policies. Recent work has sought to replace the popular choice of PPO as an RL algorithm by improving scalability (Li et al., 2023; Singla et al., 2024). In contrast our work is algorithm-agnostic and addresses the challenge of handling non-stationary data in synchronous highly-parallel regimes for on-policy algorithms like PPO.

Nonstationarity in RL Nonstationarity in the data distribution is a recognized challenge within reinforcement learning. Such nonstationarity can stem from various sources, including changes in environment dynamics, the reward function, or, as pertinent to our work, the data collection process itself. Previous research has explored related issues such as catastrophic forgetting in continual learning scenarios (Kirkpatrick et al., 2017) and representation collapse arising from biased data. The "primacy bias," wherein early experiences exert a disproportionate influence on RL training, has also been documented (Nikishin et al., 2022; Schwarzer et al., 2023). Our work specifically

highlights and aims to mitigate the cyclical nonstationarity induced by the interplay of synchronous resets and short rollouts in massively parallel RL settings.

Short Rollouts in Parallel RL The use of short rollouts ($K \ll H$) in parallel RL is frequently motivated by the desire to increase update frequency (achieving a high update-to-data ratio) and maximize wall-clock training speed (Xu et al., 2021; Li et al., 2023). While this strategy can be effective, underlying issues related to data distribution bias are often overlooked. Some implementations incorporate partial resets—resetting environments upon task success, failure, or termination—which can introduce some eventual desynchronization. However, this process can be slow and may prove insufficient to counteract the initial bias, particularly when K is very small. Furthermore, while some simulation environment implementations include versions of the reset staggering technique we propose (Rudin et al., 2022), these are typically not detailed in accompanying publications. To our knowledge, our work is the first to thoroughly investigate and analyze this method, as well as to characterize the environmental conditions under which it proves most effective.

3 Methodology

3.1 Cyclical Nonstationarity in Massively Parallel PPO

We consider a standard MDP setting where the goal is to learn a policy π_θ to maximize the expected return over a horizon H. On-policy algorithms like PPO (Schulman et al., 2017) are often trained in a massively parallel setup with N synchronous environments. Data is collected for K steps (the rollout length) from all environments, creating a batch of $N \times K$ transitions which is then used for policy and value function updates.

In massively parallel RL, a common strategy to increase the update-to-data (UTD) ratio is to use a short rollout length K much smaller than the task horizon H ($K \ll H$). When combined with standard synchronous resets (where all environments are reset together at the start and after completing an episode), this creates a harmful cyclical nonstationarity in the training data. As illustrated in Figure 1(a), all environments start at $t = 0$. The first training batch thus contains data exclusively from the time window $[0, K-1]$. The next batch contains data from $[K, 2K-1]$, and so on. After approximately H/K updates, all environments complete their episodes and are reset synchronously, causing the next batch to abruptly revert to containing data only from $[0, K-1]$.

This process feeds the learner a data stream where the state distribution shifts predictably and dramatically from one batch to the next. The learner is forced to chase a "moving target," which can destabilize value function learning, hinder credit assignment over long horizons, and undermine the benefits of parallelization.

To counteract this detrimental cyclical nonstationarity, we introduce **staggered resets**. The core principle is to desynchronize the parallel environments such that each training batch contains experiences from diverse segments of the task horizon, as illustrated in Figure 1(b).

The mechanism is implemented efficiently. Before training begins, the N parallel environments are partitioned into N_B distinct groups. Each group is advanced for a specific number of offset steps, t_{offset}, using random actions or the initial policy. To make this process efficient, especially in GPU-based simulators where individual reset operations can be costly, these offsets are chosen from a discrete set. A practical choice is to set the offset for group $i \in \{0, \dots, N_B - 1\}$ to be $i \cdot K$, where K is the PPO rollout length and $N_B \approx H/K$. This pre-initialization phase positions environments at different effective starting points. Consequently, when the standard synchronous PPO data collection proceeds for K steps, the aggregated batch naturally contains a temporally heterogeneous mix of transitions from across the task horizon.

Reset Management During Training: To maintain this temporal diversity while preserving computational efficiency, we handle resets as follows:

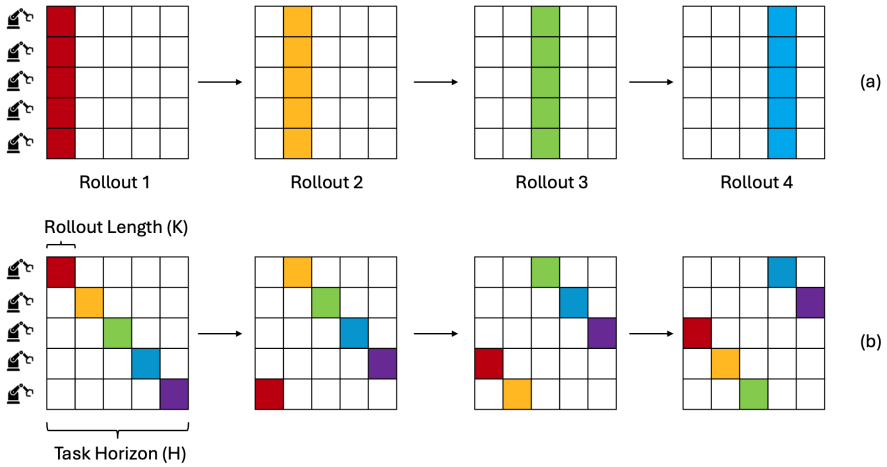

Figure 1: Data collection in massively parallel RL. Rows are environments, columns are time within task horizon H. Colors (red, orange, etc.) mark distinct task stages. (a) **Synchronous Resets (Naive)**: All environments start at $t = 0$ (red stage). Each rollout batch is temporally homogeneous. Batch content cycles, causing nonstationarity. (b) **Staggered Resets**: Environments start at varied points. Each rollout batch contains a mix of task stages. This yields a more stationary and representative data distribution.

- **End-of-Horizon Resets:** When an environment completes its full H-step lifetime, it is reset to a new initial state $s_0 \sim \rho_0$ at effective time $t = 0$. Since environments were initially staggered into groups, these resets occur in batches, maximizing efficiency.
- **Partial Resets (Early Termination):** If an environment terminates early (e.g., due to task success or failure), it is flagged for reset. To avoid costly un-batched operations, it waits for the next scheduled "reset gate"—the moment when any group of environments is undergoing a batched end-of-horizon reset. At this point, all flagged environments are reset together. This strategy maintains the benefits of staggering with minimal wall-clock overhead.

By ensuring temporal diversity *within each batch*, staggered resets provide the learner with a data distribution that better approximates the true state visitation distribution $\rho_\pi^{(0:H-1)}$ encountered over complete episodes. This stabilization is crucial for allowing the use of short rollouts K (and thus high update-to-data ratios) without succumbing to the cyclical nonstationarity bias that plagues naive synchronous reset schemes. The improved data quality promotes more stable learning, better value estimates for states across the entire task, and ultimately, enhanced performance on long-horizon tasks, as our empirical results corroborate.

4 Characterizing the Problem in Toy Environments

To isolate the factors that amplify the harm of cyclical nonstationarity, we designed 1D toy environments where we could control for: (1) the ratio of rollout length to task horizon (K/H), (2) the stochasticity of reset states, and (3) the presence of "skill gates" that force mastery of early-stage skills to progress. Full details of the environment design and results are in the appendix.

