GRL: FROM GPU UNDERUTILIZATION TO FASTER REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Compact networks in replay-based reinforcement learning (RL) often leave GPU capacity unused, yet increasing execution parallelism can also change the learning work needed to reach a target quality. We present GRL, a GPU execution workflow that preserves algorithm-defined learner-update structure while reorganizing collection, data delivery, and evaluation around it. Actor-side batching and cross-role overlap expose parallel work; GPU-resident handoff separates transfer completion from subsequent computation, minimizing synchronization overhead. Its learning-aware execution configurator screens candidates by throughput and compares learning quality through short trials, using a bounded search to limit selection overhead before training. Across eight environments with different complexity, GRL achieves end-to-end speedups up to $7.85\times$ over CPU-GPU asynchronous execution and $4.11\times$ over GPU-synchronous execution, measured across ten training runs sharing one configuration cost. Execution and configuration ablations show how throughput, learning progress, and selection overhead jointly shape first time-to-quality.

1 INTRODUCTION

Training efficiency remains an important concern in reinforcement learning (RL) for control and interactive decision-making, where policies are learned through repeated environment interaction (Li et al., 2023; Gallici et al., 2025). We focus on replay-based RL with compact networks (Mnih et al., 2015; Silver et al., 2014), which can underutilize GPUs. Asynchronous collection and learning expose concurrency (Mnih et al., 2016; Espeholt et al., 2018; Kapturowski et al., 2019), but conventional execution mappings cannot exploit this concurrency fully, leaving plenty of headroom to improve training throughput.

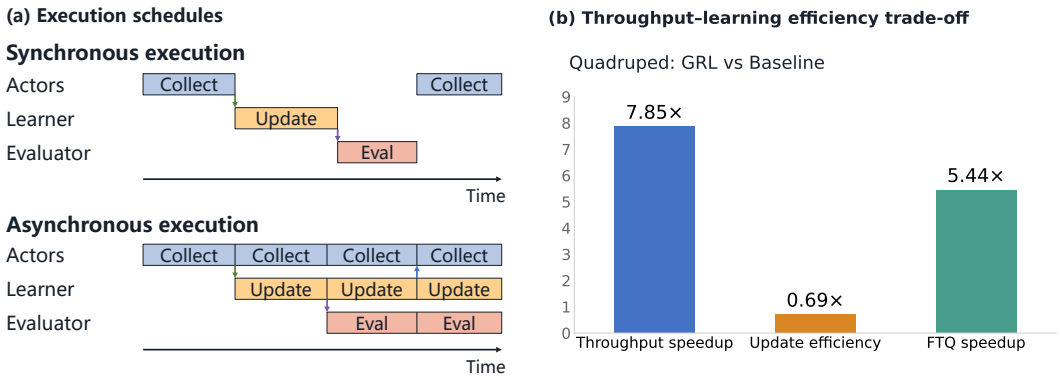


Figure 1: Execution and learning-time motivation. (a) Synchronous and asynchronous execution. (b) GRL versus CPU–GPU Async on the environment Quadruped; the baseline uses asynchronous CPU actors and a GPU learner. Update efficiency is the baseline/GRL ratio of target-reaching update counts; the value below one means GRL needs more updates. Throughput speedup is GRL/baseline, whereas FTQ speedup is baseline/GRL, excluding configuration cost (Section 6.5).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To better utilize GPU resources and improve training throughput, prior GPU-native RL systems colocate simulation and policy training on the GPU and exploit parallel environments, reducing CPU–GPU transfers and exposing more parallel work (Makoviychuk et al., 2021; Lan et al., 2022). But as Figure 1(a) illustrates, synchronous execution leaves collection, learning, and evaluation largely non-overlapped. GRL targets this remaining concurrency through asynchronous cross-role execution while coordinating data handoff, resource contention, and synchronization.

However, another issue is whether higher throughput translates into faster learning. Execution organization can change experience timing, reuse, and policy lag—the delay between learner and actor policies (Espeholt et al., 2018; Kapturowski et al., 2019)—thereby altering the number of updates needed to reach a target quality. We therefore measure first time-to-quality (FTQ): elapsed time until the evaluated reward first reaches the target. This metric combines execution throughput and learning quality, representing the overall learning speed. Figure 1(b) illustrates this tradeoff: GRL increases throughput but needs more learner updates to reach the target reward. The throughput gain outweighs this added work, yielding an overall FTQ speedup.

Prior work has studied this tradeoff through sensitivity analysis and empirical tuning, and has introduced runtime mechanisms such as rate control and off-policy correction to mitigate its effects (Babaeizadeh et al., 2017; Espeholt et al., 2018; Petrenko et al., 2020; Li et al., 2023). However, these analyses and mechanisms have not been developed into a unified procedure for automatically determining execution parameters before training; parameter selection still largely relies on predefined settings or empirical tuning.

