FROM DISCOVERY TO MASTERY: REALLOCATING THE REPLAY BUDGET IN MASSIVELY PARALLEL REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Massively parallel interaction makes rare successful behavior available for learning, yet discovering these behaviors does not ensure that a policy learns to reproduce them. We identify a *discovery–mastery gap*: policies learn to reliably reproduce discovered high-quality behaviors only after substantial delays or fail to do so within the training budget. We find that increasing the replay share of successful experience can accelerate learning and enable task success without increasing interaction or update budgets, suggesting that insufficient exposure to such experience contributes to this gap. Motivated by these findings, we propose **r**eturn-guided **rea**llocation of the re**p**lay budget (**REAP**), a generic sampling mechanism that mixes a base sampling distribution with one proportional to positive centered return-to-go. Its return-based scores are computed once from observed rewards when each episode ends and remain fixed as training proceeds. Across 48 control tasks, REAP enables learning on challenging sparse-reward tasks where baselines struggle and substantially improves sample efficiency on the remaining tasks.

1 INTRODUCTION

Massively parallel simulation allows reinforcement learning (RL) agents to interact with thousands of environments on a single GPU (Makoviychuk et al., 2021). By keeping simulation and learning on the same device, these systems have enabled locomotion training in minutes and extended large-scale learning to a growing range of manipulation and whole-body control tasks (Rudin et al., 2022; Tao et al., 2025; Mittal et al., 2025; Zakka et al., 2025). Recent off-policy systems pair this throughput with exploration mechanisms that promote diverse and temporally coherent behavior (Li et al., 2023; Seo et al., 2025; Kim et al., 2026). Together, massive parallelism and these exploration machanisms substantially increase the likelihood of discovering successful behavior.

However, discovering successful behaviors does not guarantee that the learned policy will reliably reproduce them. We observe settings in which successful trajectories enter the replay buffer while policy performance remains near zero, with improvement emerging only after millions of additional interactions or failing to emerge within the training budget. We refer to this phenomenon as the *discovery–mastery gap*: successful behaviors are already present in the collected experience, yet reliable policy performance emerges much later or does not emerge within the training budget. We observe this gap under both sparse and dense rewards.

A possible explanation for this gap is that successful experience receives insufficient replay during training. Under uniform replay, the probability of sampling successful experience matches its fraction of the buffer, so rare successes also remain rare in the sampled batches (Figure 1, left). Recent work shows that favoring recent experience helps with learning when replay volume is limited (Korniak et al., 2026). Recency, however, provides only a coarse criterion for replay allocation and does not directly indicate which transitions help the policy learn to reproduce discovered high-quality behaviors. This motivates our central question:

> RQ: *How does replay allocation affect whether and how quickly policies learn to reliably reproduce discovered high-quality behaviors in massively parallel reinforcement learning?*

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.

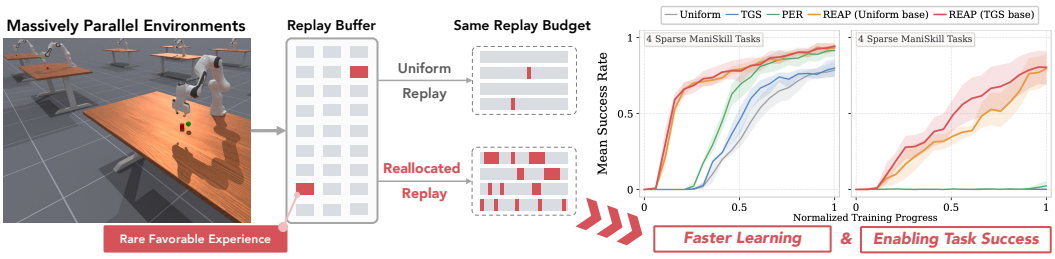

Figure 1: **From discovery to mastery through replay reallocation.** Left: REAP increases the sampling probability of rare high-return experience (red) already in the buffer, without increasing the replay budget. Right: on eight sparse-reward ManiSkill tasks, REAP accelerates learning on four tasks that uniform replay already learns and enables success on four where uniform replay makes little progress. Collection and update budgets are unchanged across methods.

Motivated by prior work on learning from past successful behavior (Oh et al., 2018; Ecoffet et al., 2021), we examine whether increasing replay of successful experience can narrow the discovery–mastery gap while keeping collection and update budgets fixed. Interventions at different stages of training accelerate learning in both sparse- and dense-reward settings and enable task success where uniform replay makes no progress. These results suggest that insufficient exposure to successful experience contributes to the gap and support a simple allocation principle: give successful experience more replay within the existing budget. To apply this principle across reward settings, we seek a signal that captures both successful outcomes and partial progress without requiring explicit success labels. The signal should also be simple to maintain when experience arrives rapidly and individual transitions receive limited replay. In prioritized experience replay (PER), temporal-difference (TD) errors depend on the critic and require repeated evaluation to reflect its changing estimates (Schaul et al., 2015; Lahire et al., 2021). Observed returns instead record the outcomes of collected experience and remain fixed as the critic changes, providing a simple basis for replay allocation.

We therefore present **r**eturn-guided **rea**llocation of the re**p**lay budget (**REAP**), a simple implementation of this allocation principle. REAP mixes a base sampling distribution with one proportional to positive centered return-to-go, giving higher-return experience additional replay. The return-based scores are computed once at the end of each episode. REAP can use uniform or recency-based replay as its base sampler and operate with standard off-policy updates across different reward settings.

With a single replay configuration on eight sparse-reward tasks, REAP enables success on four where uniform and recency-based replay remain near zero and accelerates learning on the remaining four (Figure 1, right). It also improves aggregate learning performance across 40 additional tasks spanning ManiSkill (Tao et al., 2025), Genesis (Authors, 2024), Isaac Lab (Mittal et al., 2025), and MuJoCo Playground (Zakka et al., 2025). Budget analyses show that REAP outperforms the baselines at their default budgets even with fewer updates or smaller batches, while uniform replay does not close the performance gap even with four times the default batch size. Effective replay allocation also improves training efficiency as the number of parallel environments increases.

Our key contributions are:

- We identify a *discovery–mastery gap* in massively parallel RL: policies learn to reliably reproduce discovered high-quality behaviors only after substantial delays or fail to do so within the training budget (Section 2.1).
- We show that increasing the replay share of successful experience can accelerate learning and enable task success under fixed interaction and update budgets (Section 2.2).
- We introduce REAP, a generic implementation of return-guided replay allocation, and demonstrate its benefits across 48 control tasks.

2 Understanding the Discovery–Mastery Gap

In this section, we first show that massively parallel interaction makes rare successful behavior available for learning, yet reliable task performance emerges much later or remains absent throughout training (Section 2.1). Building on this observation, we show that a simple success-guided realloca-

tion of the existing replay budget shortens this gap (Section 2.2). Together, these findings identify replay allocation as a bottleneck between discovery and mastery, and provide the foundation for the simple outcome-guided replay rule developed in the next section.

2.1 Parallel discovery does not ensure mastery

Rare success at scale. Massively parallel interaction expands the scale of trial and error, allowing an agent to attempt a task in thousands of environments simultaneously (Makoviychuk et al., 2021). Recent off-policy methods show that more diverse exploration helps make effective use of this scale (Li et al., 2023; Seo et al., 2025). We find that successful behavior can be discovered even before learning begins. Using randomly initialized FlashSAC policies (Kim et al., 2026) with the stochastic action-sampling rule used in training, we observe successful trajectories on seven of eight sparse-reward ManiSkill tasks (Tao et al., 2025) (as shown in Figure 2 (A)). With success probabilities on the order of 10^{-6} to 10^{-3}, large numbers of parallel attempts make these rare outcomes available for learning. However, a higher probability of discovering success does not imply a larger share of successful experience. For a fixed policy with episode success probability p, one independent episode from each of N environments gives a discovery probability of $1-(1-p)^N$, while the expected fraction of successful episodes remains p (Figure 2 (B)).