As shown in Figure 2, staggered resets consistently outperform naive synchronous resets. The performance gap widens significantly as: (a) the horizon H increases relative to K, lengthening the data-distribution cycle; (b) resets become more deterministic (homogeneous), synchronizing the environments more strongly; and (c) skill gates are relaxed, meaning the environment provides less of a natural curriculum. These results confirm that staggered resets are most crucial in long-horizon tasks where the data from synchronous rollouts is least representative of the full state space on a per-batch basis.

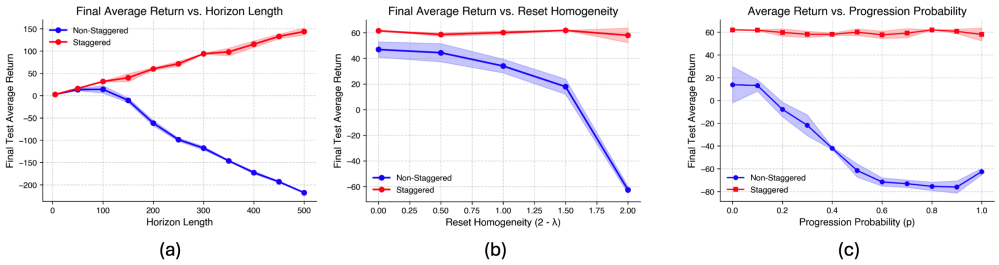

Figure 2: PPO with Non-Staggered (blue) vs. Staggered (red) resets on toy environments (mean $\pm$ 1 std dev). Staggered resets show robust performance as (a) horizon H increases, (b) reset homogeneity increases, and (c) progression probability varies, unlike non-staggered PPO which degrades especially with longer horizons, more deterministic resets, and easier skill gates.

5 Experiments on High-Dimensional Robotics Tasks

5.1 Experimental Setup

Our evaluation suite includes several challenging robotics tasks from ManiSkill3 (Tao et al., 2025), a GPU-accelerated robotics framework based on SAPIEN (Xiang et al., 2020). We test on `StackCube-v1`, a manipulation task requiring an agent to stack one cube onto another; `PushT`, where a T-shaped block must be pushed to a target pose; `TwoRobotPushCube`, where two robots work together to move a cube to a goal; `Unitree Transport Box`, a humanoid task where a box must be transported to a table; and `Anymal Reach C`, where an Anymal C robot must move to a specific goal location. We also test on `MS-HumanoidWalk`, a humanoid walking control task. These environments involve high-dimensional continuous state and action spaces, providing a suitable testbed for evaluating the effectiveness of staggered resets.

5.2 State Visitation Dynamics in High-Dimensional Robotics

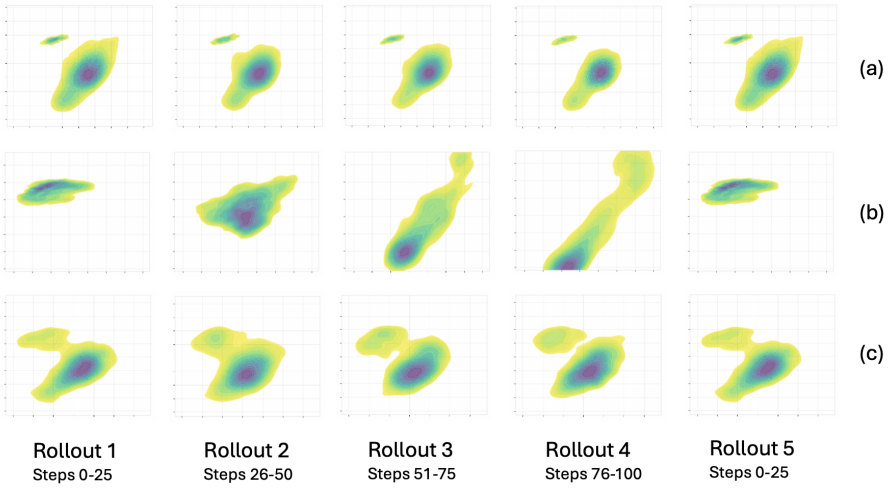

Figure 3: State visitation KDEs in `StackCube-v1` over five rollouts. (a) Long Rollout ($K = 100$): stable, broad coverage. (b) Naive Short Rollout ($K = 25$): cyclical non-stationarity, narrow/erratic coverage. (c) Staggered Short Rollout ($K = 25$): stable, diverse coverage, emulating (a) despite short trajectories.

We hypothesize that staggered resets create a more stable and informative data distribution. To test this, we visualize the state visitation distributions for `StackCube-v1` over consecutive rollouts using KDE plots (Figure 3). A long rollout ($K = 100$) policy (a) serves as our "ideal" baseline, showing broad and stable state coverage. In contrast, the naive short rollout ($K = 25$) approach (b) exhibits the predicted cyclical nonstationarity: the distribution is narrow, shifts erratically between rollouts, and snaps back to the initial state distribution after a mass reset (compare Rollout 5 to 1). Our staggered short rollout ($K = 25$) method (c) successfully mitigates this. Its state coverage remains broad and stable across rollouts, qualitatively matching the long-rollout baseline. This provides strong evidence that staggered resets restore the data quality lost when using short rollouts with synchronous resets.

5.3 Performance on High-Dimensional Robotics Tasks

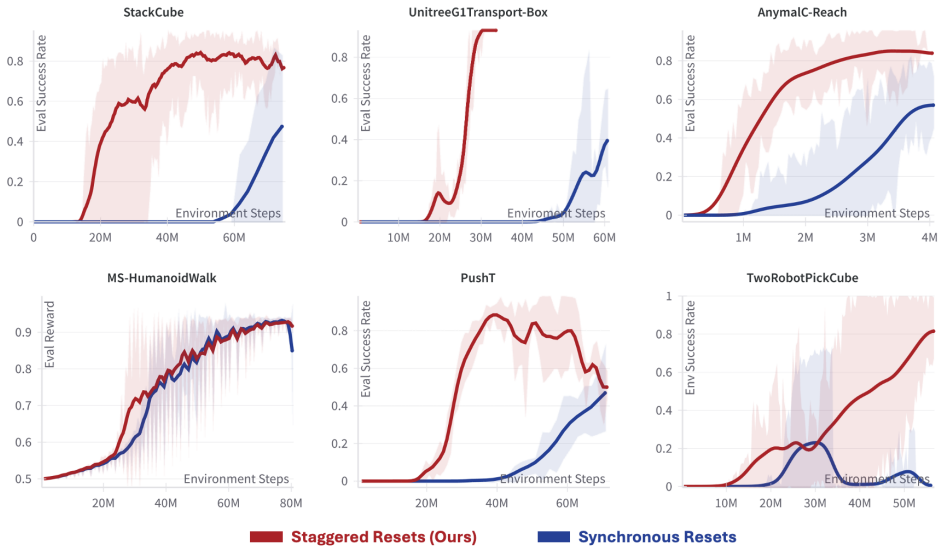

Figure 4: Staggered Resets (Ours, red) vs. Synchronous Resets (blue) on robotics tasks. Plots show the average evaluation metric (success rate or reward) vs. environment steps. Shaded area show the standard deviation over 3 seeds. Staggered resets consistently improve learning speed, final performance, and stability on diverse manipulation tasks (`StackCube`, `PushT`, `TwoRobotPickCube`, `UnitreeG1Transport-Box`, `AnymalC-Reach`). Performance is comparable on the locomotion task `MS-HumanoidWalk`, where cyclical batch nonstationarity is less salient due to the repetitive nature of short horizon skills.