To address these challenges, we present GRL, a GPU execution framework with learning-aware configuration for replay-based asynchronous RL. GRL combines hierarchical parallelism and GPU-resident data handoff to increase throughput, while selecting execution settings with limited search overhead. GRL preserves algorithm-defined losses, critic organization, and update dependencies within the learner. By organizing collection, data delivery, and evaluation around these learner-defined updates, GRL provides a common execution workflow for different learner structures without redesigning the underlying RL algorithms.

Our contributions are threefold. **First,** we develop a GPU execution workflow that combines hierarchical parallelism across actors, the learner, and the evaluator with GPU-resident pipelined data handoff, while preserving algorithm logic and dependencies (Section 4). **Second,** we introduce a low-overhead learning-aware configurator that screens candidates by throughput and compares their quality after equal-time training, using a bounded search to limit selection overhead (Section 5). **Third,** experiments across eight environments on GH200 show end-to-end speedups of up to $7.85\times$ over CPU–GPU Async and $4.11\times$ over GPU-Sync, including configuration cost across ten runs, and analyze how throughput, learning progress, and selection cost affect these gains (Section 6).

2 Related Work

RL execution. Prior RL systems mainly follow two representative execution paradigms. CPU–GPU asynchronous systems use CPU actors with a GPU learner to decouple collection and optimization (Espeholt et al., 2018; Horgan et al., 2018; Kapturowski et al., 2019; Petrenko et al., 2020), while GPU-native systems colocate simulation and learning on the GPU and typically alternate parallel collection and optimization synchronously (Lan et al., 2022; Makoviychuk et al., 2021; Freeman et al., 2021). Other frameworks mainly provide evaluation, modular components, or execution abstractions (Hoffman et al., 2020; Liang et al., 2018; Bou et al., 2024; Zhu et al., 2023). These two execution paradigms motivate our CPU–GPU Async and GPU-Sync baselines.

GPU-resident RL systems also explore different organizations of learning and execution. PQL separates collection, policy learning, and value learning into independently scheduled components coordinated through learner-local replay buffers and rate control (Li et al., 2023). PQN uses synchronous parallel sampling with regularized online TD learning, without a large replay buffer or target network (Gallici et al., 2025). GRL instead preserves the learner's algorithm-defined update structure while reorganizing collection, data delivery, and evaluation around it. Our evaluation asks how execution organization and configuration selection affect FTQ when the learning algorithm is held fixed. We therefore use CPU–GPU Async and GPU-Sync baselines, together with execution and configuration ablations.

Configuration for efficient learning. Related work improves training efficiency from different directions. Hyperband and PBT focus on learning-oriented hyperparameter search and resource allocation (Li et al., 2018; Jaderberg et al., 2017); PER, ACER, and Reverb study experience replay, off-policy correction, and data reuse (Schaul et al., 2016; Wang et al., 2017; Cassirer et al., 2021); while GA3C, OpenTuner, and Ansor tune concurrency or system execution performance (Babaeizadeh et al., 2017; Ansel et al., 2014; Zheng et al., 2020). PQL further coordinates training roles through runtime rate control (Li et al., 2023). These lines of work typically emphasize either learning efficiency or execution performance. In contrast, GRL targets configuration before training, jointly using measured throughput and learning quality under a limited budget, with the aim of obtaining practical training-time gains after accounting for configuration overhead.

3 Execution-Learning Formulation

We relate the two baseline schedules (Section 2) and GRL to execution throughput, then incorporate the learning work required for target attainment to formulate the FTQ configuration objective.

3.1 Execution Organization and Workflow Time

To compare execution schedules, consider fixed amounts of collection and evaluation work and W learner updates. Let $T_{A,m}$, $T_{L,m}$, and $T_{E,m}$ denote the corresponding processing times of actors, learner and evaluator under workflow m, excluding inter-role waiting. These times depend on each workflow's device placement and work grouping and are not assumed equal across workflows. We use Δ_m to represent additional exposed handoff and coordination costs, including synchronization and pipeline startup or drain effects.

GPU-synchronous execution. GPU-Sync performs collection and learner updates on the GPU but alternates them and pauses training for evaluation. Its workflow time is therefore represented as

$$T_{\text{GPU-Sync}} \approx T_{A,\text{sync}} + T_{L,\text{sync}} + T_{E,\text{sync}} + \Delta_{\text{sync}}. \tag{1}$$

GPU residency reduces cross-device data movement, but sequential scheduling leaves collection, learning, and evaluation on the same execution path.

CPU–GPU asynchronous execution. CPU–GPU Async overlaps CPU-based collection with GPU learner updates, while pausing both for CPU-based evaluation. Under idealized overlap between collection and learning, its workflow time is represented as

$$T_{\text{CPU–GPU Async}} \approx \max(T_{A,\text{async}}, T_{L,\text{async}}) + T_{E,\text{async}} + \Delta_{\text{async}}. \tag{2}$$

This organization exploits actor–learner concurrency, but throughput can still be limited by CPU-side collection and exposed cross-device handoff costs. Evaluation remains outside the overlapped training path.