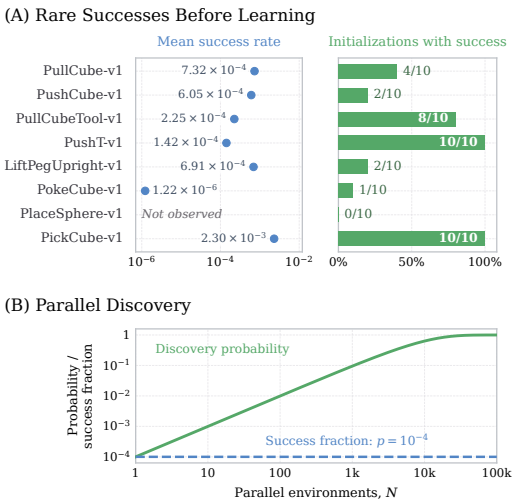

Figure 2: **Discovery at scale.** (A) Success rate under random-initialized policy over 10 runs. (B) Discovery probability and expected success fraction for a fixed policy.

Do discoveries translate into mastery? To understand how discovery translates into learning, we examine whether exploration can produce successful trajectories, whether successful experience is available in replay, and whether the policy learns to reproduce success reliably. Across representative sparse- and dense-reward tasks, we track these through stochastic evaluation, the fraction of successful episodes represented in replay, and deterministic evaluation, respectively. We follow FlashSAC training with uniform replay, evaluating the same frozen checkpoints both deterministically and with the stochastic action rule used in training. The full protocol is provided in Appendix A.

In tasks that are eventually learned, stochastic evaluation produces successes and successful experience is present in replay millions of environment steps before deterministic performance improves (Figure 3). This delay occurs across both sparse- and dense-reward tasks; in the latter, return also

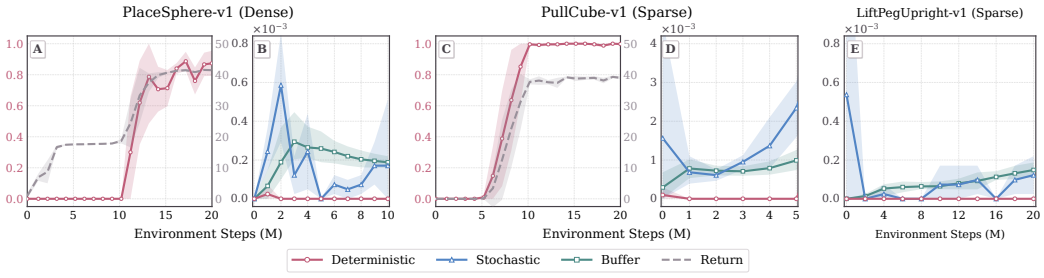

Figure 3: **Discovery does not ensure mastery.** (A,C) Full training curves under dense and sparse rewards. (B,D) Early diagnostics compare stochastic success rates, the fraction of successful episodes in replay, and deterministic success rates. (E) Successful experience remains available without deterministic success throughout training. Shading indicates 95% bootstrap confidence intervals.

stagnates before task success rises. More strikingly, successful experience can remain available throughout the entire 20M-step training budget while deterministic evaluation records no success.

Takeaway: Massively parallel interaction makes rare success available for learning, but its presence in replay does not ensure mastery. Reliable task performance can emerge much later or remain absent throughout the training budget.

2.2 Reallocating replay narrows the gap

When successful experience is already available in replay, a natural question is whether it receives enough opportunities to influence learning. Under uniform sampling, its replay share simply matches its fraction of stored transitions. We therefore test whether increasing this share, within the existing replay budget, can help the policy turn discovered successes into reliable task performance.

We investigate this question through controlled interventions on the same three tasks in Figure 3, where successful experience remains available in replay buffer. We introduce the same intervention at five checkpoints per task to examine its effect as the policy and the fraction of successful experience evolve over training. At each checkpoint, we restore the learner and buffer and use the task's success signal to draw half of the replay samples uniformly from transitions in successful trajectories and half uniformly from the full buffer. Online training continues with the original collection and update budgets. Full experimental details are provided in Appendix A.

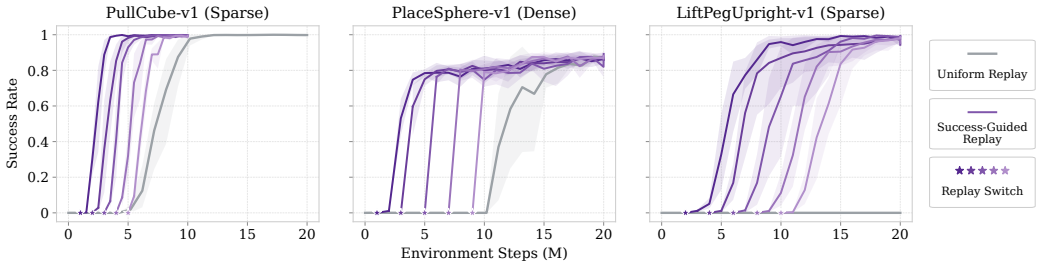

Figure 4: **Success-guided replay narrows the gap.** Gray curves use uniform replay; purple curves switch to success-guided replay at different steps, indicated by stars; darker shades denote earlier switches. Curves show means over five random seeds, with 95% bootstrap confidence intervals.

Across the tested checkpoints, the intervention speeds up learning where uniform replay eventually succeeds and achieves high success where it fails (Figure 4). Later switches remain effective after millions of steps without reliable success. These gains arise from changing replay with the same starting learner and buffer, under matched collection and update budgets. They identify replay allocation as a bottleneck between available experience and reliable task performance.

Takeaway: The transition from discovery to mastery can be accelerated by allocating a larger share of the existing replay budget to higher-return experience.

3 REAP: A Simple Implementation for Replay Allocation

We introduce **REAP** (***r**eturn-guided **rea**llocation of the rep**l**ay budget*) as a simple implementation of the allocation principle established in Section 2. On top of a base sampling distribution that covers every transition in the buffer, REAP increases the probability of sampling higher-return experience across reward settings. We develop REAP by first scoring experience using observed outcomes (Section 3.1), then using these scores to construct the replay distribution (Section 3.2).

3.1 Scoring experience with return-to-go

We seek a metric that identifies higher-return experience across tasks while remaining simple to maintain. Three considerations guide our choice. • ***General applicability.*** The score should use the task's existing rewards to reflect both successful outcomes and partial progress, allowing it to guide replay when success labels are unavailable or too coarse to distinguish different degrees of progress.

• ***Simple maintenance.*** When transitions arrive rapidly and receive limited replay (Korniak et al., 2026), the score should be computed once from observed experience and remain fixed as the learner changes. This avoids repeatedly recomputing TD-error priorities as the critic changes (Schaul et al., 2015; Lahire et al., 2021). • ***Richer outcome information.*** The score should use outcome information to identify a broader set of useful transitions, including transitions with little immediate reward that nevertheless lead to higher returns.

Return-to-go provides a simple score with these properties. It summarizes the discounted rewards from each transition onward, connecting earlier experience with later outcomes while allowing scores to differ within the same trajectory. To determine which experience should receive additional replay, we compare outcomes with a fixed reward reference. A large positive reward baseline can obscure differences between outcomes, while better outcomes may still have negative returns in tasks with negative rewards. We therefore center rewards around r_{ref}, the mean per-step reward during random-policy warmup, and keep this reference fixed throughout training. For transition i, let L_i denote the number of observed steps remaining in its episode, including the current step. We compute the centered return-to-go

$$G'_i = \sum_{k=0}^{L_i-1} \gamma^k \big(r_{i+k} - r_{\text{ref}}\big), \tag{1}$$

where γ is the discount factor. We use the positive part, $(G'_i)_+ = \max(G'_i, 0)$, as the replay score, assigning additional replay only to transitions with positive centered return-to-go. Computed once from observed rewards when an episode ends, the score remains fixed as the learner changes. We examine alternative outcome signals and score transformations in Section 4.3.

3.2 Allocating the Replay Budget

We combine return-guided sampling with a base sampler to increase replay of above-reference outcomes while keeping other experience available for learning.

Return-guided sampling. We sample transitions in direct proportion to $(G'_i)_+$. The buffer shown in Figure 5, collected from sparse-reward PullCube at 5M environment steps, illustrates why this changes replay allocation. Here, the replay scores equal the return-to-go because rewards take values in $\{0, 1\}$ and $r_{\text{ref}} = 0$. Positive-return transitions constitute only 2.4% of the buffer and are therefore rarely sampled under uniform replay. Truncated Geometric Sampling (TGS), which emphasizes recent experience (Korniak et al., 2026), samples these transitions more often, but still assigns over 90% of its sampling probability to zero-return transitions. Direct return guidance instead concentrates sampling on transitions with positive scores.