As shown in Figure 4, across manipulation tasks like `PushT`, `StackCube-v1`, `UnitreeG1Transport-Box`, `AnymalC-Reach`, and `TwoRobotPickCube`, staggered resets achieve substantially faster convergence, higher final success rates, and greater stability. For instance, in `PushT` and `StackCube-v1`, staggering leads to significantly higher and more stable success rates. Similarly, in `AnymalC-Reach` and `TwoRobotPickCube`, learning is markedly quicker and reaches better asymptotic performance with staggered resets. The variance across seeds ($n = 3$) is also lower with staggered resets, indicating higher overall training stability.

Interestingly, on the locomotion task `MS-HumanoidWalk`, both methods perform comparably. Locomotion tasks often feature shorter effective skill horizons per cycle. Concretely, locomotion environments feature extremely large task horizons (sometimes $H \approx 1000$), but the actual locomotion skill itself is much shorter horizon. Additionally, locomotion environments feature highly stochastic dynamics and reset behaviors (i.e. partially resetting by falling down early). These factors can induce a degree of natural desynchronization, making the cyclical batch nonstationarity less severe and thereby reducing the marginal benefit of explicit staggered resets. These observations

align with our toy environment findings, where higher reset stochasticity and shorter effective skill horizons diminish the advantages of staggering. It is noteworthy that prior work which employs a form of staggering (Rudin et al., 2022) primarily applies it in such locomotion contexts—scenarios where, as our analysis suggests, the benefits of staggering are less pronounced compared to more complex, longer-horizon manipulation tasks.

5.4 Scaling with Parallel Environments and Overcoming Performance Saturation

A key challenge in massively parallel RL is effectively utilizing the increased parallelism, as performance gains can saturate or even reverse with more environments (Singla et al., 2024). We show that staggered resets directly address this by improving the marginal utility of each added environment.

Figure 5 plots wall-clock time to reach a 70% success rate versus the number of parallel environments N. While Naive PPO's convergence speed saturates at high N, staggered resets enable continued performance gains. This demonstrates that by ensuring data diversity, our method scales more effectively and makes better use of parallel compute, overcoming the diminishing returns caused by the temporally redundant data generated by naive synchronous resets.

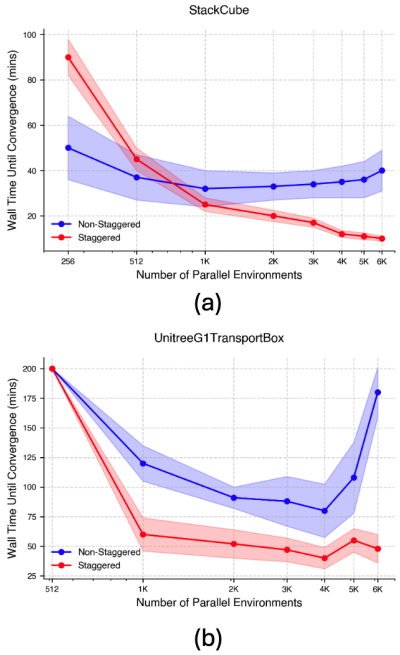

Figure 5: Wall-clock time to convergence versus number of parallel environments (N) for (a) `StackCube-v1` and (b) `Unitree G1 Transport Box`. Staggered PPO (red) scales more efficiently than Naive PPO (blue).

6 Discussion

Our investigation reveals a significant challenge in modern massively parallel on-policy RL. The cyclical nonstationarity introduced by synchronous environment resets when coupled with short rollouts ($K \ll H$) inadvertently biases the learning signal by repeatedly oversampling states from the initial segments of episodes. This bias can detrimentally affect learning stability, convergence speed, and ultimate policy quality.

Staggered resets directly address this nonstationarity without any changes to the core learning algorithm. By deliberately initializing parallel environments at varied effective time steps within the task horizon, we ensure that each training batch encompasses a temporally diverse set of experiences. This creates a more stationary and representative data distribution for the learner. The state visitation KDEs (Figure 3) offer compelling visual evidence: staggered resets with short rollouts yield broad and stable state coverage, closely emulating the desirable properties of much longer rollouts, whereas naive short rollouts suffer from erratic, cycling state distributions.

The illustrative toy experiments (Section 4) further characterize the conditions where staggered resets are most impactful. Specifically, tasks with longer horizons relative to the rollout length, more deterministic (homogeneous) reset states, and weaker intrinsic task curricula (e.g., where agents are not strictly forced to master early skills to progress) show pronounced benefits from the explicit temporal diversification that staggering provides.

A key practical advantage of staggered resets is their ability to enhance the scalability of on-policy RL in massively parallel settings (Section , Figure 5). While naive PPO often encounters diminishing returns or even performance degradation as the number of parallel environments grows—likely due to increasingly redundant data—staggered resets facilitate continued improvements in wall-clock convergence time. This indicates a more effective utilization of parallel compute resources, as the increased data volume is also more diverse and informative.

In conclusion, staggered resets provide a robust, easily implementable, and computationally inexpensive method to significantly enhance the performance and scalability of on-policy RL in common high-throughput, short-rollout regimes. By directly addressing the issue of cyclical data nonstationarity, this technique allows for more stable value estimation, faster convergence, and better final policies, paving the way for more effective learning in complex, long-horizon tasks.

References

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1406–1415. PMLR, 2018. URL http://proceedings.mlr.press/v80/espeholt18a.html.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023.

Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. *arXiv*, 2022.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1Dy---0Z.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, and K Milan. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Sci. Robotics*, 4(26), 2019. DOI: 10.1126/SCIROBOTICS.AAU5872. URL https://doi.org/10.1126/scirobotics.aau5872.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. In *Journal of Machine Learning Research*, volume 17, pp. 1–40, 2016.

Yunan Li, Viktor Makoviychuk, Yash Narang, Fabio Ramos, Marco Hutter, Arthur Allshire, and David Hoeller. Parallel advantage actor-critic for rapid end-to-end robotics training. *IEEE Robotics and Automation Letters*, 8(11):7113–7120, 2023.

Toru Lin, Kartik Sachdev, Linxi Fan, Jitendra Malik, and Yuke Zhu. Sim-to-real reinforcement learning for vision-based dexterous manipulation on humanoids. *arXiv:2502.20396*, 2025.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Gabriel Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. In *Robotics: Science and Systems*, 2022.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. DOI: 10.1109/LRA.2023.3270034.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, pp. 1928–1937, 2016.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015. URL http://arxiv.org/abs/1507.04296.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2022.

Dmitry Nikulin, Francis Tasse, Berna Balaguer, John Anderson, Zhe Rui, Anton Raichuk, Mohammadjavad Barekatain, Nicolas Heess, Jean-Baptiste Lespiau, Anita Muldal, et al. Xland-minigrid: A fast and configurable rl environment framework. In *International Conference on Machine Learning*, pp. 26401–26422. PMLR, 2023.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann (eds.), *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 91–100. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/rudin22a.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. volume abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level Atari with human-level efficiency. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 30365–30380. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/schwarzer23a.html.