3.2 From GPU Parallelism to Execution Throughput

GRL addresses these limitations by combining GPU-resident execution with overlap among actors, the learner, and the evaluator. With sufficient pipeline overlap, its time can be represented as

$$T_{\text{GRL}} \approx \max(T_{A,\text{GRL}}, T_{L,\text{GRL}}, T_{E,\text{GRL}}) + \Delta_{\text{GRL}}. \tag{3}$$

The maximum term represents an idealized overlap reference. Shared compute, memory, and bandwidth can increase processing times, while data dependencies and synchronization can leave additional costs exposed.

GRL combines hierarchical parallelism and GPU-resident pipelined handoff to improve the modeled execution time. Actor-side batching targets $T_{A,\text{GRL}}$, while cross-role concurrency seeks the idealized overlap represented by the maximum term in Equation 3. Pipelined handoff targets the exposed handoff component of Δ_{GRL} (Section 4).

For an execution setting c, learner-update throughput is $U(c) = W/T(c)$ in updates/s, where $T(c)$ measures the time to complete W updates. Reducing this time increases throughput, but reaching a quality target may require different amounts of work across configurations.

3.3 From Throughput to Time-to-Quality

Execution choices can change experience ordering, freshness, and reuse, thereby changing the learner work required for target attainment. Let $N_q(c)$ denote the number of learner updates required to reach target reward q, and $T_q(c)$ the corresponding training time. FTQ can be written as

$$T_q(c) = \frac{N_q(c)}{U(c)}. \tag{4}$$

The configuration objective must therefore account for both execution rate and required learning work:

$$c^* \in \arg\min_c T_q(c). \tag{5}$$

Measuring every candidate's FTQ requires full training runs, whose cost may outweigh subsequent savings. We therefore seek useful configurations through a limited search rather than exhaustive optimization.

4 GRL Training Workflow

4.1 Workflow Overview

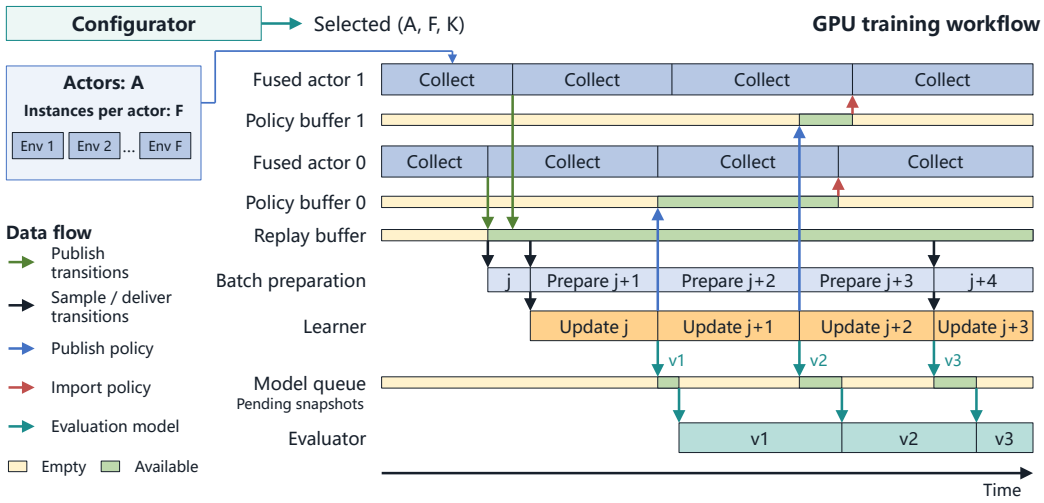


Figure 2: GRL training workflow. Indices j label learner updates and their batches; v_i labels evaluation snapshots. Each of the A actors batches F environments and publishes transitions to the replay buffer. The learner consumes sampled batches, publishes policy versions to actors, and sends model snapshots to the evaluator. Actor execution, batch preparation, learner updates, and evaluation can proceed concurrently through GPU-resident buffers and model queue.

Following Section 3, GRL combines hierarchical GPU execution and GPU-resident pipelined handoff around algorithm-defined learner updates (Figure 2). Its execution setting is $c = (A, F, K)$: A is the number of independently progressing actors, F is the number of environments batched per actor (fusion degree), and K is the configured target transition reuse ratio.

4.2 Hierarchical GPU Execution

The first two controls, A and F, target the actor-side execution time and cross-role overlap identified in Section 3. Within each actor, F independent environment instances are fused into the same GPU kernels at thread or thread-block granularity, depending on the environment implementation. This increases intra-kernel simulation parallelism and batches observations for policy inference (Makoviychuk et al., 2021; Lan et al., 2022). Across roles, the A actors, the learner, and the evaluator run on separate CUDA streams and can advance concurrently when their data dependencies permit. Because F controls within-actor batching and A controls actor concurrency, the same total environment count $A \times F$ can yield different parallelism and resource contention.