Mixing with a base sampler. Return-guided sampling concentrates replay on positive-score transitions, but using it alone would exclude the rest of the buffer. REAP therefore combines this distribution with a base sampler over the full buffer, keeping other experience available for value learning (Schaul et al., 2015). Let $\mathcal{D}$ be the replay buffer, $\mathcal{C} \subseteq \mathcal{D}$ its transitions with available scores, and p_i^{base} the base sampling distribution. REAP samples transition i with probability

$$p_i = \lambda \frac{(G'_i)_+}{\sum_{j \in \mathcal{C}} (G'_j)_+} + (1-\lambda) p_i^{\text{base}}, \qquad \lambda \in [0, 1]. \tag{2}$$

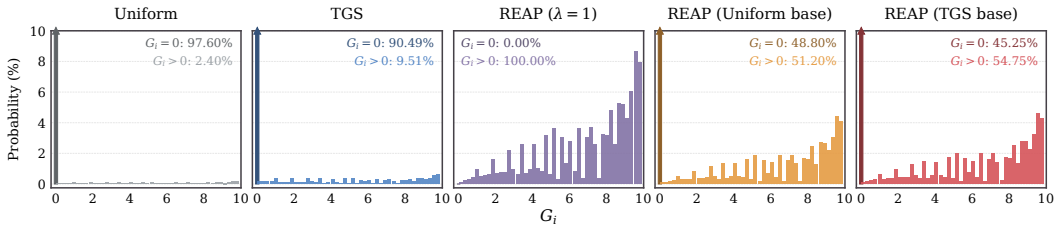

Figure 5: **REAP shifts sampling toward rare positive-return transitions.** Sampling probability mass by return-to-go for the same replay buffer. Positive-return transitions make up only 2.4% of the buffer but receive over half of the sampling probability under both REAP variants.

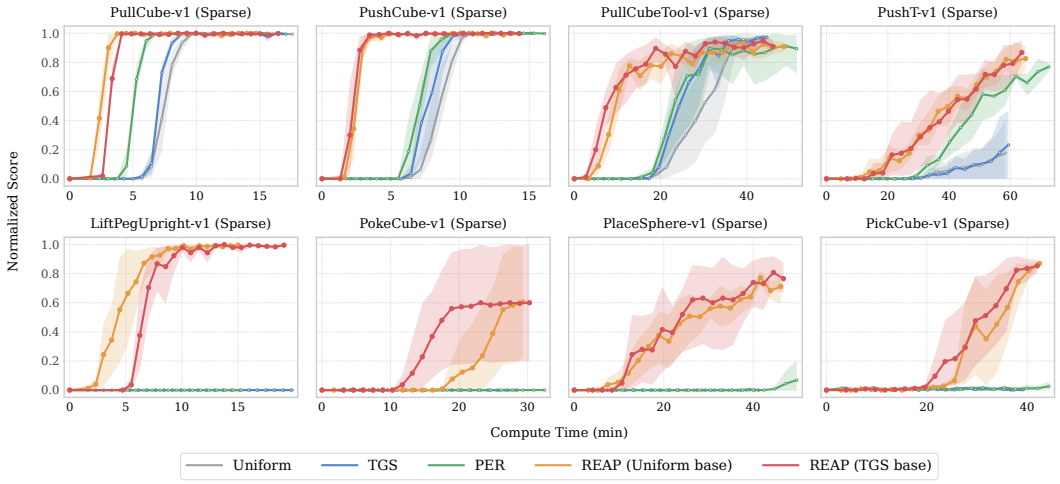

Figure 6: **REAP enables success and accelerates learning under sparse rewards.** Success rates versus wall-clock time on eight ManiSkill tasks. Both REAP variants accelerate learning on tasks where the baselines make progress (top row). They also enable task success where Uniform and TGS remain near zero throughout training (bottom row).

Separate normalization lets λ control the return-guided sampling budget independently of how rare positive-score experience is. Setting $\lambda = 1$ would restrict replay to the positive-return subset in Figure 5, whereas mixing in the base sampler keeps the rest of the buffer available for learning. Although λ can be tuned for individual tasks, our aim is to test whether a common setting remains effective across tasks and reward settings. We therefore fix $\lambda = 1/2$ throughout the main experiments and examine sensitivity to this choice in Section 4.3.

In practice, transitions enter replay with zero return-based weight and remain available through the base sampler. When an episode ends, we compute and write the scores for all its transitions together. Sampling falls back to the base distribution when no positive scores are available. The learner uses the sampled minibatch with its original rewards and standard update rules, without importance correction. Implementation details are provided in Appendix B.

4 Experiments

In this section, we evaluate REAP and examine the benefits of reallocating the replay budget. We first test whether reallocation accelerates learning and enables success on sparse-reward tasks where uniform replay struggles (Section 4.1). We then examine whether these gains extend to dense-reward tasks and another off-policy algorithm (Section 4.2). We use ablation studies to examine REAP's main design choices and understand how they affect learning performance (Section 4.3). Finally, we investigate how replay allocation affects budget efficiency and parallel scaling (Section 4.4).

Experimental setup. Our main study compares Uniform, TGS (Korniak et al., 2026), PER (Schaul et al., 2015), REAP (Uniform base), and REAP (TGS base) using FlashSAC (Kim et al., 2026) across four benchmarks (Tao et al., 2025; Team, 2026; Mittal et al., 2025; Zakka et al., 2025). Both REAP variants use $\lambda = 1/2$ in the main comparisons. Within each task, we keep the FlashSAC configuration, exploration settings, and collection and update budgets fixed across replay methods. All experiments use five independent training seeds per setting. Curves show means, and shading and error bars indicate 95% bootstrap confidence intervals. We summarize performance throughout training using the area under the learning curve (AUC) normalized by the training horizon (Hessel et al., 2018). For comparisons across multiple tasks, we first normalize performance using fixed task-specific reference values and then aggregate the curves and AUCs with equal weight for each task. The normalization reference values and details of AUC calculation are provided in Appendix C.

4.1 Learning under sparse rewards

Sparse rewards are attractive for robot learning because they specify task success without requiring carefully designed reward shaping for intermediate behavior (Andrychowicz et al., 2017). Making effective use of this simple feedback in massively parallel simulation could reduce the task-specific

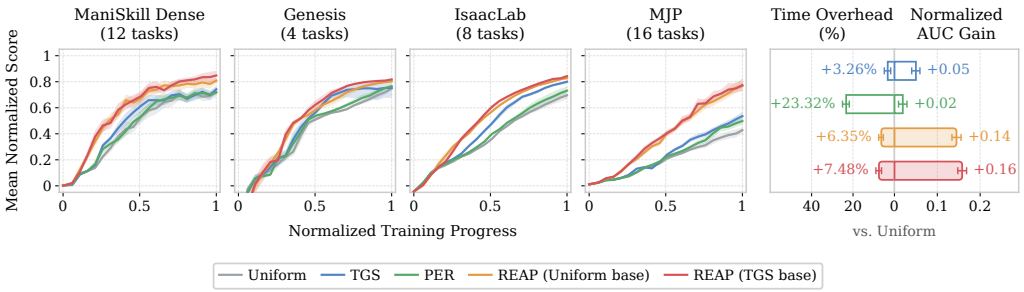

Figure 7: **REAP improves learning across benchmarks with modest overhead.** Both REAP variants improve aggregate learning performance and normalized AUC across diverse tasks, with a small increase in wall-clock training time relative to uniform replay.

reward engineering needed to train new robotic skills. We therefore first test whether REAP can turn rare successes into reliable task performance using only the original sparse rewards. We apply the same implementation throughout training on eight sparse-reward ManiSkill tasks (Tao et al., 2025) and report success rates against wall-clock time.

Figure 6 shows that REAP substantially improves learning on all eight sparse-reward tasks. • On tasks where FlashSAC with uniform replay already makes progress (top row), TGS and PER can further accelerate learning through non-uniform sampling based on recency and TD error, respectively, consistent with previous findings (Korniak et al., 2026). Even against these stronger baselines, both REAP variants reach comparable success rates in substantially less wall-clock time. These gains are consistent with the shorter discovery–mastery gap observed in our diagnostic interventions. • On harder tasks where uniform replay fails to learn (bottom row), sampling based on recency or TD error provides only limited improvement. Both REAP variants, however, achieve substantial success under the same collection and update budgets, enabling effective learning on tasks left unsolved by the other replay methods. These results support our central principle: increasing replay of rare successful experience can turn discovery into effective learning within the existing budget. REAP provides a simple implementation of this principle using the task's observed rewards.