Ritvik Singh, Arthur Allshire, Ankur Handa, Nathan Ratliff, and Karl Van Wyk. Dextrah-rgb: Visuomotor policies to grasp anything with dexterous hands, 2025. URL https://arxiv.org/abs/2412.01791.

Jayesh Singla, Ananye Agarwal, and Deepak Pathak. Sapg: Split and aggregate policy gradients. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, Proceedings of Machine Learning Research, Vienna, Austria, July 2024. PMLR.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *Robotics: Science and Systems*, 2025.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. DOI: 10.1109/IROS.2012.6386109.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. DOI: https://doi.org/10.1016/j.simpa.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carlo Sferrazza, Yuval Tassa, and Pieter Abbeel. Mujoco playground: An open-source framework for gpu-accelerated robot learning and sim-to-real transfer., 2025. URL https://github.com/google-deepmind/mujoco_playground.

A Implementation Details

A.1 Implementation Details and Hyperparameters for ManiSkill Experiments

This section details the Proximal Policy Optimization (PPO) configuration for experiments on ManiSkill robotics tasks (Section 5), utilizing ManiSkill3 Tao et al. (2025) for GPU-accelerated simulation. The exact PPO implementation is based on the one provided by ManiSkill3 baselines, which is based on LeanRL and CleanRL. Table 1 provides a comprehensive list of hyperparameters.

Table 1: PPO Hyperparameters for ManiSkill Experiments. Common settings are listed first, followed by per-environment variations where applicable. Shaded rows categorize parameters.

gray!30 **Hyperparameter**	**Value / Per-Environment Specification**
gray!10	**PPO Algorithm Core Settings**
Learning Rate	3×10^{-4}
Update Epochs	4
Number of Minibatches	32
PPO Clipping Coefficient (ϵ)	0.2
Value Function Loss Coefficient	0.5
Entropy Bonus Coefficient	0.005
Max Gradient Norm	0.5
Advantage Normalization	True (Per minibatch)
Target KL for Early Stopping	0.1
gray!10	**Network Architecture (Actor & Critic MLP)**
Hidden Layers	3
Units per Hidden Layer	[256, 256, 256]
Activation Function	Tanh
Weight Initialization	Orthogonal
Policy Output	Gaussian mean, learnable state-independent log std. dev.
gray!10	**Optimization**
Optimizer	Adam
Adam Epsilon	1×10^{-5}
Learning Rate Annealing	False
gray!10	**Environment Interaction & Data Collection (Common)**
Total Training Timesteps	2×10^{8}
Partial Resets (Training)	True
Evaluation Environments	128
Evaluation Partial Resets	False
Observation Normalization	Via environment wrappers / running mean & std
Reward Scaling	Environment-dependent (aim for std approx. 1)
gray!10	**Staggered Resets Mechanism (When Enabled)**
Staggering Mode	Uniform distribution of start times
Number of Stagger Blocks (N_B)	$\lceil H/K \rceil$ (Task Horizon / Rollout Length)
Stagger Step Size (S)	K (Rollout Length)
gray!10	**Per-Environment Specific Hyperparameters**
Parameter	**Values for: StackCube / PushT / AnymalC / HumanoidWalk / TwoRobotCube / UnitreeBox**
Rollout Length (K)	8 / 8 / 16 / 64 / 16 / 32
Task Horizon (H)	100 / 100 / 200 / 1000 / 100 / 500
Discount Factor (γ)	0.8 / 0.99 / 0.99 / 0.97 / 0.8 / 0.8
GAE Lambda (λ)	0.9 / 0.9 / 0.95 / 0.9 / 0.9 / 0.9
Num. Parallel Env. (N)	4096 / 4096 / 512 / 4096 / 2048 / 1024

A.2 Staggered Reset Implementation Details

The staggered reset mechanism aims to distribute the effective starting timesteps of the N parallel environments across the task horizon H. This was achieved by dividing environments into $N_B = \lceil H/K \rceil$ groups, with each group j starting its first "effective" episode step after an initial offset of $j \cdot K$ simulation steps (typically performed with random actions or the initial policy). This ensures that each PPO batch contains data from various segments of the task horizon.

A.3 Details on Toy Environments

The toy environments described in Section 4 were designed to isolate and study the effects of data nonstationarity under different environmental conditions. See Table 2 for the hyperparameters cho-

sen for PPO in the toy environment. We describe more concretely the ablation and environment implementation details below.

A.3.1 Environment Dynamics

The environment is a 1-dimensional chain of B discrete levels or "blocks". An episode lasts for a maximum of H time steps. Each level $b \in \{0, \ldots, B-1\}$ covers $L = H/B$ steps. The agent's state s_t is its current level index $b_t = \lfloor \text{elapsed\_steps}_t / L \rfloor$. At each time step t, the agent, being in level b_t, chooses an action a_t from a discrete set of A_c categories (e.g., $A_c = 20$). Each level b has a pre-assigned target action $a_b^* \in \{0, \ldots, A_c - 1\}$. The reward function is:

$$r(s_t, a_t) = \begin{cases} +0.5 & \text{if } a_t = a_{b_t}^* \\ -0.5 & \text{if } a_t \neq a_{b_t}^* \end{cases}$$

The episode terminates if $\text{elapsed\_steps}_t \geq H$. Success in an episode is defined as being in the final block ($b_t = B - 1$) when the episode terminates.

A.3.2 Further Details on Ablations on Toy Environments

The following parameters were varied to create the different experimental conditions shown in Figure 2:

- **Horizon Length (H vs. K):** (Figure 2a) The task horizon H (max_steps in code) was varied across values [50, 100, 200, 300, 400, 500]. The PPO rollout length K (num_steps in PPO loop, i.e., buffer size per environment before update) was kept fixed at $K = 5$. The block length L was also fixed at 5. For this experiment, skill gating was moderate ($p_{\text{prog}} = 0.5$, $k_{\text{mastery}} = 3$) and reset was deterministic ($\lambda_R = 0$).
- **Reset Stochasticity/Homogeneity (λ_R):** (Figure 2b) Upon episode termination, the reset mechanism was varied. The parameter λ_R (reset_stochasticity_lambda in code) controls the mean of a Poisson distribution from which the starting block b_0 is sampled, i.e., $b_0 \sim \text{Poisson}(\lambda_R)$, clamped to $[0, B-1]$. $\lambda_R = 0$ corresponds to a deterministic reset to $b_0 = 0$. The "Reset Homogeneity" axis in the plot is $2.0 - \lambda_R$ for visualization purposes (higher values = more deterministic starts at $b_0 = 0$). λ_R was varied in $[0.0, 0.1, \ldots, 1.0]$. For this experiment, $H = 50$, $L = 5$, $K = 5$, $p_{\text{prog}} = 1.0$ (easy progression), $k_{\text{mastery}} = 3$.
- **Skill Gating Dynamics (p_{prog}):** (Figure 2c) Progression from the current block b to $b + 1$ (when enough steps within block b have nominally passed to enter $b + 1$) occurs if either:
  1. The agent has achieved "mastery" in block b, defined as making at least k_{mastery} correct actions a_b^* within block b during the current episode. ($k_{\text{mastery}} = 3$ was used).
  2. A random chance p_{prog} for unconditional progression succeeds.