4.3 STREAMLINED GPU-RESIDENT DATA EXCHANGE

The concurrency exposed in Section 4.2 can still be limited by data handoff between roles. GRL therefore uses three GPU-resident structures—the replay buffer, per-actor policy buffers, and the model queue—to coordinate experience, policy, and evaluation-model delivery.

For experience delivery, the replay buffer is partitioned into regions assigned to each actor. Actors publish completed regions, while the learner samples only from available regions. Sampled transitions are copied into one of two learner-private batch buffers: the source region can be released once the copy completes, while the private batch remains protected until learner consumption finishes. The two private buffers allow preparation of the next batch to overlap the current update.

For policy delivery, each actor has a dedicated GPU-resident policy buffer. The learner visits these buffers in round-robin order and publishes a new policy when the target buffer is available; otherwise, that publication opportunity is skipped without blocking the learner. Each actor copies an available version into actor-private storage before subsequent inference. This publication schedule also realizes the configured target transition reuse ratio K. If one actor collection round produces D transitions and the learner batch size is B, GRL schedules a publication opportunity every $I_{\mathrm{pub}} = KD/B$ learner updates, corresponding to KD learner samples per collection round. After initial replay filling, an actor must acquire a newly published policy before beginning its next collection round; if none is available, it waits. This back-pressure couples experience generation to learner progress and keeps the workflow near the reuse operating point specified by K.

For evaluation, the learner places model snapshots into a GPU-resident model queue. The evaluator consumes these snapshots independently and evaluates each fixed policy concurrently with other roles. Detailed synchronization, publication, and reuse accounting are provided in Appendix C.

5 LEARNING-AWARE EXECUTION CONFIGURATOR

Guided by the FTQ objective in Section 3.3, GRL uses short throughput probes to narrow the search before learning-quality trials. This limits costly trial training to a small candidate set. For each candidate target reuse ratio K, GRL estimates the balance between actor generation and learner consumption. Let $g_1(F)$ be the standalone transition-generation rate of one actor with fusion degree F, and ℓ the standalone learner transition-consumption rate. The estimated supply–consumption ratio is

$$\rho(A, F, K) = \frac{AKg_1(F)}{\ell}. \tag{6}$$

At the target reuse ratio K, the estimated actor supply $Ag_1(F)$ corresponds to a sampling-work rate of $AKg_1(F)$. Thus, $\rho < 1$ suggests that actor generation may limit learner throughput at this reuse level. Increasing A or F can alleviate this limitation, but additional parallelism also introduces GPU contention. This tradeoff motivates starting the search near the estimated balance: GRL chooses the allowed (A, F) pair with ρ closest to one, then tests its neighborhood under concurrent execution.

Joint probes account for contention and scheduling effects absent from standalone measurements. GRL performs one search round at the center and up to four axial neighbors, obtained by changing either A or F to its next lower or higher candidate value, without recentering. For each K, it retains

$$\hat{c}_K \in \arg\max_{c \in \mathcal{S}_K} U(c), \tag{7}$$

where $\mathcal{S}_K$ is the measured set and $U(c)$ is learner-update throughput over the joint-probe interval.

To compare learning progress across the retained configurations, each $\hat{c}_K$ is trained from scratch for the same wall-clock budget τ. Its endpoint quality $\widehat{Q}_\tau(\hat{c}_K)$ is the mean return over ten evaluation episodes with the final policy held fixed. GRL selects the representative with the highest endpoint quality. Equal-time comparison reflects both execution rate and learning progress.

Quality trials reveal learning differences that throughput probes cannot capture, but add selection cost. For short training workloads, this cost may outweigh subsequent FTQ savings, motivating an option to skip quality comparison. A switch q_{on}, fixed before selection, controls this tradeoff: $q_{\mathrm{on}} = 1$ retains endpoint-quality comparison, whereas $q_{\mathrm{on}} = 0$ uses throughput screening at a preset K. In our evaluation, this preset is $K = 4$, and the latter branch returns $\hat{c}_4$.

Algorithm 1 Learning-aware execution configuration

Require: Candidate sets for A, F, and K; quality-trial budget τ; quality-search switch q_{on}

If $q_{\text{on}} = 0$, restrict the candidate reuse ratios to $\{4\}$.
Measure the standalone learner consumption rate ℓ.
Measure the standalone single-actor generation rate $g_1(F)$ for each candidate F.
for each candidate K **do**
  Choose the allowed (A, F) with $\rho(A, F, K)$ closest to one.
  Jointly probe the center and its in-grid axial neighbors once to form $\mathcal{S}_K$.
  Retain $\hat{c}_K \in \arg\max_{c \in \mathcal{S}_K} U(c)$.
end for
if $q_{\text{on}} = 1$ **then**
  Train each retained $\hat{c}_K$ from scratch for wall-clock budget τ.
  Evaluate $\widehat{Q}_\tau(\hat{c}_K)$ for each retained configuration.
  Select $c_{\text{sel}} \in \arg\max_{\hat{c}_K} \widehat{Q}_\tau(\hat{c}_K)$.
else
  Set $c_{\text{sel}} = \hat{c}_4$.
end if
Record the total configuration cost C_{select}.
Return c_{sel}.