4.2 Learning beyond sparse rewards

We next examine whether the benefits of replay reallocation extend to dense-reward settings. By scoring experience with return-to-go, REAP can account for partial progress as well as task completion. We therefore evaluate the same implementation on 40 additional tasks across four benchmarks, covering manipulation and locomotion under different reward designs.

Figure 7 shows faster learning and higher final performance with both REAP variants across all four benchmarks. Across the 40 tasks, REAP (Uniform base) improves mean normalized AUC by approximately 0.14 compared with uniform replay. TGS already improves learning through recency-based sampling, and REAP (TGS base) provides a further gain of approximately 0.11 compared with TGS. These results show that return guidance remains beneficial even when recency-based replay already improves learning, extending the allocation principle to tasks with richer reward feedback. Detailed task-level results and learning curves are provided in Appendices D.2–D.4.

We also replace SAC with TD3 (Fujimoto et al., 2018) within the FlashSAC training framework (Kim et al., 2026) and evaluate REAP on four sparse-reward and four dense-reward tasks. Both REAP variants improve normalized AUC in each setting (Figure 8), showing that the benefits of replay reallocation also extend to another representative off-policy RL algorithm.

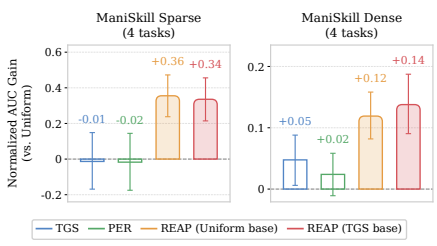

Figure 8: REAP improves learning with TD3 under both sparse and dense rewards.

REAP achieves substantial learning gains with only a modest increase in wall-clock training time: 6–8% relative to uniform replay, compared with 23% for PER (Figure 7). Because return-based scores depend on observed rewards rather than changing model estimates, REAP computes and stores each score once when an episode ends, avoiding the repeated priority evaluation used by PER.

4.3 Design ablations

Our ablations use four representative tasks spanning sparse and dense rewards and a TGS base unless otherwise stated. Detailed definitions and task-level results are provided in Appendix E.

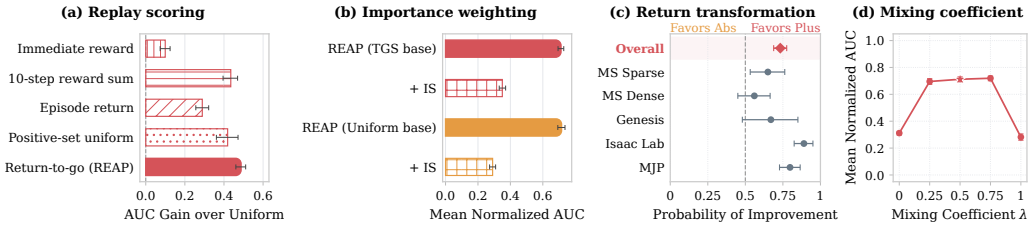

Figure 9: **Ablations of REAP.** We examine replay scoring (a), importance weighting (b), positive-part versus absolute-value scores (c), and sensitivity to the mixing coefficient λ (d). Results are aggregated across 44 tasks in panel c and four tasks in the remaining panels.

(a) We compare alternative ways to score experience for replay. Return-to-go yields the largest mean AUC gain. Ten-step reward sums also perform well, while immediate rewards and episode returns are less effective, supporting the use of subsequent rewards to score individual transitions. Uniform sampling within the positive-score set retains much of the improvement, while weighting these transitions by return-to-go yields a further gain. This suggests that distinguishing different degrees of progress among these transitions helps guide replay more effectively.

(b) We test PER-style importance weighting (Schaul et al., 2015) that moves updates toward the corresponding base distribution while keeping sampling unchanged. This reduces mean normalized AUC for both REAP variants, supporting our choice to preserve the additional emphasis introduced by return-guided sampling during updates. This choice is also consistent with TGS, which uses non-uniform replay without importance correction (Korniak et al., 2026).

(c) We compare positive-part scores $(G'_i)_+$ with absolute-value scores $|G'_i|$ across 44 tasks. Positive-part scoring achieves an aggregate probability of improvement of 0.73. Absolute-value scores also emphasize outcomes far below the reference; the advantage of positive-part scoring supports directing additional replay toward better observed outcomes rather than large deviations in either direction.

(d) The intermediate mixtures $\lambda \in \{0.25, 0.5, 0.75\}$ outperform both base-only ($\lambda = 0$) and return-only ($\lambda = 1$) sampling, supporting the use of return guidance while retaining a base sampler. Their similar performance suggests that the benefits of replay reallocation do not depend on fine-tuning λ, motivating $\lambda = 1/2$ as a common default across tasks.

4.4 Replay efficiency across budgets and scales

The previous experiments show that reallocating replay improves learning at a fixed budget. We now ask: *can better replay allocation make more effective use of the training budget as replay volume and parallel collection change?* These analyses use the same four tasks and REAP (TGS base). Our default setting is $(N, B, U) = (1{,}024, 2{,}048, 2)$, where N denotes the number of parallel environments, B the batch size, and U the number of updates per collection step. We vary these quantities while keeping the total environment-step budget fixed.

(a) ***Can REAP retain its advantage with a smaller replay budget?*** Reducing U by 50% leads to a larger drop in performance than reducing B by the same amount. In both cases, REAP is less affected than uniform replay and TGS: its mean normalized AUC decreases by 4.5% with a smaller batch and 20.6% with fewer updates. Even with half the replay budget, REAP still outperforms both baselines at their default budgets, demonstrating more effective use of the replayed experience.

(b) ***Do the benefits of replay reallocation persist with larger batches?*** FastTD3 and FlashSAC highlight the benefits of large batches in massively parallel off-policy training (Seo et al., 2025; Kim et al., 2026). Larger batches also increase replay volume, which prior work finds can reduce the advantage of recency-based sampling (Korniak et al., 2026). We therefore increase B from 2,048 to 8,192 while keeping N and U fixed. REAP maintains a clear advantage across these batch sizes: even with four times the default batch size, uniform replay and TGS remain below REAP at its

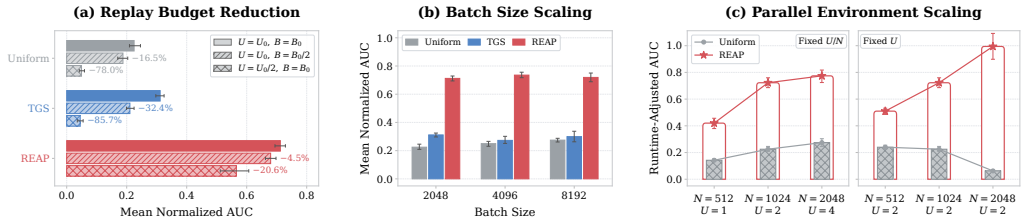

Figure 10: **Replay allocation across budgets and scales.** We examine reduced replay budgets (a), larger batches (b), and increased parallelism at fixed U/N or fixed U (c).

default setting. These results show that outcome-guided replay remains beneficial even when each update uses substantially more samples.

(c) ***Does increasing the number of parallel environments improve training efficiency?*** Increasing parallelism can speed up data collection, but whether this improves overall training efficiency also depends on how the data are used. We increase N from 512 to 2,048 at a fixed environment-step budget and batch size. Fixing U/N maintains the same replay volume (Korniak et al., 2026), whereas fixing $U = 2$ reduces replay volume as N grows. We report runtime-adjusted AUC, which divides normalized AUC by training time relative to default uniform replay on the same task. At fixed U/N, both methods improve on this measure. At fixed U, it continues to rise for REAP but falls for uniform replay at the largest scale. These results highlight the role of replay allocation in turning faster data collection into more efficient training.

5 Related Work

Non-uniform replay in RL. Effective experience replay depends on both how much stored data are reused and which transitions are selected for updates (Fedus et al., 2020; Nauman et al., 2024). Non-uniform sampling addresses the latter through TD errors or gradient information (Schaul et al., 2015; Lahire et al., 2021; Pleiss et al., 2026), preferences for recent experience (Wang et al., 2020), or equal mixtures of offline and online data (Ball et al., 2023). These two aspects are closely related: Korniak et al. (2026) show that recency-based sampling is particularly effective when each collected transition receives little replay, a setting common in massively parallel RL, while uniform replay remains competitive when data reuse is high. WarpSAC further examines age-based replay together with normalization and critic design across collection scales (Wu et al., 2026). In massively parallel RL, where large-scale interaction makes rare successful experience available for learning, we study how replay allocation helps turn these discoveries into reliable policy performance.