  The probability p_{prog} (progression_prob in code) was varied in $[0.0, 0.1, \ldots, 1.0]$. $p_{\text{prog}} = 0$ means hard gating requiring mastery. For this experiment, $H = 200$, $L = 5$, $K = 5$, $\lambda_R = 0$.

B Additional Results on Toy Environments

To further illustrate the impact of synchronous versus staggered resets on the data distribution and learning progress, we visualize the training dynamics in one of the toy environments (specifically, $H = 200, L = 5, K = 5, p_{\text{prog}} = 0.5, \lambda_R = 0$). Figure 6 shows the training average accuracy and the mean distribution of environments across different blocks (states) over the course of training updates. For these experiments, we define accuracy as the rolling percentage of correct one-hot action guesses from the PPO agent.

Subplots (c) and (d) in Figure 6 are heatmaps where the x-axis represents the PPO training update index, the y-axis represents the block index (state) within the toy environment's episode, and the

Table 2: PPO and Environment Hyperparameters for Toy Environment Experiments. Shaded rows categorize parameters. Values listed are defaults; specific sweeps varied H, p_{prog}, or λ_R as detailed in text and figures.

gray!30 **Hyperparameter**	**Value**
gray!10	**PPO Algorithm Core Settings**
Learning Rate	3×10^{-4}
Discount Factor (γ)	0.99
GAE Lambda (λ)	0.95
PPO Rollout Length (K)	5 steps per environment
Number of Parallel Environments (N)	512
Total Training Updates	150
Update Epochs	4
Number of Minibatches	4 (Minibatch size: $(512 \times 5)/4 = 640$)
PPO Clipping Coefficient (ϵ)	0.2
Value Function Loss Coefficient	0.5
Entropy Bonus Coefficient	0.01
Max Gradient Norm	0.5
gray!10	**Network Architecture (Actor & Critic MLP)**
Input	Current block index (integer state)
Embedding Layer	Input block index to 64-dim embedding
Hidden Layers	4
Units per Hidden Layer	256
Activation Function	ReLU
Policy Output	Categorical distribution over actions
Value Output	Scalar state value
gray!10	**Optimization**
Optimizer	Adam
gray!10	**Toy Environment Base Parameters (Defaults for Sweeps)**
Episode Horizon (H)	Varied (e.g., 50, 100, 200, 375 for specific experiments)
Block Length (L)	5 steps
Number of Action Categories (A_c)	20
Reward for Correct Action	+0.5
Reward for Incorrect Action	-0.5
Success Definition	Agent is in the final block at episode end
Skill Gating: Progression Prob. (p_{prog})	Varied (0.0 to 1.0)
Skill Gating: Mastery Threshold (k_{mastery})	3 correct actions
Reset Stochasticity (λ_R)	Varied (Poisson mean for start block, 0.0 for deterministic)
gray!10	**Staggered Resets Mechanism (When Enabled)**
Number of Stagger Blocks (N_B)	$\lceil H/K \rceil = \lceil \text{Episode Horizon}/5 \rceil$
Stagger Step Size (S)	$K = 5$

color intensity indicates the mean number of parallel environments present in that block at that specific training update.

Naive Synchronous Resets (Figure 6c) The heatmap for naive synchronous resets clearly shows distinct diagonal bands. Each band signifies that the cohort of parallel environments is synchronously progressing through the episode's blocks. A crucial observation is the abrupt termination of these bands followed by an immediate restart from block 0 (the bottom of the y-axis). This occurs approximately every 40 updates, corresponding to the episode horizon ($H = 200$) divided by the rollout length ($K = 5$). This pattern visually confirms the cyclical nonstationarity discussed in Section **??**: at any given update, the training batch is predominantly composed of states from a narrow segment of the episode, and this segment predictably cycles. For instance, for updates 1-5, data is from blocks near 0-4; for updates 35-40, data is from blocks near 35-39; then at update 41, data abruptly shifts back to blocks 0-4.

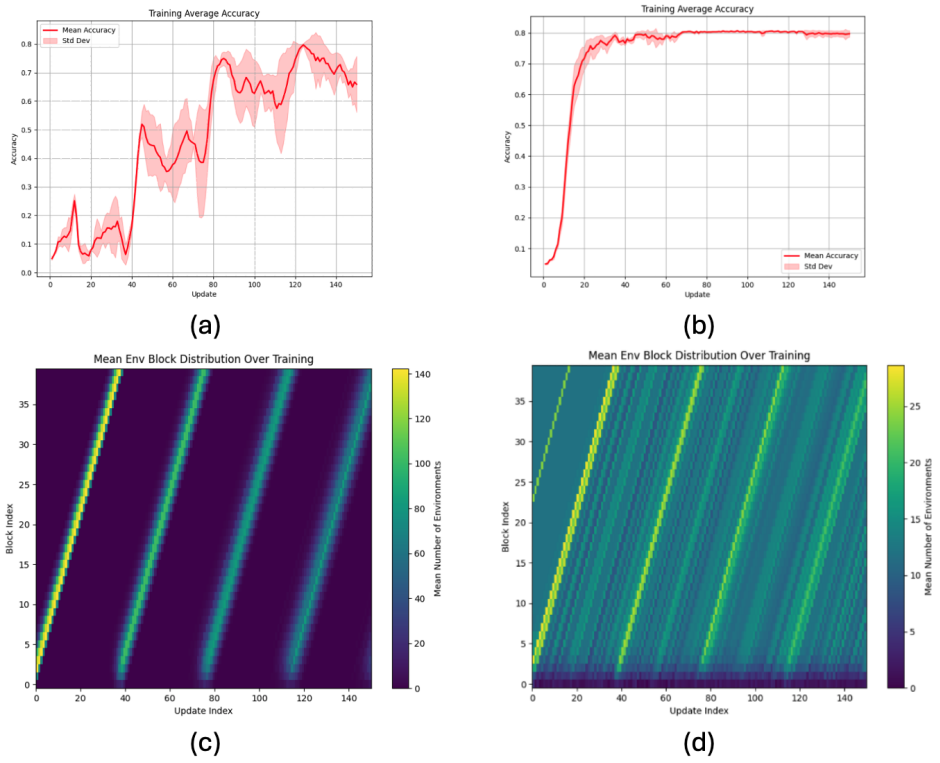

Figure 6: Comparison of training dynamics in a toy environment with (a, c) naive synchronous resets versus (b, d) staggered resets. **(a) & (b):** Training average accuracy over 150 PPO updates. With naive resets (a), accuracy is unstable and struggles to converge. With staggered resets (b), accuracy rises quickly and stabilizes at a high level. **(c) & (d):** Heatmaps showing the mean number of environments occupying each block (y-axis) at each PPO update index (x-axis). (c) With naive synchronous resets, environments progress through blocks in tight, synchronized waves. After approximately 40 updates (when $H/K = 200/5 = 40$ rollouts complete an episode), all environments abruptly reset to block 0, leading to a cyclical pattern where training batches are temporally homogeneous (all early-episode, then all mid-episode, etc.). (d) With staggered resets, the distribution of environments across blocks is far more uniform at any given update index. This indicates that each training batch contains a mix of experiences from different stages of the episode, leading to a more stationary data distribution for the learner.

