

## 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 [26] 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.

Hyperparameter	Value / Per-Environment Specification
<b>PPO Algorithm Core Settings</b>	
Learning Rate	$3 \times 10^{-4}$
Update Epochs	4
Number of Minibatches	32
PPO Clipping Coefficient ( $\epsilon$ )	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
<b>Network Architecture (Actor &amp; Critic MLP)</b>	
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.
<b>Optimization</b>	
Optimizer	Adam
Adam Epsilon	$1 \times 10^{-5}$
Learning Rate Annealing	False
<b>Environment Interaction &amp; Data Collection (Common)</b>	
Total Training Timesteps	$2 \times 10^8$
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)
<b>Staggered Resets Mechanism (When Enabled)</b>	
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)
<b>Per-Environment Specific Hyperparameters</b>	
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 ( $\gamma$ )	0.8 / 0.99 / 0.99 / 0.97 / 0.8 / 0.8
GAE Lambda ( $\lambda$ )	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 chosen 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  $b \in \{0, \dots, B-1\}$  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  $a_b^* \in \{0, \dots, A_c - 1\}$ . 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 ( $\lambda_R$ ):** (Figure 2b) Upon episode termination, the reset mechanism was varied. The parameter  $\lambda_R$  (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  $b_0 = 0$ ).  $\lambda_R$  was varied in  $[0.0, 0.1, \dots, 1.0]$ . For this experiment,  $H = 50$ ,  $L = 5$ ,  $K = 5$ ,  $p_{\text{prog}} = 1.0$  (easy progression),  $k_{\text{mastery}} = 3$ .
- **Skill Gating Dynamics ( $p_{\text{prog}}$ ):** (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  $k_{\text{mastery}}$  correct actions  $a_b^*$  within block  $b$  during the current episode. ( $k_{\text{mastery}} = 3$  was used).
  2. A random chance  $p_{\text{prog}}$  for unconditional progression succeeds.

The probability  $p_{\text{prog}}$  (progression\_prob in code) was varied in  $[0.0, 0.1, \dots, 1.0]$ .  $p_{\text{prog}} = 0$  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 color intensity indicates the mean number of parallel environments present in that block at that specific training update.

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

Hyperparameter	Value
<b>PPO Algorithm Core Settings</b>	
Learning Rate	$3 \times 10^{-4}$
Discount Factor ( $\gamma$ )	0.99
GAE Lambda ( $\lambda$ )	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 ( $\epsilon$ )	0.2
Value Function Loss Coefficient	0.5
Entropy Bonus Coefficient	0.01
Max Gradient Norm	0.5
<b>Network Architecture (Actor &amp; Critic MLP)</b>	
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
<b>Optimization</b>	
Optimizer	Adam
<b>Toy Environment Base Parameters (Defaults for Sweeps)</b>	
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. ( $p_{\text{prog}}$ )	Varied (0.0 to 1.0)
Skill Gating: Mastery Threshold ( $k_{\text{mastery}}$ )	3 correct actions
Reset Stochasticity ( $\lambda_R$ )	Varied (Poisson mean for start block, 0.0 for deterministic)
<b>Staggered Resets Mechanism (When Enabled)</b>	
Number of Stagger Blocks ( $N_B$ )	$\lceil H/K \rceil = \lceil \text{Episode Horizon}/5 \rceil$
Stagger Step Size ( $S$ )	$K = 5$

545 **Naive Synchronous Resets (Figure 6c)** The heatmap for naive synchronous resets clearly shows  
546 distinct diagonal bands. Each band signifies that the cohort of parallel environments is synchronously  
547 progressing through the episode’s blocks. A crucial observation is the abrupt termination of these  
548 bands followed by an immediate restart from block 0 (the bottom of the y-axis). This occurs  
549 approximately every 40 updates, corresponding to the episode horizon ( $H = 200$ ) divided by the  
550 rollout length ( $K = 5$ ). This pattern visually confirms the cyclical nonstationarity discussed in  
551 Section 3.2 at any given update, the training batch is predominantly composed of states from a  
552 narrow segment of the episode, and this segment predictably cycles. For instance, for updates 1-5,  
553 data is from blocks near 0-4; for updates 35-40, data is from blocks near 35-39; then at update 41,  
554 data abruptly shifts back to blocks 0-4.