Learning from successful experience. A closely related line of work studies how agents can learn more effectively from past successes. Observed returns can guide the retention and sampling of useful experience (Schaul et al., 2015; Zhang et al., 2019; Tafazzol et al., 2021; Banerjee et al., 2022), while event-based replay extends this emphasis to histories leading to selected events (Kompella et al., 2023). Beyond replay selection, self-imitation turns high-return decisions or efficient successful trajectories into policy supervision (Oh et al., 2018; Jia & Chen, 2026). This idea also extends to language models, where self-generated correct solutions are reused through iterative fine-tuning or replay alongside fresh rollouts (Zelikman et al., 2022; Zhan et al., 2026). REAP has a similar motivation: to make better use of successful experience. We address the discovery–mastery gap through a simple replay implementation that uses observed rewards to guide sampling across tasks.

6 Conclusion

We identify replay allocation as a bottleneck between discovering successful behavior and learning to reproduce it in massively parallel RL. To address this gap, we introduce REAP, a simple implementation that uses observed returns to guide replay and substantially improves learning across reward settings under the same collection and update budgets. As massively parallel simulation and real-to-sim systems enable diverse interaction at scale, a key question is how to turn this interaction into scalable skill learning. Our work takes an initial step in this direction by showing how replay allocation can improve learning from collected experience.

REPRODUCIBILITY STATEMENT

We provide our implementation in the supplementary material. The appendices document experimental settings, normalization and statistical procedures, and task-level results.

AI USE STATEMENT

Generative AI tools assisted with manuscript refinement, figure preparation, code review and debugging, literature search, and feedback on experiments and results. The authors made the final research decisions and take responsibility for the manuscript, code, and results.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Genesis Authors. Genesis: A generative and universal physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/genesis-world.

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pp. 1577–1594. PMLR, 2023.

Chayan Banerjee, Zhiyong Chen, and Nasimul Noman. Improved soft actor-critic: Mixing prioritized off-policy samples with on-policy experiences. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3121–3129, 2022.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International conference on machine learning*, pp. 3061–3071. PMLR, 2020.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. Pmlr, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Yinsen Jia and Boyuan Chen. Temporal self-imitation learning. *arXiv preprint arXiv:2606.19752*, 2026.

Donghu Kim, Youngdo Lee, Minho Park, Kinam Kim, I Made Aswin Nahendra, Takuma Seno, Sehee Min, Daniel Palenicek, Florian Vogt, Danica Kragic, Jan Peters, Jaegul Choo, and Hojoon Lee. Flashsac: Fast and stable off-policy reinforcement learning for high-dimensional robot control. *arXiv preprint arXiv:2604.04539*, 2026.

Varun Raj Kompella, Thomas Walsh, Samuel Barrett, Peter R. Wurman, and Peter Stone. Event tables for efficient experience replay. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=XejzjAjKjv.

Michal Korniak, Mikołaj Czarnecki, Yarden As, Piotr Miłoś, Pieter Abbeel, and Michal Nauman. When does non-uniform replay matter in reinforcement learning? *arXiv preprint arXiv:2605.10236*, 2026.

Thibault Lahire, Matthieu Geist, and Emmanuel Rachelson. Large batch experience replay. *arXiv preprint arXiv:2110.01528*, 2021.

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 *International Conference on Machine Learning*, pp. 19440–19459. PMLR, 2023.

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. *arXiv preprint arXiv:2108.10470*, 2021.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, Milad Rakhsha, Alain Denzler, Eric Heiden, Ales Borovicka, Ossama Ahmed, Iretiayo Akinola, Abrar Anwar, Mark T. Carlson, Ji Yuan Feng, Animesh Garg, Renato Gasoto, Lionel Gulich, Yijie Guo, M. Gussert, Alex Hansen, Mihir Kulkarni, Chenran Li, Wei Liu, Viktor Makoviychuk, Grzegorz Malczyk, Hammad Mazhar, Masoud Moghani, Adithyavairavan Murali, Michael Noseworthy, Alexander Poddubny, Nathan Ratliff, Welf Rehberg, Clemens Schwarke, Ritvik Singh, James Latham Smith, Bingjie Tang, Ruchik Thaker, Matthew Trepte, Karl Van Wyk, Fangzhou Yu, Alex Millane, Vikram Ramasamy, Remo Steiner, Sangeeta Subramanian, Clemens Volk, CY Chen, Neel Jawale, Ashwin Varghese Kuruttukulam, Michael A. Lin, Ajay Mandlekar, Karsten Patzwaldt, John Welsh, Huihua Zhao, Fatima Anes, Jean-Francois Lafleche, Nicolas Moënne-Loccoz, Soowan Park, Rob Stepinski, Dirk Van Gelder, Chris Amevor, Jan Carius, Jumyung Chang, Anka He Chen, Pablo de Heras Ciechomski, Gilles Daviet, Mohammad Mohajerani, Julia von Muralt, Viktor Reutskyy, Michael Sauter, Simon Schirm, Eric L. Shi, Pierre Terdiman, Kenny Vilella, Tobias Widmer, Gordon Yeoman, Tiffany Chen, Sergey Grizan, Cathy Li, Lotus Li, Connor Smith, Rafael Wiltz, Kostas Alexis, Yan Chang, David Chu, Linxi "Jim" Fan, Farbod Farshidian, Ankur Handa, Spencer Huang, Marco Hutter, Yashraj Narang, Soha Pouya, Shiwei Sheng, Yuke Zhu, Miles Macklin, Adam Moravanszky, Philipp Reist, Yunrong Guo, David Hoeller, and Gavriel State. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025. URL https://arxiv.org/abs/2511.04831.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control. *Advances in neural information processing systems*, 37:113038–113071, 2024.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3878–3887. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/oh18b.html.

Leonard S Pleiss, Tobias Sutter, and Maximilian Schiffer. Reliability-adjusted prioritized experience replay. In *International Conference on Learning Representations*, volume 2026, pp. 59206–59232, 2026.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on robot learning*, pp. 91–100. PMLR, 2022.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control. *arXiv preprint arXiv:2505.22642*, 2025.

Saeed Tafazzol, Erfan Fathi, Mahdi Rezaei, and Ehsan Asali. Curious exploration and return-based memory restoration for deep reinforcement learning. *arXiv preprint arXiv:2105.00499*, 2021.

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.

Genesis AI Team. The role of simulation in scalable robotics, genesis world 1.0, and the path forward. *Genesis AI Blog*, May 2026. URL https://www.genesis.ai/blog/the-role-of-simulation-in-scalable-robotics-genesis-world-10-and-the-path-forward.

Che Wang, Yanqiu Wu, Quan Vuong, and Keith Ross. Striving for simplicity and performance in off-policy drl: Output normalization and non-uniform sampling. In *International Conference on Machine Learning*, pp. 10070–10080. PMLR, 2020.

Zihao Wu, Hongyao Tang, Yi Ma, Huizhong Song, Pengyi Li, Yifu Yuan, Fei Ni, Jinyi Liu, Wei Wei, Jianrong Wang, et al. Warpsac: Towards the pinnacle of scalable off-policy rl by rethinking exploration and exploitation. *arXiv preprint arXiv:2608.24479*, 2026.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Runzhe Zhan, Yafu Li, Zhi Wang, Xiaoye Qu, Dongrui Liu, Jing Shao, Derek Wong, and Yu Cheng. Exgrpo: Learning to reason from experience. In *International Conference on Learning Representations*, volume 2026, pp. 12810–12846, 2026.

Zhizheng Zhang, Jiale Chen, Zhibo Chen, and Weiping Li. Asynchronous episodic deep deterministic policy gradient: Toward continuous control in computationally complex environments. *IEEE transactions on cybernetics*, 51(2):604–613, 2019.

APPENDIX

This appendix provides diagnostic and intervention protocols (Appendix A), implementation details (Appendix B), experimental settings (Appendix C), individual-task results (Appendix D), and ablation and budget settings (Appendix E).