Staggered Resets (Figure 6d) In contrast, the heatmap for staggered resets shows a much more diffuse and uniform pattern. While environments still progress through blocks (indicated by the general upward-right trend), there is no global, abrupt reset of all environments. At any given PPO update index, environments are distributed across a wide range of blocks. This means that each training batch collected under staggered resets contains a temporally diverse mix of experiences—some from early parts of episodes, some from middle, and some from later parts. This significantly reduces the cyclical nonstationarity of the data fed to the PPO algorithm.

Impact on Learning (Figure 6a and 6b) The consequences of these different state visitation dynamics are evident in the training accuracy plots. With naive synchronous resets (Figure 6a), the learning curve for average accuracy is highly erratic, exhibiting periodic dips and slow overall improvement, struggling to consistently achieve high accuracy. This instability likely results from the PPO learner trying to adapt to the rapidly shifting data distributions. Conversely, with staggered resets (Figure 6b), the average accuracy rises smoothly and rapidly, quickly converging to a high

and stable level. This demonstrates that providing the learner with more temporally diverse and stationary batches facilitates more effective and stable learning.

B.1 State Visitation Distribution Analysis

We provide further detail on the state visitation dynamics in our toy environments by visualizing the mean environment block distribution over training updates (similar to Figure 6c and 6d) while systematically varying key environmental parameters. For all visualizations in this section, the PPO rollout length $K = 5$, number of parallel environments $N = 512$, total training updates shown are 150, environment block length $L = 5$, number of action categories $A_c = 20$, and mastery threshold $k_{\text{mastery}} = 3$, unless specified otherwise. Base PPO hyperparameters are detailed in Appendix A.3. Each figure's top row shows results for naive synchronous resets, and the bottom row shows results for staggered resets.

B.2 Impact of Progression Probability (p_{prog})

Figure 7 illustrates how the probability of unconditional progression, p_{prog}, affects state visitation. The experiment uses a fixed horizon $H = 200$ and deterministic resets ($\lambda_R = 0$).

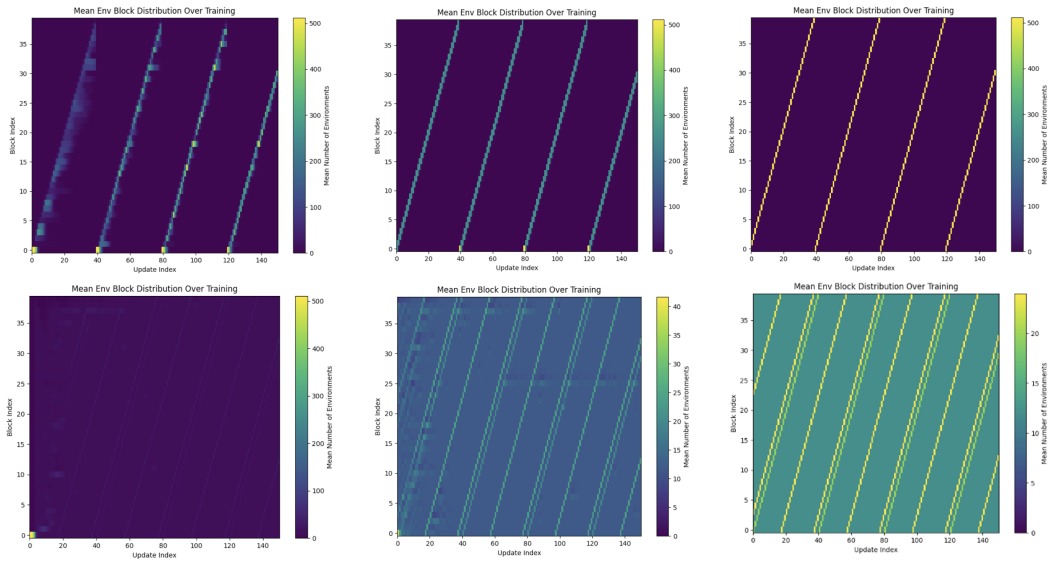

Figure 7: Mean environment block distribution over training for varying progression probabilities (p_{prog}). **Top Row (Non-Staggered):** From left to right, $p_{\text{prog}} = 0.0, 0.5, 1.0$. **Bottom Row (Staggered):** From left to right, $p_{\text{prog}} = 0.0, 0.5, 1.0$. With non-staggered resets, low p_{prog} (hard skill gating) leads to environments bunching at early blocks, with sparse exploration of later stages. As p_{prog} increases, environments progress more freely, but the strong cyclical reset pattern remains. With staggered resets, coverage is more uniform across updates regardless of p_{prog}, though higher p_{prog} allows environments to explore the full range of blocks more rapidly within their individual (staggered) episode timelines.

Non-Staggered Resets (Top Row):

- $p_{\text{prog}} = 0.0$ **(Left):** With hard skill gating, environments get stuck at early blocks if they fail to achieve mastery. The heatmap shows most environments concentrated at very low block indices, with only very few managing to progress. The cyclical reset pattern is still evident for those that do run the full course or get reset.

- $p_{\textbf{prog}} = 0.5$ **(Center):** Partial random progression allows more environments to reach later blocks, but the density remains higher at earlier stages due to the gating. The cyclical nature of resets is clear.
- $p_{\textbf{prog}} = 1.0$ **(Right):** Environments progress freely through blocks irrespective of mastery. This results in the most pronounced cyclical bands, as all environments synchronously march through the episode blocks and reset together.

Staggered Resets (Bottom Row): Across all values of p_{prog}, staggered resets maintain a significantly more uniform distribution of environments across different blocks at any given training update. While a lower p_{prog} means individual environments might take longer or struggle more to traverse all blocks within their own episode lifetime, the staggering ensures that the batch fed to the learner still contains diverse experiences. The overall "texture" of the heatmap becomes denser as p_{prog} increases, indicating that more environments are successfully exploring the full range of blocks over time, but the crucial within-batch temporal diversity is preserved by the staggered mechanism itself.

B.3 Impact of Episode Horizon Length (H)

Figure 8 shows the effect of varying the episode horizon length H. For these plots, progression probability $p_{\text{prog}} = 0.5$ and reset stochasticity $\lambda_R = 0$ are fixed. The number of blocks $B = H/L$ changes with H.