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

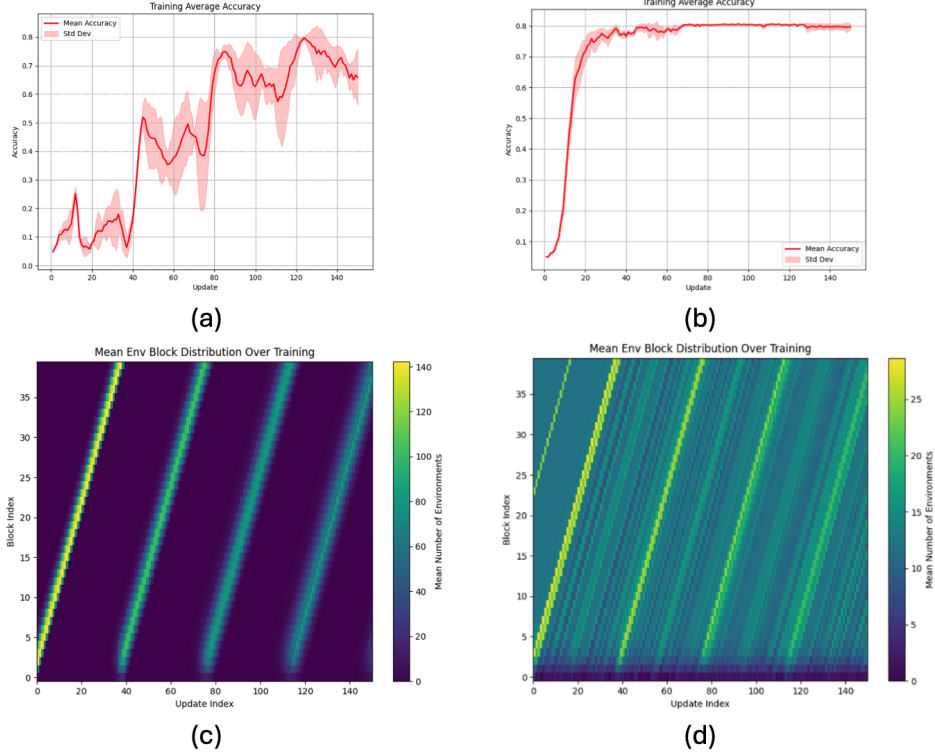


Figure 6: Comparison of training dynamics in a toy environment with (a, c) naive synchronous resets versus (b, d) staggered resets. **(a) & (b):** Training average accuracy over 150 PPO updates. With naive resets (a), accuracy is unstable and struggles to converge. With staggered resets (b), accuracy 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.

**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 ( $p_{\text{prog}}$ )

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

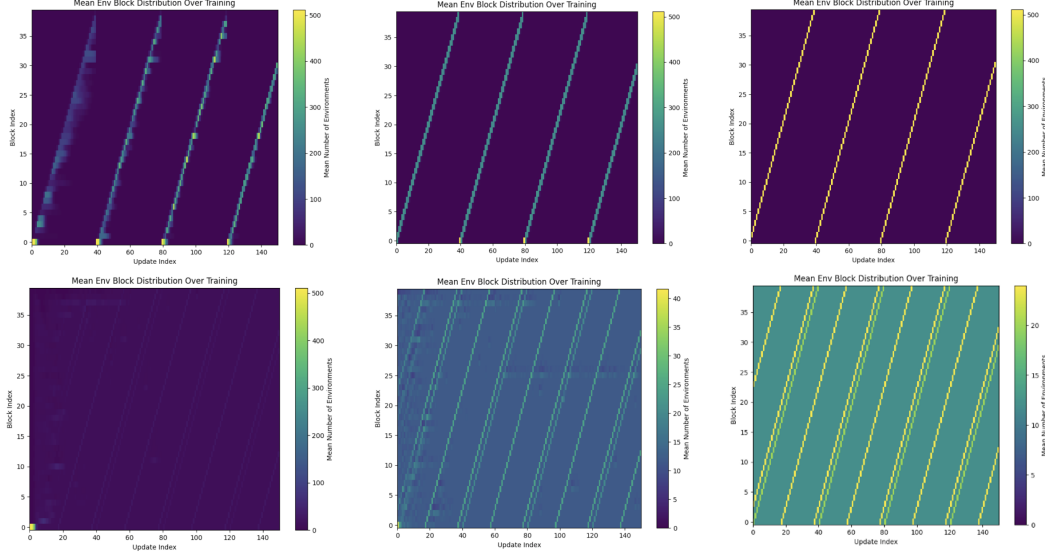


Figure 7: Mean environment block distribution over training for varying progression probabilities ( $p_{\text{prog}}$ ). **Top Row (Non-Staggered):** From left to right,  $p_{\text{prog}} = 0.0, 0.5, 1.0$ . **Bottom Row (Staggered):** From left to right,  $p_{\text{prog}} = 0.0, 0.5, 1.0$ . With non-staggered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