A DIAGNOSTICS AND INTERVENTIONS

Random initialization. For Figure 2, we evaluate ten independently initialized frozen policies on each sparse-reward task using the training action-sampling rule. Each policy runs 81,920 complete episodes: 40 episodes in each of 2,048 environments. Success means meeting the task criterion at least once during an episode. Panel B illustrates independent attempts with fixed per-episode success probability $p = 10^{-4}$.

Training diagnostics. Each frozen checkpoint is evaluated deterministically and stochastically over 1,024 episodes per mode, with matched initial-state distributions and episode limits. Evaluation neither updates the policy nor adds data to replay. The buffer statistic counts successful episodes represented in replay; the intervention samples retained transitions from completed successful episodes. Buffer values average the nearest logged record and its available immediate neighbors at each checkpoint.

Online intervention. At each switch checkpoint, we restore the saved policy, critics, target networks, optimizer, and replay buffer. Let $\mathcal{D}$ denote the replay buffer and $\mathcal{H} \subseteq \mathcal{D}$ the retained transitions from completed successful episodes. When $\mathcal{H}$ is nonempty, we sample transition $i \in \mathcal{D}$ with probability

$$p_i^{\text{int}} = \frac{1}{2|\mathcal{D}|} + \frac{\mathbf{1}\{i \in \mathcal{H}\}}{2|\mathcal{H}|}. \tag{3}$$

This draws half of replay uniformly from successful episodes and half uniformly from the full buffer. If $\mathcal{H}$ is empty, we use uniform replay. Online training continues with the original collection and update settings. Each switch setting uses five training runs, and evaluation success is zero at the switch checkpoint.

Task	Reward setting	Switch times (M steps)
PullCube-v1	Sparse	1, 2, 3, 4, 5
PlaceSphere-v1	Dense	1, 3, 5, 7, 9
LiftPegUpright-v1	Sparse	2, 4, 6, 8, 10

Switches occur at collection-batch boundaries. Curves include the environment steps collected before switching and end at the last recorded evaluation..

B REAP IMPLEMENTATION

Listing 1 summarizes the core scoring and sampling logic. The input `rewards` contains the original single-step rewards from one completed episode in temporal order, and `r_ref` is fixed after warmup. Both termination and time-limit truncation end the return calculation, without value bootstrapping.

Listing 1: Core scoring and sampling logic of REAP.

```
import numpy as np

def score_episode(rewards, gamma, r_ref):
    g = 0.0
    weights = np.empty(len(rewards))
    for t in range(len(rewards) - 1, -1, -1):
        g = rewards[t] - r_ref + gamma * g
        weights[t] = max(g, 0.0)
    return weights

def sample_reap(tree, sample_base, batch_size, lam=0.5):
    use_return = np.zeros(batch_size, dtype=bool)
    if tree.total > 0:
        use_return = np.random.random(batch_size) < lam
    indices = np.empty(batch_size, dtype=np.int64)
    n_return = int(use_return.sum())
    if n_return:
        indices[use_return] = tree.sample(n_return)
    if n_return < batch_size:
        indices[~use_return] = sample_base(batch_size - n_return)
    return indices
```

Completed-episode scores are written once to a sum-tree. New or overwritten buffer slots receive zero return weight. The tree samples in proportion to these weights, while `sample_base` samples from uniform replay or TGS; both sample with replacement. Parallel episode tracking and buffer storage are handled by the full implementation.

C EXPERIMENTAL SETTINGS

The tables below list the evaluation metrics, training budgets, and fixed normalization references used in our experiments. For run r on task e, we normalize the evaluation score $Y_{e,r}(s)$ at environment step s as

$$Z_{e,r}(s) = \frac{Y_{e,r}(s) - L_e}{H_e - L_e}. \tag{4}$$

Success rates use $L_e = 0$ and $H_e = 1$ and are therefore unchanged. Scores are not clipped to $[0, 1]$, and normalization does not affect training rewards.

Let $s_0 < \cdots < s_K$ be the evaluation steps within the common observed window for a task. We compute AUC using the trapezoidal rule and normalize by the window length:

$$\mathrm{AUC}_{e,r} = \frac{1}{s_K - s_0} \sum_{k=1}^{K} \frac{Z_{e,r}(s_{k-1}) + Z_{e,r}(s_k)}{2} (s_k - s_{k-1}). \tag{5}$$

We average runs within each task and weight tasks equally. Aggregate learning curves use normalized training progress when task budgets differ. Main comparisons use five independent training runs per setting. The 95% bootstrap confidence intervals use 10,000 resamples of complete runs within each task and method, keeping the task set fixed.

Table 1: **ManiSkill** (Tao et al., 2025). Evaluation metrics, normalization references, and training budgets in millions of environment steps.

Task	Metric	L_e	H_e	Steps (M)
Sparse rewards				
PullCube-v1	Success rate	0	1	20
PushCube-v1	Success rate	0	1	20
PullCubeTool-v1	Success rate	0	1	60
PushT-v1	Success rate	0	1	100
LiftPegUpright-v1	Success rate	0	1	20
PokeCube-v1	Success rate	0	1	40
PlaceSphere-v1	Success rate	0	1	60
PickCube-v1	Success rate	0	1	60
Dense rewards				
LiftPegUpright-v1	Success rate	0	1	20
PegInsertionSide-v1	Success rate	0	1	20
PickCubeSO100-v1	Success rate	0	1	20
PickSingleYCB-v1	Success rate	0	1	20
PlaceSphere-v1	Success rate	0	1	20
PullCube-v1	Success rate	0	1	20
StackCube-v1	Success rate	0	1	20
UnitreeG1PlaceAppleInBowl-v1	Success rate	0	1	40
MS-HumanoidStand-v1	Episode return	0	1000	40
MS-HumanoidWalk-v1	Episode return	500	1000	40
MS-HumanoidRun-v1	Episode return	500	5000	40
MS-AntRun-v1	Episode return	0	1000	40

Table 2: **Genesis** (Team, 2026). Normalization references and training budgets. All tasks are evaluated using episode return.

Task	L_e	H_e	Steps (M)
go2-walk_easy	0	30	10
go2-walk	-5	25	20
go2-backflip	-5	25	10
panda-grasp	0	3	20

Table 3: **Isaac Lab** (Mittal et al., 2025). Normalization references and training budgets. All tasks are evaluated using episode return.

Task	L_e	H_e	Steps (M)
Isaac-Velocity-Flat-Digit-v0	0	40	20
Isaac-Ant-Direct-v0	0	20000	20
Isaac-Velocity-Flat-Unitree-Go1-v0	0	40	20
Isaac-Velocity-Flat-G1-v0	0	40	20
Isaac-Velocity-Flat-Cassie-v0	0	80	20
Isaac-Lift-Cube-Franka-v0	0	160	20
Isaac-Repose-Cube-Allegro-Direct-v0	0	6000	40
Isaac-Repose-Cube-Shadow-Direct-v0	0	10000	40

Table 4: **MuJoCo Playground** (Zakka et al., 2025). Normalization references and training budgets. All tasks are evaluated using episode return.

Task	L_e	H_e	Steps (M)
AcrobotSwingupSparse	0	20	40
FingerTurnHard	0	1000	20
HopperHop	0	400	20
HopperStand	0	1000	20
HumanoidStand	0	1000	20
HumanoidWalk	0	1000	20
HumanoidRun	0	1000	40
PendulumSwingup	0	800	40
ApolloJoystickFlatTerrain	0	30	20
BarkourJoystick	0	50	20
G1JoystickRoughTerrain	-5	35	20
T1JoystickFlatTerrain	0	50	20
G1JoystickFlatTerrain	-5	35	20
T1JoystickRoughTerrain	0	40	20
AlohaHandOver	0	10	40
PandaPickCube	0	12000	40

D INDIVIDUAL-TASK RESULTS

The following figures report the five-method FlashSAC comparison for all 48 tasks. Each curve averages five training runs, with 95% bootstrap confidence intervals. Task metrics and budgets are listed in Appendix C.