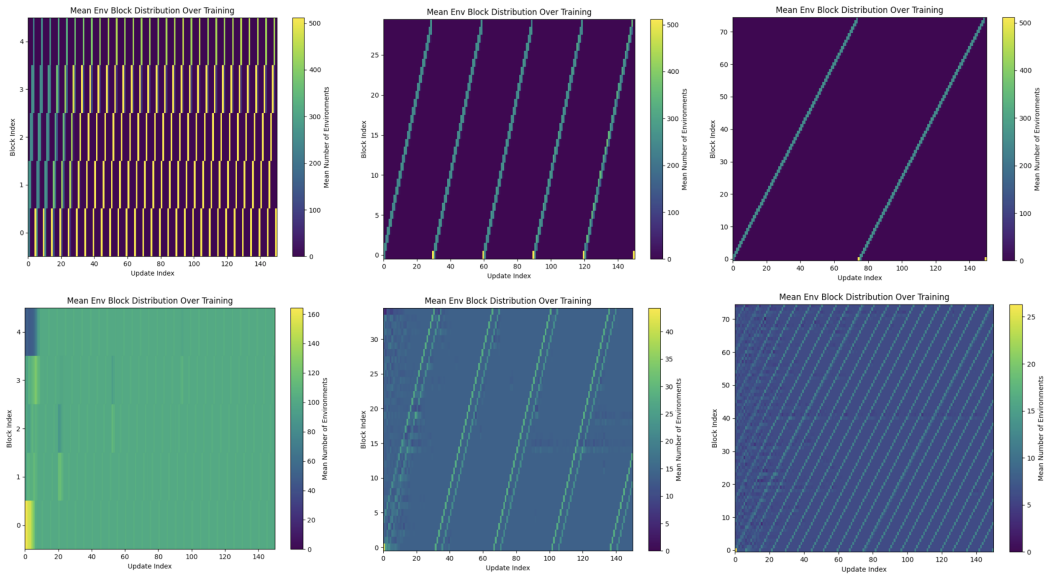

Figure 8: Mean environment block distribution over training for varying episode horizon lengths (H). **Top Row (Non-Staggered):** From left to right, $H = 10, 150, 375$. (Note: y-axis Block Index scales with H). **Bottom Row (Staggered):** From left to right, $H = 10, 150, 375$. For non-staggered resets, shorter horizons ($H = 10$) lead to very rapid cycles ($H/K = 10/5 = 2$ updates per cycle). As H increases, the period of these cycles becomes longer, potentially exacerbating learning instability. Staggered resets maintain uniform coverage irrespective of H.

Non-Staggered Resets (Top Row):

- $H = 10$ **(Left):** With a very short horizon, the full episode cycle $H/K = 10/5 = 2$ updates. The cyclical pattern appears as very rapid, almost vertical bands. While cyclical, all parts of this very short episode are revisited extremely frequently. The y-axis shows only 2 blocks ($10/5$).

- $H = 150$ **(Center):** The cycle period is $150/5 = 30$ updates. Clear diagonal bands show synchronous progression and reset. The y-axis spans 30 blocks.
- $H = 375$ **(Right):** The cycle period becomes $375/5 = 75$ updates. The bands are elongated, indicating a longer time between revisiting the same episode phase. The y-axis spans 75 blocks. This long periodicity is hypothesized to be particularly detrimental.

Staggered Resets (Bottom Row): Staggered resets consistently provide diverse state visitations within each batch, regardless of the horizon length H. The heatmaps show that environments are distributed across the available blocks (which scale with H) at each update. This ensures the learner receives a more stationary data stream, which is particularly beneficial for longer horizons where the non-staggered approach suffers from infrequent revisitation of early-episode states.

B.4 Impact of Reset Stochasticity (λ_R)

Figure 9 visualizes the influence of reset stochasticity, λ_R, which controls the mean of a Poisson distribution for sampling the starting block upon reset. Here, $H = 200$ and $p_{\text{prog}} = 0.5$.

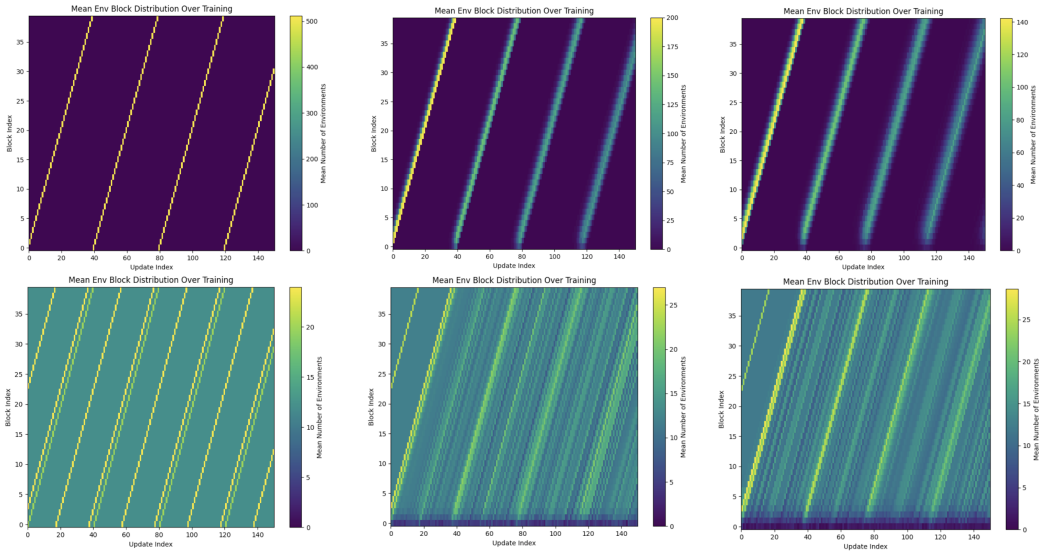

Figure 9: Mean environment block distribution over training for varying reset stochasticity (λ_R). **Top Row (Non-Staggered):** From left to right, $\lambda_R = 0.0, 1.0, 2.0$. **Bottom Row (Staggered):** From left to right, $\lambda_R = 0.0, 1.0, 2.0$. For non-staggered resets, increasing λ_R slightly "fuzzes" the start of each cycle after a mass reset, but the overall cyclical progression remains. Staggered resets maintain uniform coverage; λ_R primarily influences the initial state distribution within each environment's individual staggered timeline.

Non-Staggered Resets (Top Row):

- $\lambda_R = 0.0$ **(Left):** Deterministic reset to block 0. This results in the sharpest cyclical bands, as all environments restart precisely from the same initial state after completing an episode.
- $\lambda_R = 1.0$ **(Center) and** $\lambda_R = 2.0$ **(Right):** Stochastic resets mean environments restart from a distribution of initial blocks centered around block 0 (due to Poisson sampling with mean λ_R, then clamped). This causes the beginning of each major cycle (after most environments have run for H steps) to appear slightly "fuzzier" or more spread out near block 0. However, once this initial phase passes, the environments that didn't terminate early tend to re-synchronize in their progression, and the cyclical bands through the bulk of the episode remain prominent. The inherent reset stochasticity offers only a minor and temporary desynchronization.

Staggered Resets (Bottom Row): With staggered resets, the distribution of environments across blocks remains largely uniform over updates, irrespective of λ_R. The primary mechanism for achieving temporal diversity in batches is the explicit staggering of episode start times. The reset stochasticity parameter λ_R further diversifies the exact starting block for an environment when its individual (staggered) episode concludes and it resets, but it does not fundamentally change the broad, uniform coverage ensured by the staggering mechanism itself. Staggered resets are effective even with deterministic environment resets ($\lambda_R = 0.0$).

These visualizations across different parameter sweeps consistently highlight the ability of staggered resets to create more temporally diverse and stationary training batches compared to naive synchronous resets, providing a more stable learning signal for the on-policy RL agent.