The selected c_{sel} is reused across repeated training runs under a fixed setup, with C_{select} counted once as the configuration overhead. Section 6.1 specifies the candidate spaces and quality-stage settings; Appendix B gives the detailed search and cost-accounting protocol.

6 Experiments

We evaluate whether GRL translates GPU parallelism into practical FTQ gains through workflow comparisons, execution ablations, and configuration-quality and cost analyses. Case studies then connect these gains to execution throughput and required learner updates.

6.1 Experimental Setup

We evaluate eight environments spanning discrete/continuous action spaces and single-agent/multi-agent RL tasks (Towers et al., 2024; Farid et al., 2025; Yan et al., 2024; Tassa et al., 2018). DQN, DDPG (including parameter-shared DDPG, or PS-DDPG, for multi-UAV), and TD3 (Mnih et al., 2015; Lillicrap et al., 2016; Fujimoto et al., 2018) exercise different learner structures within the same replay-based workflow. Table 1 summarizes the algorithm and per-instance GPU execution granularity for each environment.

All experiments use one NVIDIA GH200 Grace Hopper Superchip (NVIDIA, 2026), with a Hopper GPU containing 96 GB HBM3 and a 72-core Arm Neoverse V2 Grace CPU. GRL and both baselines use C++ and CUDA 13.0 with the same environment implementations; Walker-Run, Quadruped, and Fish-Swim use our custom CUDA physics kernels.

Table 1: Environment and algorithm settings.

Environment	Algorithm	Threads/env.
CartPole	DQN	1
single-UAV	DDPG	1
Navigation	DQN	1
Acrobot	DQN	1
multi-UAV	PS-DDPG	128
Walker-Run	TD3	32
Fish-Swim	TD3	32
Quadruped	TD3	32

The configurator uses $A \in \{1, 2, 4, 8, 16\}$ for all environments. For the first four environments in Table 1, we use $F \in \{16, 64, 256, 512\}$, prescribe $K = 4$, and disable the quality stage ($q_{\text{on}} = 0$). For the remaining four environments, we use $F \in \{1, 2, 4, 8\}$ and $K \in \{1, 2, 4\}$ with the quality stage enabled ($q_{\text{on}} = 1$).

All methods share a fixed quality target for each environment. For each environment, q is the equal-weight mean over the final five checkpoints of ten synchronous reference runs, with each

checkpoint evaluated over ten fixed evaluation seeds. Reference runs use the same algorithm and non-configuration training hyperparameters, $K = 1$. Targets are fixed before configuration selection and workflow comparison.

FTQ follows a common evaluation and stopping protocol across all three workflows. Each evaluation holds the policy fixed and averages ten episode returns, and checkpoints are generated at the same learner-update interval. FTQ is measured from run initialization until the first qualifying evaluation result becomes available. Runs that exhaust the update budget without reaching the target are marked NR (not reached) after all queued evaluations finish. We report cumulative FTQ over ten main training seeds; when quality search is enabled, its trial uses a separate training seed. Configuration-selection cost is accounted for separately where reported. Additional experimental settings are provided in Appendix A, and configurator details in Appendix B.

6.2 End-to-End Workflow Comparison

We compare GRL against two execution baselines while fixing the algorithms, networks, environment logic, and non-configuration learning hyperparameters. To isolate execution organization, *CPU–GPU Async* uses CPU actors with a GPU learner and adopts GRL's selected (A, F, K). *GPU-Sync* alternates GPU-based collection and learner optimization in a single synchronized group. Because it has no independent actor concurrency, we set $F_{\text{sync}} = A_{\text{sel}} F_{\text{sel}}$ to match GRL's total environment count. All three workflows therefore use the same selected K: $K = 2$ for Walker-Run and Quadruped and $K = 4$ for the other environments.

Each evaluation averages ten episodes executed concurrently. GRL overlaps GPU evaluation with training, whereas both baselines pause training during evaluation; GPU-Sync evaluates on the GPU and CPU–GPU Async on the CPU. The implementation details are provided in Appendix C.2.

Figure 3 reports cumulative FTQ over the same ten training seeds for all three workflows. To account for the cost of configuration before training, GRL additionally includes the one-time selection cost C_{select} once per environment; Figure 3 distinguishes this cost from training FTQ.