D.1 SPARSE-REWARD MANISKILL

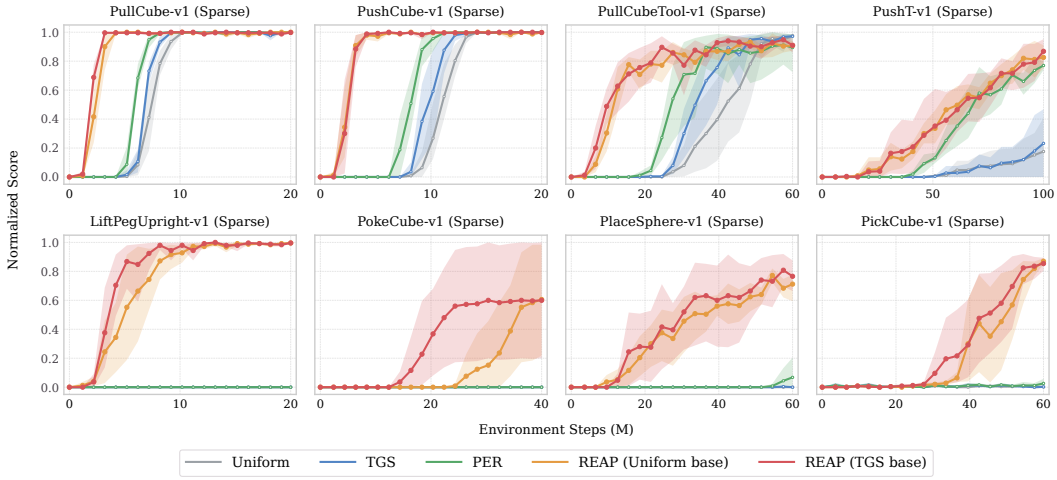

Figure 11: Success rates against environment steps on eight sparse-reward tasks. Figure 6 shows the same results against training time.

D.2 DENSE-REWARD MANISKILL

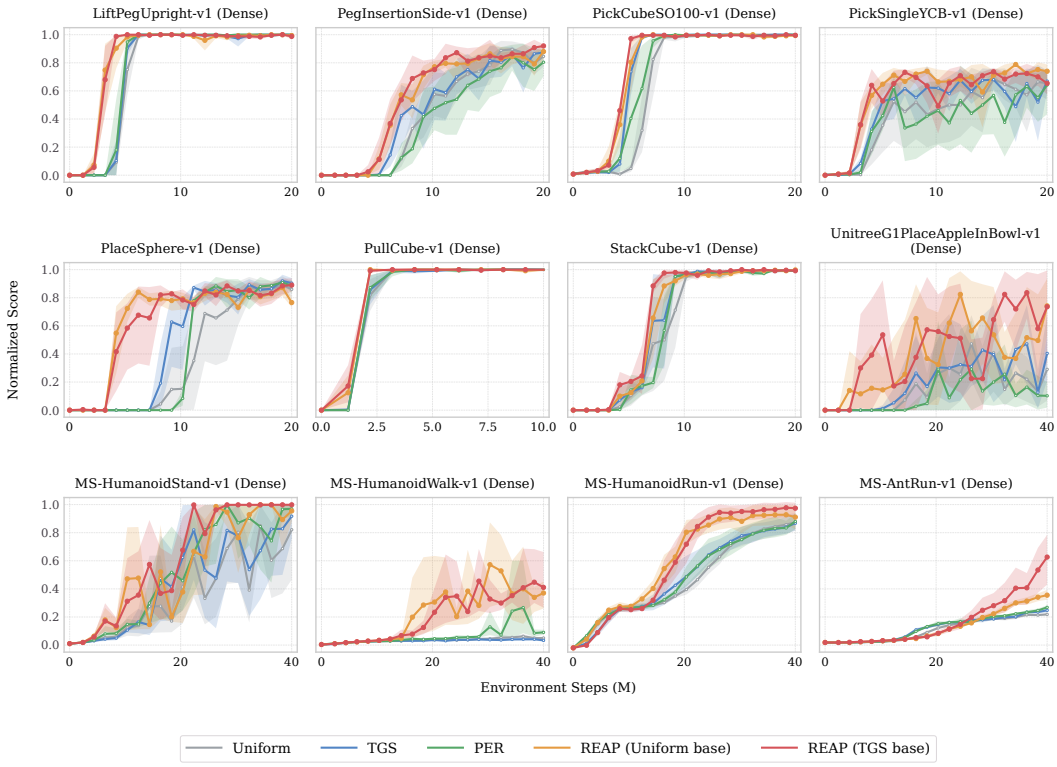

Figure 12: Learning curves on twelve dense-reward tasks. The first two rows show success rates; the third shows normalized returns.

D.3 GENESIS AND ISAAC LAB

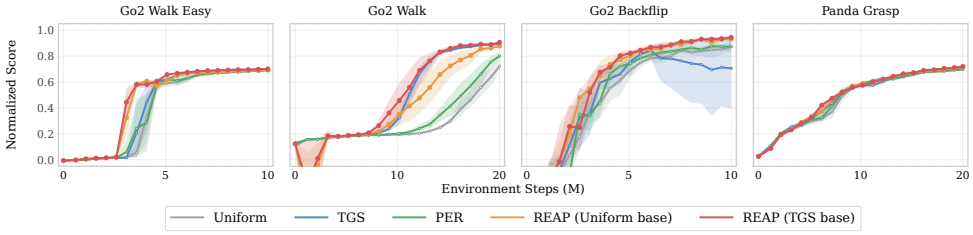

Figure 13: Normalized returns on four Genesis tasks.

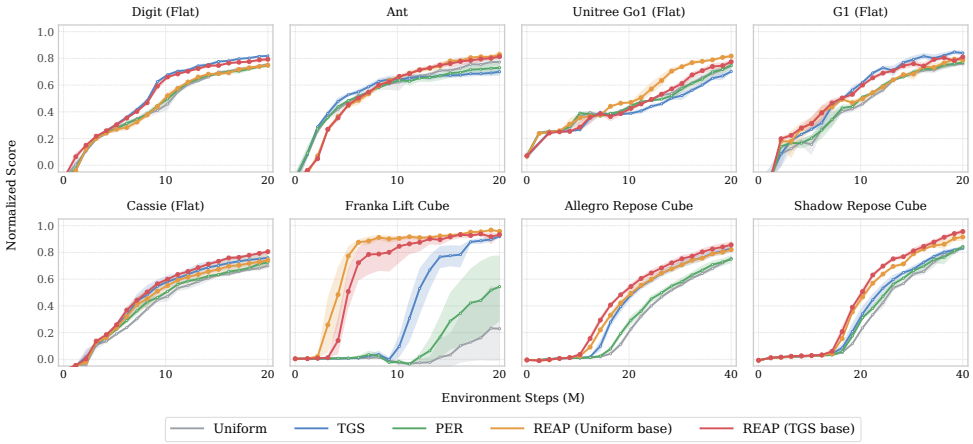

Figure 14: Normalized returns on eight Isaac Lab tasks.

D.4 MUJOCO PLAYGROUND



Figure 15: Normalized returns on sixteen MuJoCo Playground tasks.

E ABLATION AND BUDGET SETTINGS

The four-task studies use sparse-reward LiftPegUpright-v1 and PushCube-v1, Isaac-Lift-Cube-Franka-v0, and T1JoystickRoughTerrain. Unless varied below, the default settings are $N = 1{,}024$, $B = 2{,}048$, $U = 2$, $20{,}000{,}768$ environment steps including warmup, and a replay capacity of $10{,}240{,}000$ transitions. REAP uses a TGS base and $\lambda = 1/2$. Each setting uses five training runs; normalization and statistical procedures follow Appendix C.

E.1 SCORING AND MIXING

Scoring variants. We compare five replay scores while keeping the base distribution, mixing coefficient, reward reference, discount, and score-availability time unchanged. Let s_i denote the nonnegative score used for proportional sampling in the return branch, and let $a(i)$ denote the first transition of the completed episode or segment containing i.

- **Immediate reward:**
$$s_i = (r_i - r_{\mathrm{ref}})_+.$$
- **Ten-step reward sum:**
$$s_i = \left[\sum_{k=0}^{\min(10, L_i)-1} \gamma^k (r_{i+k} - r_{\mathrm{ref}}) \right]_+ .$$
- **Episode return:** All transitions in an episode or segment receive its initial centered return:
$$s_i = (G'_{a(i)})_+.$$
- **Positive-set uniform:** Transitions with positive centered return receive equal weight:
$$s_i = \mathbf{1}\{G'_i > 0\}.$$
- **Return-to-go (REAP):**
$$s_i = (G'_i)_+.$$

All scores become available at episode or segment completion. Reward sums stop at the observed episode boundary without value bootstrapping.

Importance weighting. We implement PER-style importance weighting (Schaul et al., 2015) following the public implementation accompanying Korniak et al. (2026).[1] We keep sampling unchanged and adapt the weights to move updates toward the corresponding base distribution, uniform replay or TGS. Table 5 reports the task-level results.