### 600 B.3 Impact of Episode Horizon Length ( $H$ )

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

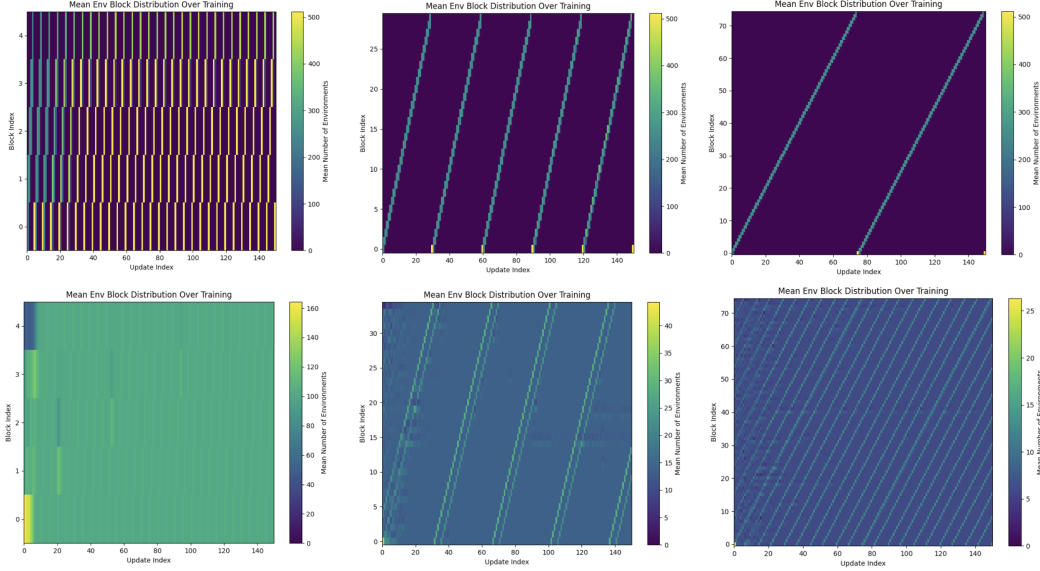


Figure 8: Mean environment block distribution over training for varying episode horizon lengths ( $H$ ).  
**Top Row (Non-Staggered):** From left to right,  $H = 10, 150, 375$ . (Note: y-axis Block Index scales with  $H$ ).  
**Bottom Row (Staggered):** From left to right,  $H = 10, 150, 375$ . For non-staggered 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  $H$ .

#### 604 Non-Staggered Resets (Top Row):

- 605 •  $H = 10$  (Left): With a very short horizon, the full episode cycle  $H/K = 10/5 = 2$  updates.  
 606 The cyclical pattern appears as very rapid, almost vertical bands. While cyclical, all parts of  
 607 this very short episode are revisited extremely frequently. The y-axis shows only 2 blocks  
 608 ( $10/5$ ).
- 609 •  $H = 150$  (Center): The cycle period is  $150/5 = 30$  updates. Clear diagonal bands show  
 610 synchronous progression and reset. The y-axis spans 30 blocks.
- 611 •  $H = 375$  (Right): The cycle period becomes  $375/5 = 75$  updates. The bands are elongated,  
 612 indicating a longer time between revisiting the same episode phase. The y-axis spans 75  
 613 blocks. This long periodicity is hypothesized to be particularly detrimental.