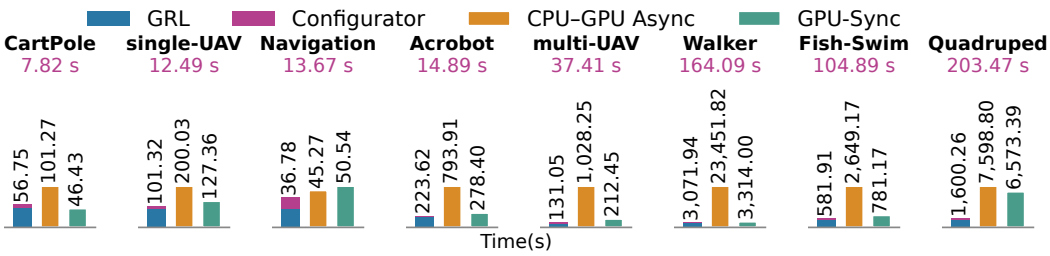


Figure 3: Cumulative time over ten training seeds (seconds). Bar-top values give total time. Purple segments and purple numbers beneath environment names show GRL's one-time configuration-selection cost, included in its total.

Including configuration cost, GRL achieves speedups of 1.23–7.85× over CPU–GPU Async across all eight environments. Compared with GPU-Sync, GRL is faster in seven environments, with speedups of 1.08–4.11×, but is slower on CartPole: 56.75 s versus 46.43 s. CartPole's lightweight workload leaves little room for additional overlap, so GRL remains slightly slower than GPU-Sync even before configuration cost. Appendix D reports the per-seed FTQ statistics and learning curves.

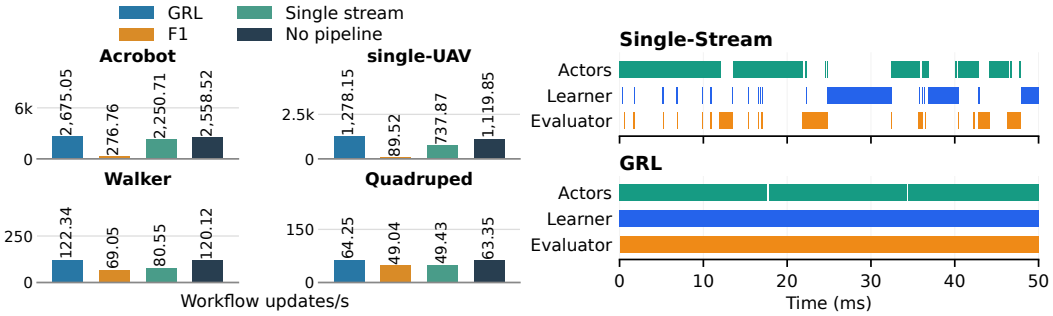
6.3 Ablation Study on Throughput

To verify that the execution mechanisms in Section 4 translate into the throughput gains targeted by the model in Section 3, we ablate actor-side fusion, cross-role concurrency, and pipelined handoff across four environments. *F1* sets $F = 1$; *Single-Stream* serializes all roles; *No Pipelined Handoff* disables batch prefetching, synchronizes policy delivery, and waits for evaluations. Within each comparison, other settings remain fixed, and variants complete identical workloads.

Figure 4(a) summarizes the throughput effects of the three execution mechanisms. Fusion improves workflow throughput by 9.67–14.28× on the two single-thread environments and 1.31–1.77× on the

two multi-thread environments. The larger gains on the single-thread environments are consistent with their lower per-instance parallelism, for which fusion exposes more GPU work per launch. Cross-role concurrency provides additional speedups of 1.19–1.73× across all four environments. Pipelined handoff contributes 1.05–1.14× speedups on the single-thread environments and smaller but consistent gains of 1.01–1.02× on the multi-thread environments, where longer actor operations better amortize synchronization overhead.

To examine the execution behavior associated with these concurrency gains, Figure 4(b) compares GRL and Single-Stream on Walker-Run using NVIDIA Nsight Systems (NVIDIA). Wall time with kernels from at least two roles increases from 0% to 88.92%, while kernel-active time rises from 94.22% to 99.53%. These profiles show cross-role overlap and reduced gaps in kernel activity, complementing the throughput gains observed in the concurrency ablation.



(a) Throughput comparison

(b) Cross-role execution overlap

Figure 4: GRL ablation study. (a) Throughput averaged over three measurements. (b) Walker-Run kernel timelines within a 50 ms profiling window: green, blue, and orange denote actor, learner, and evaluator kernels, respectively; learner includes batch preparation.

6.4 Configurator Evaluation

We compare Algorithm 1 against two alternatives. *Max* tests whether maximum parallelism suffices, using $(A, F, K) = (16, 512, 4)$ for the first four environments and $(16, 8, 4)$ for the remaining four. *Random* tests the value of structured search by independently sampling one configuration per training seed from the same candidate space. We also isolate quality selection and assess whether training-time savings recover the one-time cost C_{select}, included where reported.