Mixing coefficient. We test $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$, reusing the TGS and default REAP runs for $\lambda = 0$ and $\lambda = 1/2$, respectively. When no positive scores are available, all settings use the base sampler. Figure 16 shows the task-level learning curves.

For Franka, alternative scoring rules and mixtures with $\lambda \in \{0.25, 0.75, 1\}$ use 4,096 evaluation episodes. The reused TGS, default REAP, and importance-weighting runs use 1,024.

Table 5: **Task-level scoring and importance-weighting results.** Mean normalized AUC over five runs. The first five rows use a TGS base; IS targets the base named in its row.

Replay rule	LiftPeg	PushCube	Franka	T1 Rough
Immediate reward	0.002	0.768	0.009	0.519
10-step reward sum	0.603	0.869	0.639	0.527
Episode return	0.688	0.854	0.056	0.458
Positive-set uniform	0.692	0.873	0.512	0.496
REAP (TGS base)	0.791	0.871	0.654	0.533
REAP (TGS base) + IS	0.001	0.572	0.362	0.466
REAP (Uniform base)	0.724	0.872	0.736	0.530
REAP (Uniform base) + IS	0.003	0.473	0.241	0.446

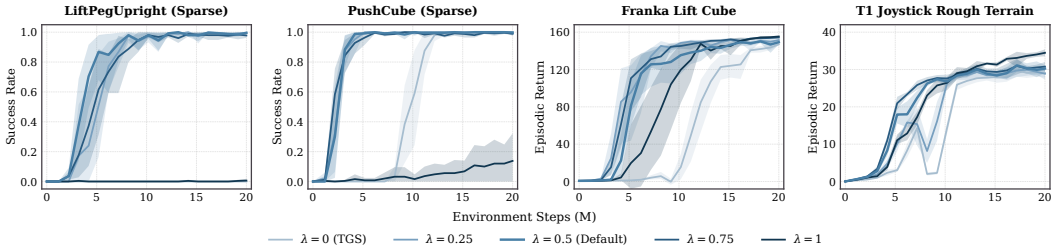

Figure 16: **Mixing coefficient across tasks.** All settings use a TGS base. Curves show success rates for sparse tasks and episodic returns for dense tasks, with 95% bootstrap confidence intervals over five runs.

[1] https://github.com/M-Korniak/when-does-replay-matter

E.2 ADDITIONAL COMPARISONS

Score transformation. The comparison of $(G_i')_+$ and $|G_i'|$ uses a TGS base, $\lambda = 1/2$, and five runs per setting on 44 tasks: the main task set excluding dense-reward PegInsertionSide-v1, PickCubeSO100-v1, PullCube-v1, and StackCube-v1. MJP HumanoidStand and HumanoidWalk use 20M-step analysis windows. Probability of improvement compares every pair of runs within a task by AUC, assigning one for a positive-part win, zero for a loss, and one half for a tie, then averages equally across tasks. Bootstrap resampling operates on runs, not run pairs. Figure 17 shows the individual-task curves.

TD3. The TD3 comparison uses all five replay methods with matched training settings. Sparse tasks are PullCube, PushCube, PullCubeTool, and PushT; dense tasks are PlaceSphere, LiftPegUpright, PegInsertionSide, and MS-HumanoidWalk, all with the `-v1` suffix. Analysis windows extend to approximately 20M environment steps, except MS-HumanoidWalk (40M), PullCubeTool (60M), and PushT (100M). MS-HumanoidWalk reports episodic return; the other tasks report success rate. Both REAP variants use $\lambda = 1/2$ and five runs per task.

Replay budgets and parallel collection. Reduced budgets set either $B = 1{,}024$ or $U = 1$. The batch sweep uses $B \in \{2{,}048, 4{,}096, 8{,}192\}$ at fixed N and U. These comparisons include uniform replay, TGS, and REAP. Parallel scaling compares uniform replay and REAP at $(N, U) = (512, 1), (1{,}024, 2), (2{,}048, 4)$, or at the same N values with $U = 2$. Both schedules retain the default batch size, environment-step budget, and buffer capacity. Runtime-adjusted AUC uses default uniform replay on the same task as the time reference (Appendix C).

E.3 TASK-LEVEL SCORE-TRANSFORMATION RESULTS

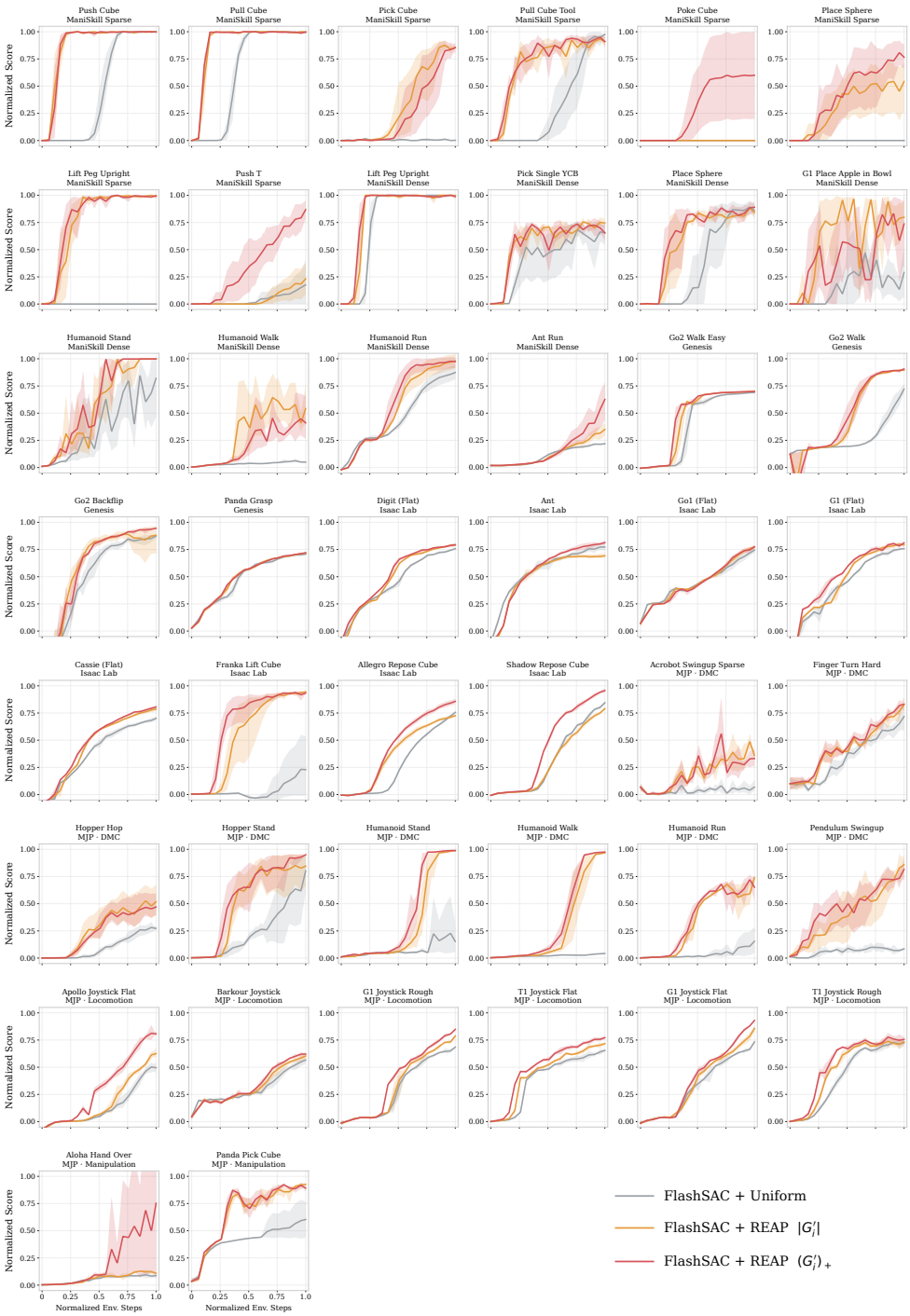

Figure 17: **Positive-part and absolute-value scores across 44 tasks.** Both score variants use a TGS base and $\lambda = 1/2$. Curves show mean normalized scores and 95% bootstrap confidence intervals over five runs. Environment steps are divided by each task's analysis budget.