C Wall-Time Results and Additional Analysis

C.1 Wall-Time Analysis for Staggering Granularity (N_B)

A critical aspect of staggered resets is balancing the desired temporal diversity of training batches with the computational overhead associated with environment reset operations. While an ideal scenario might involve resetting each environment at a unique timestep (effectively $N_B \approx N$, or even finer if $N_B \approx H/\text{sim\_dt}$), frequent, unbatched 'env.reset()' calls can be costly, especially in GPU-accelerated simulations. The parameter N_B, representing the number of distinct stagger groups or "blocks," controls this trade-off. A smaller N_B means fewer, larger groups of environments are reset/advanced synchronously, reducing reset call frequency but potentially coarsening the approximation of a truly staggered (temporally diverse) data distribution.

We empirically investigated this trade-off on the `StackCube-v1` task ($H = 100$, with PPO rollout $K = 8$, thus $H/K \approx 12.5$) by measuring the wall-clock time to reach a 70% success rate while varying N_B. Figure 10 illustrates the results.

The findings, shown in Figure 10, indicate:

- **Few Blocks ($N_B \approx 1$):** When $N_B = 1$, all environments are effectively synchronized, resembling the naive reset scheme. This results in the slowest wall-clock convergence due to the detrimental effects of cyclical batch nonstationarity.
- **Moderate Blocks ($N_B \approx H/K$):** As N_B increases, wall-clock time to convergence rapidly decreases. The optimal performance is typically observed when N_B is in the vicinity of H/K. For `StackCube-v1` with $K = 8$, this optimal is around $N_B \approx 10 - 13$. This granularity provides sufficient temporal diversity in training batches to stabilize learning and accelerate convergence.
- **Many Blocks ($N_B \gg H/K$):** Further increasing N_B beyond H/K leads to marginal improvements or even a slight degradation in wall-clock convergence time. While providing finer-grained staggering, the overhead of managing and executing resets for many small, distinct groups may start to outweigh the benefits from any additional (and likely minimal) gains in data diversity.

This analysis demonstrates that a judicious choice of N_B, typically around H/K, allows us to effectively approximate the benefits of a continuously staggered reset distribution (where each environment could theoretically start at any unique step within H) while maintaining wall-clock efficiency. This approach strikes a practical balance, achieving most of the sample efficiency gains from temporal diversity without incurring the potentially significant computational costs of excessively frequent or unbatched reset operations. The default setting in our experiments for staggering (see Appendix A.1) is chosen based on this principle, typically defaulting to $\lfloor H/K \rfloor$.

C.2 Wall-Clock Training Time for ManiSkill Environments

Beyond improvements in sample efficiency (i.e., performance per environment step), staggered resets also demonstrate significant advantages in terms of wall-clock training time. Figure 11 presents

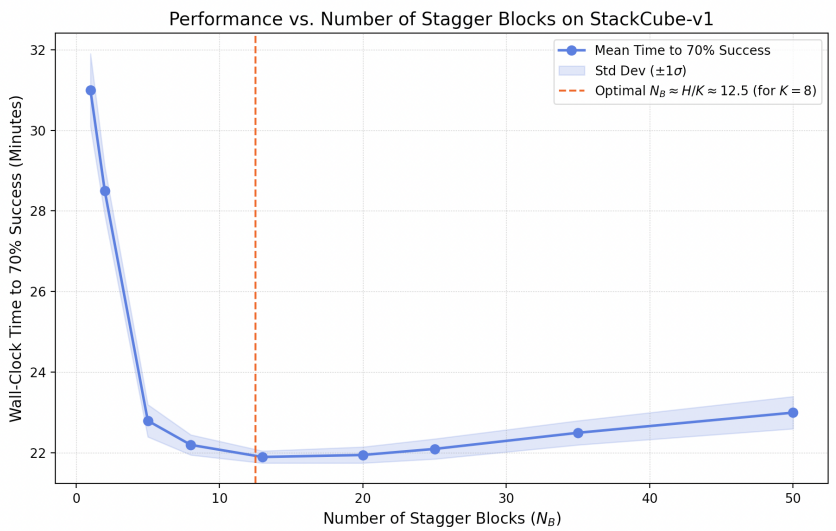

Figure 10: Wall-clock time to convergence (70% success on `StackCube-v1`) as a function of the number of stagger blocks (N_B). Performance, measured by faster convergence (lower wall-clock time), improves significantly as N_B increases from 1 (naive synchronous resets) towards $N_B \approx H/K$ (here, $H = 100, K = 8$, so $H/K \approx 12.5$). Beyond this point, further increasing N_B yields diminishing returns or even a slight increase in wall-clock time, likely due to the overhead of managing more numerous, smaller reset groups outweighing marginal gains in temporal diversity. This demonstrates that a moderate number of stagger blocks (e.g., $N_B \approx H/K$) effectively balances temporal diversity benefits with wall-time efficiency, approximating a continuously staggered reset distribution without incurring prohibitive reset costs.

a comparison of evaluation success rates against wall-clock time for several challenging ManiSkill tasks.

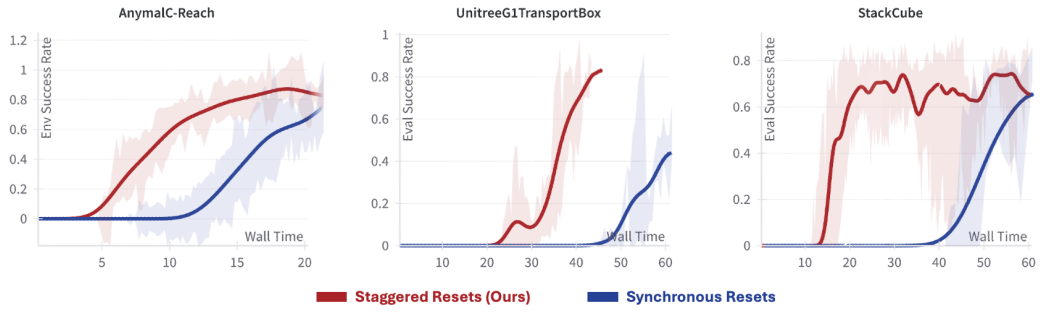

Figure 11: Comparison of evaluation success rates versus wall-clock training time for Staggered Resets (Ours, red) and Synchronous Resets (blue) on three ManiSkill tasks: `AnymalC-Reach`, `UnitreeG1TransportBox`, and `StackCube`. The x-axis represents wall-clock time (units may vary per plot, e.g., minutes or hours, but relative comparison is key). Staggered resets consistently achieve higher success rates faster, or reach comparable success rates in significantly less wall-clock time than synchronous resets. This highlights that the benefits of improved data quality and learning stability from staggered resets translate directly into more efficient use of compute time.

As illustrated in Figure 11, policies trained with staggered resets (red curves) consistently achieve target success rates in substantially less wall-clock time compared to those trained with naive synchronous resets (blue curves). For instance, in `AnymalC-Reach`, staggered resets reach over 80% success much earlier than synchronous resets begin to show significant learning. Similarly, for `UnitreeG1TransportBox` and `StackCube`, the learning curves for staggered resets are considerably steeper when plotted against wall time, indicating faster convergence to high-performing policies.

This empirical evidence supports the conclusion that the improved sample efficiency and learning stability afforded by staggered resets directly translate into reduced overall training time, making the technique not only more data-efficient but also more computationally efficient in practice for achieving desired performance levels on complex robotics tasks. The overhead of managing staggered resets is outweighed by the gains from more effective learning per unit of time.