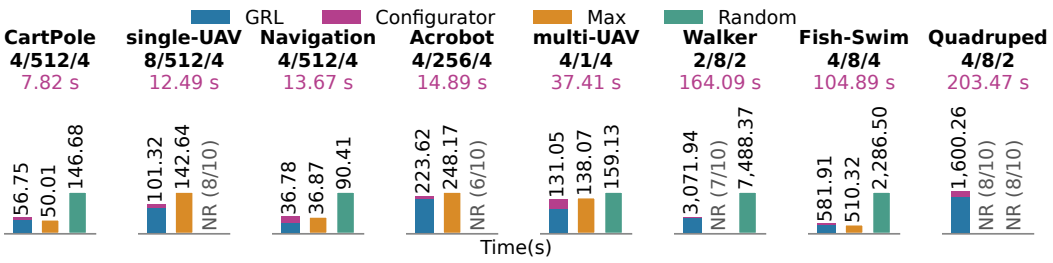


Figure 5: Configuration comparison over ten training seeds. Configurations are labeled (A, F, K). Bar-top values give total time in seconds; purple labels give GRL's one-time configuration cost, included in its total. Fractions after NR show target-reaching runs; all other settings reach the target on 10/10 seeds.

GRL reaches the target on all ten seeds per environment; Max and Random incur NR runs (Figure 5). Appendix E reports FTQ over their common target-reaching seeds. Across complete ten-seed comparisons, GRL's pure-FTQ speedups are 1.70–4.79× over Random and 1.02–1.61× over Max. Including selection cost, these become 1.21–3.93× and 0.88–1.41×, retaining an advantage in four of six Max comparisons.

Ablating quality-based selection. Quality selection changes the choices for Walker-Run and Quadruped. GRL selects $(2, 8, 2)$ and $(4, 8, 2)$, respectively, and $K = 4$ in the other six environments. No-quality omits quality trials and selects the highest-probe-throughput representative across all candidate K. It selects $(2, 8, 4)$ for both tasks and agrees with GRL on other environments. On Walker-Run, ten-seed cumulative FTQ is 2,907.86 s for GRL and 3,220.27 s for No-quality, a $1.11\times$ speedup excluding selection cost. On Quadruped, quality selection raises target attainment from 5/10 to 10/10 seeds. These comparisons evaluate the selected configurations across ten training seeds without repeating selection. Table B.2 gives the representatives, probe throughput, and endpoint rewards underlying these choices. The amortization analysis of configuration cost is provided in Appendix E.2.

6.5 Where Do Time-to-Quality Gains Come From?

To connect the end-to-end results back to the FTQ formulation in Section 3.3, we examine how learner-update throughput and the number of updates required for target attainment jointly account for the observed FTQ differences.

Excluding configuration-selection cost, N_q and T_q in this subsection denote means over the ten seeds. For the compared method m and reference r, aggregate learner-update throughput is the ratio of these means: $U^x = N_q^x / T_q^x$ for $x \in \{m, r\}$. We define

$$S_U = \frac{U^m}{U^r}, \qquad S_N = \frac{N_q^r}{N_q^m}, \qquad S_T = \frac{T_q^r}{T_q^m} = S_U S_N, \tag{8}$$

Here, $S_N > 1$ indicates that method m reaches the target with fewer learner updates than reference r. Table 2 illustrates three ways that execution rate and required updates combine to determine FTQ.

For single-UAV, changing only F from 1 to 512 yields a $13.00\times$ throughput ratio and a $1.11\times$ required-update ratio, together accounting for the $14.45\times$ FTQ speedup. The gain is dominated by the execution-rate improvement, consistent with the fusion ablation in Section 6.3.

Table 2: Case-study FTQ speedup decomposition.

Case	S_U	S_N	S_T
single-UAV			
GRL / F1	13.00	1.11	14.45
Quadruped			
GRL / CPU–GPU Async	7.85	0.69	5.44
multi-UAV			
GRL / Max	0.96	1.54	1.47

For Quadruped, GRL requires more learner updates than CPU–GPU Async ($S_N = 0.69$), but its $7.85\times$ higher learner-update throughput more than compensates for the additional learning work, yielding a $5.44\times$ FTQ speedup after excluding configuration cost.

For multi-UAV, fewer required updates dominate instead. Compared with Max at $(A, F, K) = (16, 8, 4)$, GRL has slightly lower throughput ($S_U = 0.96$), but its $1.54\times$ required-update advantage yields a $1.47\times$ FTQ speedup.

7 Conclusion

GRL combines GPU concurrency and pipelined data exchange with learning-aware configuration for replay-based asynchronous RL, preserving algorithm-defined learner-update dependencies. Throughput screening and optional quality trials select execution settings with limited overhead. Across eight environments on GH200, GRL achieves end-to-end speedups of up to $7.85\times$ over CPU–GPU Async and $4.11\times$ over GPU-Sync. Faster execution does not always yield proportional FTQ gains, and selection cost can outweigh training-time savings, highlighting the need to consider execution rate, learning progress, and configuration cost together.