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

### 619 B.4 Impact of Reset Stochasticity ( $\lambda_R$ )

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

#### 622 Non-Staggered Resets (Top Row):



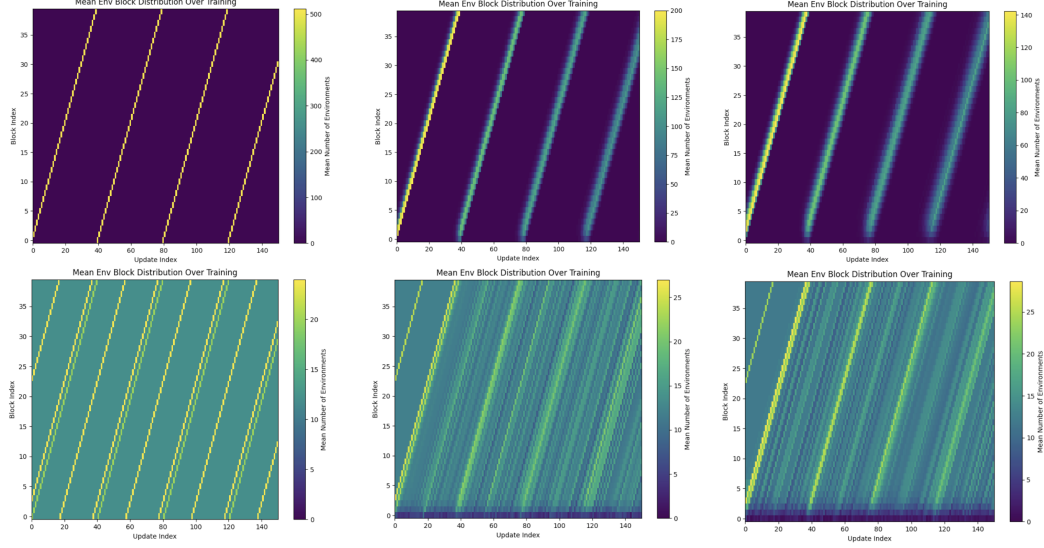


Figure 9: Mean environment block distribution over training for varying reset stochasticity ( $\lambda_R$ ). **Top Row (Non-Staggered):** From left to right,  $\lambda_R = 0.0, 1.0, 2.0$ . **Bottom Row (Staggered):** From left to right,  $\lambda_R = 0.0, 1.0, 2.0$ . For non-staggered resets, increasing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

- $\lambda_R = 0.0$  (**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  $\lambda_R$ , 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  $\lambda_R$ . The primary mechanism for achieving temporal diversity in batches is the explicit staggering of episode start times. The reset stochasticity parameter  $\lambda_R$  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.

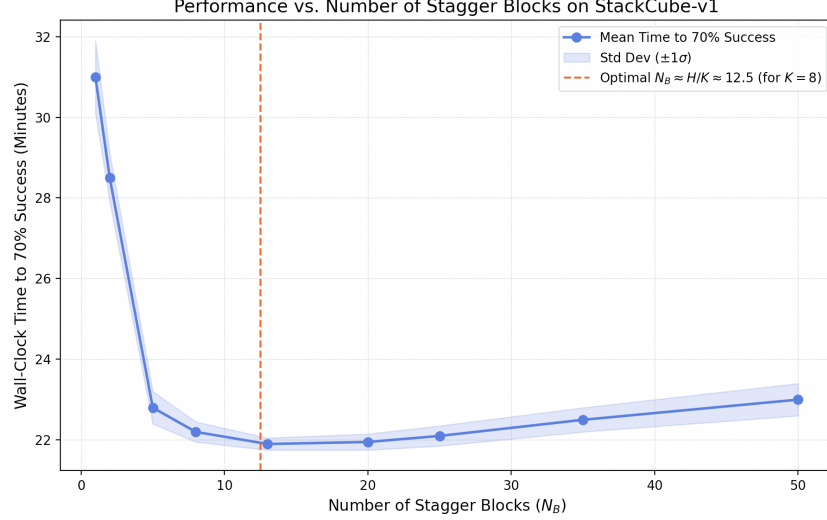


Figure 10: Wall-clock time to convergence (70% success on StackCube-v1) as a function of the number of stagger blocks ( $N_B$ ). Performance, measured by faster convergence (lower wall-clock time), improves significantly as  $N_B$  increases from 1 (naive synchronous resets) towards  $N_B \approx H/K$  (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.

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

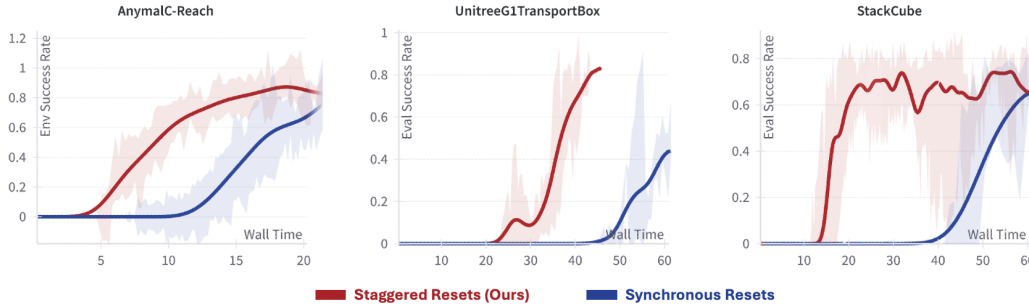


Figure 11: Comparison of evaluation success rates versus wall-clock training time for Staggered Resets (Ours, red) and Synchronous Resets (blue) on three ManiSkill tasks: AnymalC-Reach, UnitreeG1TransportBox, and StackCube. The x-axis represents wall-clock time (units may vary per plot, e.g., minutes 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.