REPRODUCIBILITY STATEMENT

Reproducibility information is provided in the main experimental setup and Appendices A–E in the supplementary document. Section 6.1 documents the hardware, software stack, tasks, and experimental protocol. Appendix A specifies environment definitions, target rewards, network architectures, training hyperparameters, exploration schedules, update rules, and evaluation, stopping, and seed protocols. Appendix B describes the configurator's search procedure, probe and quality-trial durations, and selection-cost accounting. Appendix C details policy publication, target reuse, GPU buffer synchronization, evaluator execution, and FTQ accounting. Appendices D and E provide ten-seed learning curves and FTQ summary statistics, supplementary configurator comparisons, and configuration-cost amortization.

AI USE STATEMENT

This manuscript was prepared with assistance from AI tools for literature search, language translation, grammar and style refinement, and improving the presentation of figures and visual materials. The authors have verified all references against their sources to ensure that no citations are fabricated. All technical content, experimental results, and conclusions are verified by the authors, who take full responsibility for the final manuscript.

REFERENCES

Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. OpenTuner: An extensible framework for program autotuning. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2014. URL https://commit.csail.mit.edu/papers/2014/ansel-pact14-opentuner.pdf.

Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a GPU. In *ICLR*, 2017. URL https://openreview.net/pdf?id=r1VGvBcxl.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. TorchRL: A data-driven decision-making library for PyTorch. In *ICLR*, 2024. URL https://arxiv.org/abs/2306.00577.

Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, and Manuel Kroiss. Reverb: A framework for experience replay. *arXiv preprint arXiv:2102.04736*, 2021. URL https://arxiv.org/abs/2102.04736.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Vlad 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 *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1407–1416. PMLR, 2018.

Ghulam Farid, Muhammad Bilal, Lanyong Zhang, Ayman Alharbi, Ishaq Ahmed, and Muhammad Azhar. An improved deep q-learning approach for navigation of an autonomous uav agent in 3d obstacle-cluttered environment. *Drones*, 9(8):518, 2025. ISSN 2504-446X. doi: 10.3390/drones9080518. URL https://www.mdpi.com/2504-446X/9/8/518.

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 *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/hash/d1f491a404d6854880943e5c3cd9ca25-Abstract-round1.html.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1587–1596, 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c23f3852601f6dd7f0b39223d031806f-Abstract-Conference.html.

Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, Léonard Hussenot, Robert Dadashi, Gabriel Dulac-Arnold, Manu Orsini, Alexis Jacq, Johan Ferret, Nino Vieillard, Seyed Kamyar Seyed Ghasemipour, Sertan Girgin, Olivier Pietquin, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Abe Friesen, Ruba Haroun, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1803.00933.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019.

Tian Lan, Sunil Srinivasa, Huan Wang, and Stephan Zheng. WarpDrive: Fast end-to-end deep multi-agent reinforcement learning on a GPU. *Journal of Machine Learning Research*, 23(316): 1–6, 2022. URL https://jmlr.org/papers/v23/22-0185.html.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL https://www.jmlr.org/papers/v18/16-558.html.

Zechu Li, Tao Chen, Zhang-Wei Hong, Anurag Ajay, and Pulkit Agrawal. Parallel Q-learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19440–19459. PMLR, 2023. URL https://proceedings.mlr.press/v202/li23f.html.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. URL https://proceedings.mlr.press/v80/liang18b.html.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016. URL https://arxiv.org/abs/1509.02971.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High performance GPU-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937. PMLR, 2016.

NVIDIA. *NVIDIA Nsight Systems User Guide*. URL https://docs.nvidia.com/nsight-systems/UserGuide/index.html. Accessed September 26, 2026.

NVIDIA. *Grace Performance Tuning Guide*, 2026. URL https://docs.nvidia.com/dccpu/grace-perf-tuning-guide/. Accessed September 20, 2026.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3D control from pixels at 100000 FPS with asynchronous reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. URL https://proceedings.mlr.press/v119/petrenko20a.html.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.05952.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 387–395. PMLR, 2014.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite, 2018. URL https://arxiv.org/abs/1801.00690.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1611.01224.

Chao Yan, Chang Wang, Xiaojia Xiang, Kin Huat Low, Xiangke Wang, Xin Xu, and Lincheng Shen. Collision-avoiding flocking with multiple fixed-wing uavs in obstacle-cluttered environments: A task-specific curriculum- based madrl approach. *IEEE Transactions on Neural Networks and Learning Systems*, 35(8):10894–10908, 2024. doi: 10.1109/TNNLS.2023.3245124.

Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation*, 2020. URL https://www.usenix.org/conference/osdi20/presentation/zheng.

Huanzhou Zhu, Bo Zhao, Gang Chen, Weifeng Chen, Yijie Chen, Liang Shi, Yaodong Yang, Peter Pietzuch, and Lei Chen. MSRL: Distributed reinforcement learning with dataflow fragments. In *2023 USENIX Annual Technical Conference*, 2023. URL https://www.usenix.org/conference/atc23/presentation/zhu-huanzhou.